Supervised Learning with Sonar Data

Mohammad Ali Mirzaie

2025-10-03

Hello everyone, I am `Mohammad Ali Mirzaie`. I earned my bachelor's degree in Statistic and now I am studying for a master's degree in data science. I am interested in the field of machine learning.

In this study, we aim to use `Sonar` data from the `mlbench` package and predict new observations using `Supervised Learning` methods. First, let's briefly discuss the data. Sonar is naval tool used for underwater detection. the Sonar system has 60 frequency bands and is used to predict whether an object is `Rock (R)` or `Metal (M)`. Also Sonar stands for `Sound Navigation & Ranging`. This tool is sometimes used for detecting the mines underwater. Before analyzing the data, we need to load the required packages.

```
# install.packages('caret',dep = T)
# install.packages('mlbench',dep = T)
library(caret)
data(Sonar, package = 'mlbench')
names(Sonar)
```

```
##  [1] "V1"    "V2"    "V3"    "V4"    "V5"    "V6"    "V7"    "V8"    "V9"
## [10] "V10"   "V11"   "V12"   "V13"   "V14"   "V15"   "V16"   "V17"   "V18"
## [19] "V19"   "V20"   "V21"   "V22"   "V23"   "V24"   "V25"   "V26"   "V27"
## [28] "V28"   "V29"   "V30"   "V31"   "V32"   "V33"   "V34"   "V35"   "V36"
## [37] "V37"   "V38"   "V39"   "V40"   "V41"   "V42"   "V43"   "V44"   "V45"
## [46] "V46"   "V47"   "V48"   "V49"   "V50"   "V51"   "V52"   "V53"   "V54"
## [55] "V55"   "V56"   "V57"   "V58"   "V59"   "V60"   "Class"
```

```
dim(Sonar)
```

```
## [1] 208  61
```

```
sum(is.na(Sonar))
```

```
## [1] 0
```

We have 208 observations and 61 variables. 60 variables for sound frequencies (predictors) and one variable for class (binary outcome). Fortunately, there is no missing value in data.

```
table(Sonar$Class)
```

```
##
##   M   R
## 111  97
```

```
prop.table(table(Sonar$Class))
```

```
##
##         M         R
## 0.5336538 0.4663462
```

Now we splitting the data with train and test. We propose 80% for train and 20% for test. also we use `set.seed(1)` for random seed of data generation.

```
set.seed(1)
t = sample(208, 208*0.8)
train = Sonar[t,]
test = Sonar[-t,]
table('train' = train$Class)
```

```
## train
##  M  R
## 91 75
```

```
table('test' = test$Class)
```

```
## test
##  M  R
## 20 22
```

After that, we need to train the data with corresponding model. We propose the following models:

`PLS: Partial Least Squares`,

`RDA: Reguralised Discriminant Analysis`,

`LDA: Linear Discriminant Analisys`,

`QDA: Quadratic Discriminant Analisys`,

`NB: Naive Bayes`.

The train control parameters for this process are:

```
ctrl = trainControl(
  method = 'repeatedcv',
  number = 10,
  repeats = 5,
  classProbs = T,
  summaryFunction = twoClassSummary,
  verboseIter = F,
  savePredictions = T
)
```

We set `Repeated Cross Validation` with 10 folds and 5 repeats. Also we set `two class summary` function.

Now we are ready for modeling.
PLS:

```
pls.fit = train(
  Class ~ .,
  data = train,
  method = 'pls',
  preProcess = c('center','scale'),
  tuneGrid = expand.grid(ncomp = 1:15),
  trControl = ctrl,
  metric = 'ROC'
)
```

RDA:

```
rda.fit = train(
  Class ~ .,
  data = train,
  method = 'rda',
```

```
  preProcess = c('center','scale'),
  tuneGrid = expand.grid(
    gamma = seq(0, 1, 0.2), lambda = seq(0, 1, 0.2)
    ),
  trControl = ctrl,
  metric = 'ROC'
)
```

LDA:

```
lda.fit = train(
  Class ~ .,
  data = train,
  method = 'lda',
  preProcess = c('center','scale'),
  trControl = ctrl,
  metric = 'ROC'
)
```

QDA:

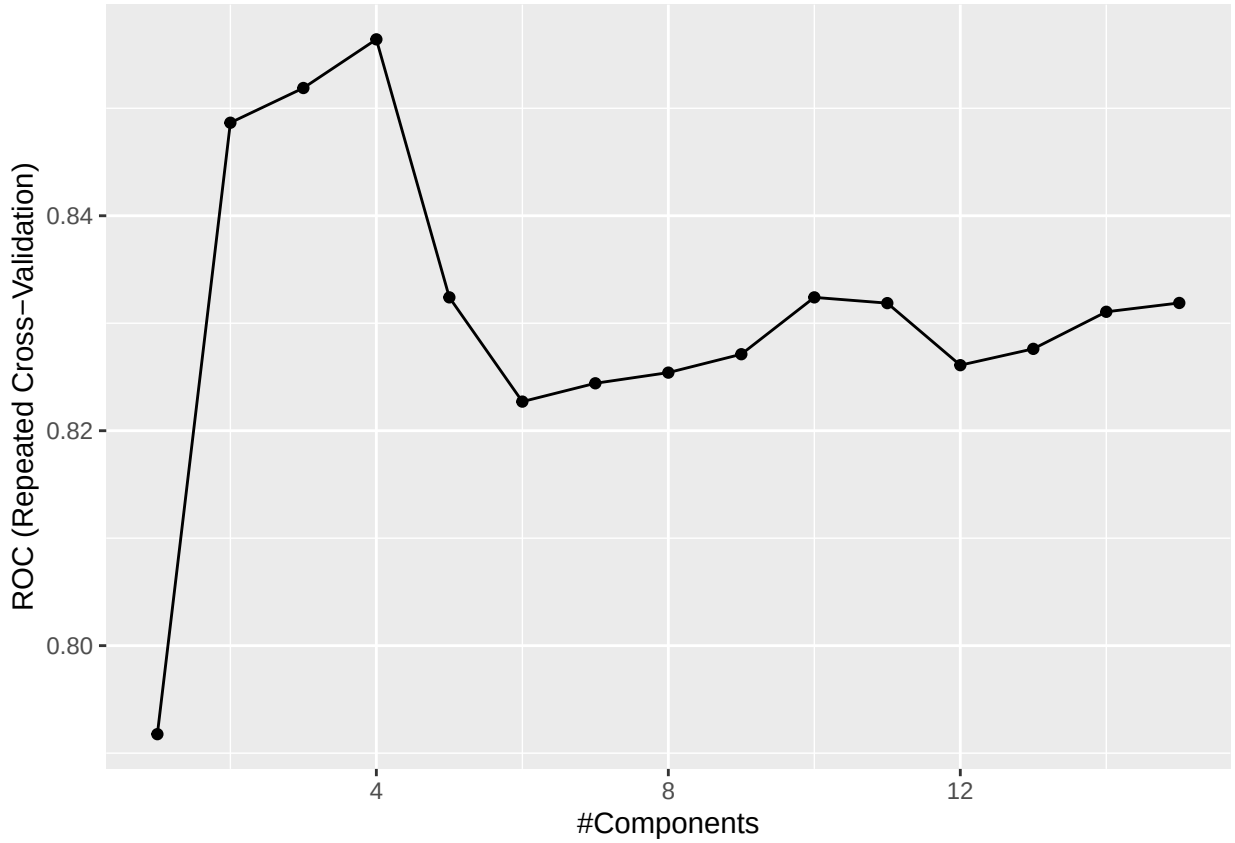
```
qda.fit = train(
  Class ~ .,
  data = train,
  method = 'qda',
  preProcess = c('center','scale'),
  trControl = ctrl,
  metric = 'ROC'
)
```

NB:

```
nb.fit = train(
  Class ~ .,
  data = train,
  method = 'nb',
  preProcess = c('center','scale'),
  tuneGrid = expand.grid(
    fL = c(0, 0.5, 1),
    usekernel = c(T,F),
    adjust = c(0.5, 1, 1.5)
    ),
  trControl = ctrl,
  metric = 'ROC'
)
```

In addition, our goal is to observe the accuracy of the models. For this manner we use corresponding `accuracy plot`, `summary`, and `summary of confussion matrix`:
PLS:

```
ggplot(pls.fit)
```




```
pls.fit
```

```
## Partial Least Squares
##
## 166 samples
##  60 predictor
##   2 classes: 'M', 'R'
##
## Pre-processing: centered (60), scaled (60)
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 149, 150, 150, 149, 149, 148, ...
## Resampling results across tuning parameters:
##
##   ncomp  ROC        Sens       Spec
##    1     0.7917659  0.7357778  0.6642857
##    2     0.8486548  0.7635556  0.7421429
##    3     0.8518849  0.7562222  0.7557143
##    4     0.8564127  0.7646667  0.7796429
##    5     0.8324048  0.7695556  0.7760714
##    6     0.8227063  0.7364444  0.7821429
##    7     0.8244127  0.7337778  0.7575000
##    8     0.8254048  0.7624444  0.7510714
##    9     0.8271151  0.7711111  0.7564286
##   10     0.8324048  0.7820000  0.7614286
##   11     0.8318690  0.7797778  0.7571429
##   12     0.8260952  0.7797778  0.7560714
```

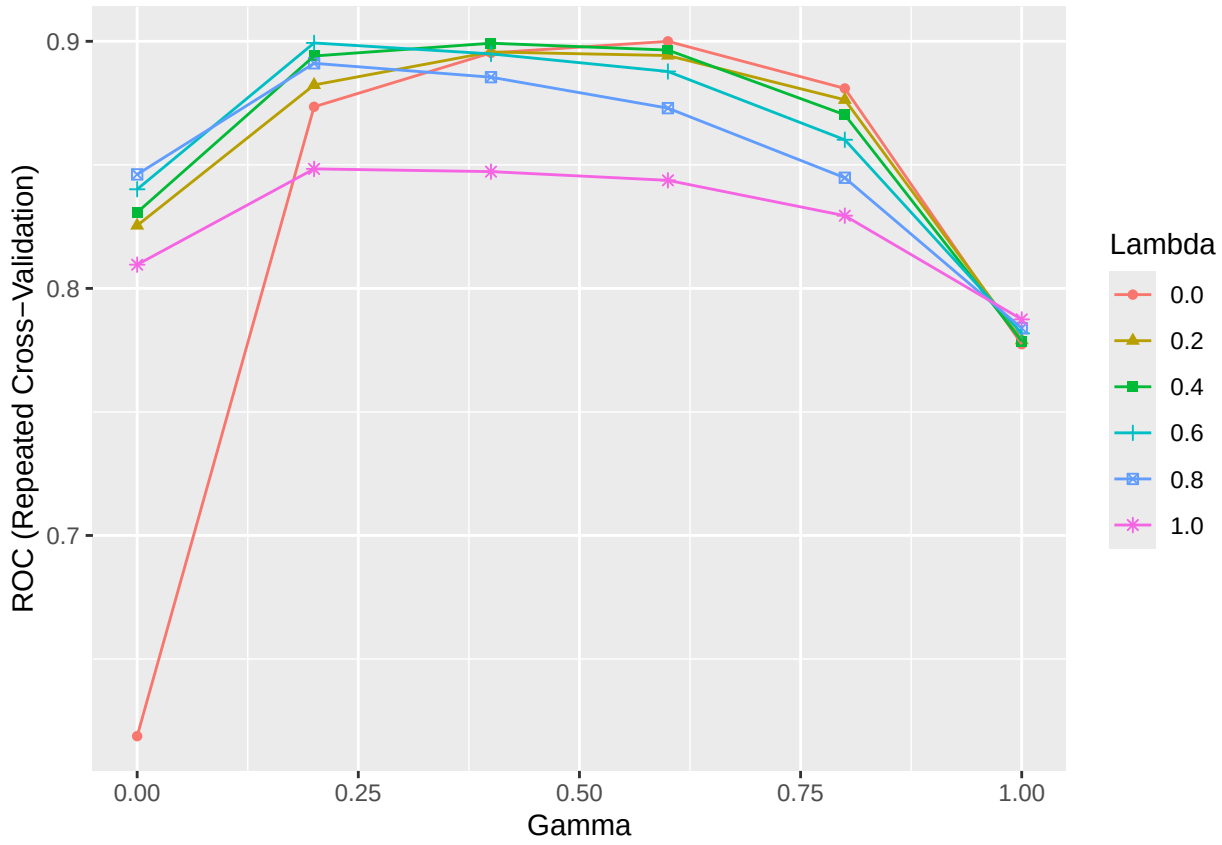
```
##   13     0.8276151  0.7822222  0.7528571
##   14     0.8310635  0.7755556  0.7667857
##   15     0.8318849  0.7802222  0.7667857
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was ncomp = 4.
```

```
pls.pre = predict(pls.fit, test)
confusionMatrix(pls.pre, test$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  M  R
##          M 18  8
##          R  2 14
##
##                Accuracy : 0.7619
##                  95% CI : (0.6055, 0.8795)
##     No Information Rate : 0.5238
##     P-Value [Acc > NIR] : 0.00133
##
##                   Kappa : 0.5291
##
##  Mcnemar's Test P-Value : 0.11385
##
##             Sensitivity : 0.9000
##             Specificity : 0.6364
##          Pos Pred Value : 0.6923
##          Neg Pred Value : 0.8750
##              Prevalence : 0.4762
##          Detection Rate : 0.4286
##    Detection Prevalence : 0.6190
##       Balanced Accuracy : 0.7682
##
##        'Positive' Class : M
##
```

RDA:

```
ggplot(rda.fit)
```




```
rda.fit
```

```
## Regularized Discriminant Analysis
##
## 166 samples
##  60 predictor
##   2 classes: 'M', 'R'
##
## Pre-processing: centered (60), scaled (60)
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 149, 150, 150, 148, 149, 149, ...
## Resampling results across tuning parameters:
##
##   gamma  lambda  ROC        Sens       Spec
##   0.0    0.0     0.6187791  0.9580000  0.2571429
##   0.0    0.2     0.8254325  0.8206667  0.7271429
##   0.0    0.4     0.8307579  0.8008889  0.7646429
##   0.0    0.6     0.8401111  0.7893333  0.7835714
##   0.0    0.8     0.8461270  0.7868889  0.7792857
##   0.0    1.0     0.8095913  0.7902222  0.7321429
##   0.2    0.0     0.8734881  0.7980000  0.7632143
##   0.2    0.2     0.8823333  0.7715556  0.8214286
##   0.2    0.4     0.8940556  0.7628889  0.8542857
##   0.2    0.6     0.8993254  0.7780000  0.8507143
##   0.2    0.8     0.8910952  0.8020000  0.8157143
##   0.2    1.0     0.8483532  0.7617778  0.7817857
##   0.4    0.0     0.8952937  0.7448889  0.8532143
```

```
##   0.4    0.2     0.8956786  0.7142222  0.8721429
##   0.4    0.4     0.8992183  0.7297778  0.8607143
##   0.4    0.6     0.8949087  0.7384444  0.8578571
##   0.4    0.8     0.8854722  0.7735556  0.8275000
##   0.4    1.0     0.8472540  0.7773333  0.7617857
##   0.6    0.0     0.8999802  0.7013333  0.8760714
##   0.6    0.2     0.8942381  0.6966667  0.8792857
##   0.6    0.4     0.8964444  0.7055556  0.8653571
##   0.6    0.6     0.8877857  0.7146667  0.8492857
##   0.6    0.8     0.8729524  0.7513333  0.8092857
##   0.6    1.0     0.8437222  0.7842222  0.7817857
##   0.8    0.0     0.8809325  0.6440000  0.8635714
##   0.8    0.2     0.8763214  0.6482222  0.8496429
##   0.8    0.4     0.8702619  0.6702222  0.8253571
##   0.8    0.6     0.8601508  0.6922222  0.7978571
##   0.8    0.8     0.8447659  0.7406667  0.7700000
##   0.8    1.0     0.8294206  0.7820000  0.7575000
##   1.0    0.0     0.7774048  0.6222222  0.6996429
##   1.0    0.2     0.7790317  0.6264444  0.6996429
##   1.0    0.4     0.7786389  0.6282222  0.6939286
##   1.0    0.6     0.7817857  0.6395556  0.6996429
##   1.0    0.8     0.7839206  0.6504444  0.6946429
##   1.0    1.0     0.7874603  0.6548889  0.6921429
##
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were gamma = 0.6 and lambda = 0.
```

```
rda.pre = predict(rda.fit, test)
confusionMatrix(rda.pre, test$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  M  R
##          M 18  5
##          R  2 17
##
##                Accuracy : 0.8333
##                  95% CI : (0.6864, 0.9303)
##     No Information Rate : 0.5238
##     P-Value [Acc > NIR] : 2.795e-05
##
##                   Kappa : 0.6682
##
##  Mcnemar's Test P-Value : 0.4497
##
##             Sensitivity : 0.9000
##             Specificity : 0.7727
##          Pos Pred Value : 0.7826
##          Neg Pred Value : 0.8947
##              Prevalence : 0.4762
##          Detection Rate : 0.4286
##    Detection Prevalence : 0.5476
##       Balanced Accuracy : 0.8364
##
```

```
##        'Positive' Class : M
##
```

LDA:

```
lda.fit
```

```
## Linear Discriminant Analysis
##
## 166 samples
##  60 predictor
##   2 classes: 'M', 'R'
##
## Pre-processing: centered (60), scaled (60)
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 149, 150, 149, 150, 149, 149, ...
## Resampling results:
##
##   ROC        Sens       Spec
##   0.8139921  0.7893333  0.7285714
```

```
lda.pre = predict(lda.fit, test)
confusionMatrix(lda.pre, test$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  M  R
##          M 17 12
##          R  3 10
##
##                Accuracy : 0.6429
##                  95% CI : (0.4803, 0.7845)
##     No Information Rate : 0.5238
##     P-Value [Acc > NIR] : 0.08147
##
##                   Kappa : 0.2984
##
##  Mcnemar's Test P-Value : 0.03887
##
##             Sensitivity : 0.8500
##             Specificity : 0.4545
##          Pos Pred Value : 0.5862
##          Neg Pred Value : 0.7692
##              Prevalence : 0.4762
##          Detection Rate : 0.4048
##    Detection Prevalence : 0.6905
##       Balanced Accuracy : 0.6523
##
##        'Positive' Class : M
##
```

QDA:

```
qda.fit
```

```
## Quadratic Discriminant Analysis
```

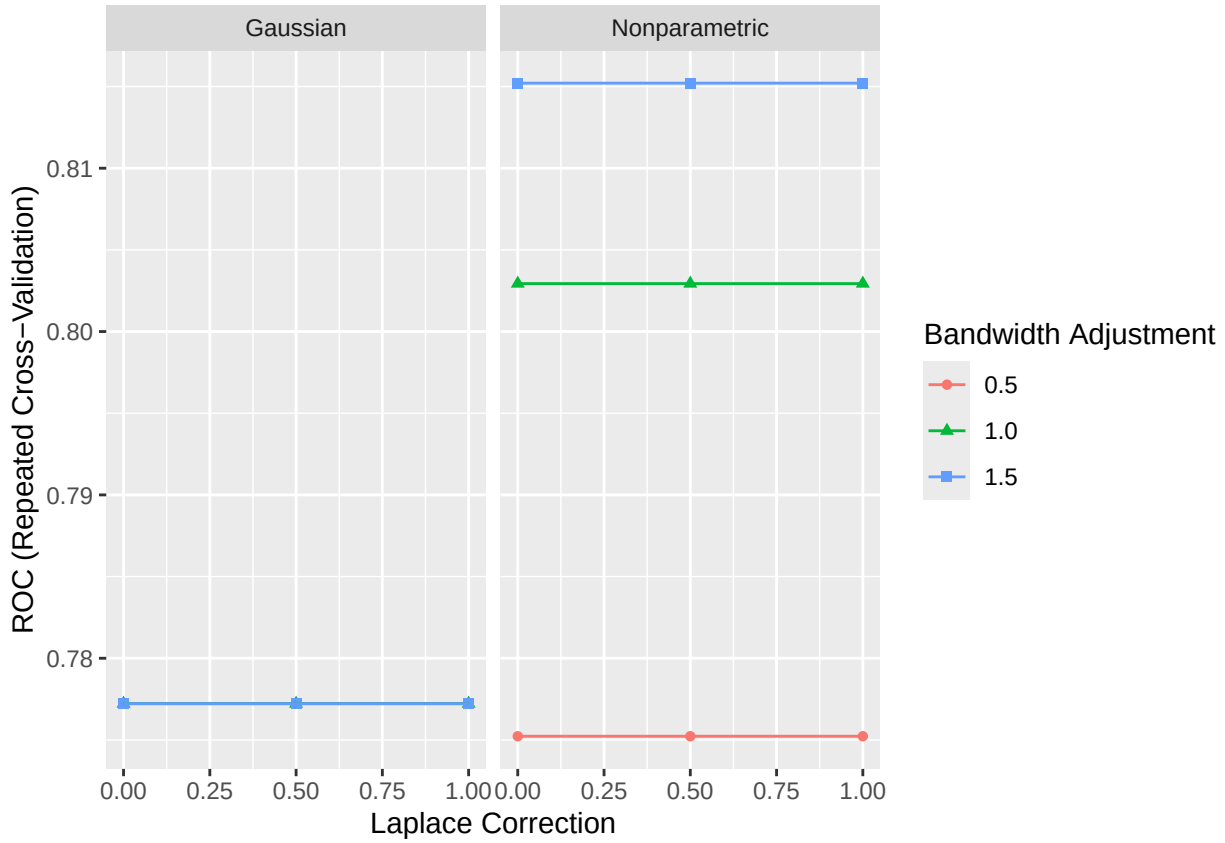
```
##
## 166 samples
##  60 predictor
##   2 classes: 'M', 'R'
##
## Pre-processing: centered (60), scaled (60)
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 148, 150, 150, 150, 149, 149, ...
## Resampling results:
##
##   ROC        Sens       Spec
##   0.6293115  0.9582222  0.2625
```

```
qda.pre = predict(qda.fit, test)
confusionMatrix(qda.pre, test$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  M  R
##          M 19 13
##          R  1  9
##
##                Accuracy : 0.6667
##                  95% CI : (0.5045, 0.8043)
##     No Information Rate : 0.5238
##     P-Value [Acc > NIR] : 0.043579
##
##                   Kappa : 0.3496
##
##  Mcnemar's Test P-Value : 0.003283
##
##             Sensitivity : 0.9500
##             Specificity : 0.4091
##          Pos Pred Value : 0.5938
##          Neg Pred Value : 0.9000
##              Prevalence : 0.4762
##          Detection Rate : 0.4524
##    Detection Prevalence : 0.7619
##       Balanced Accuracy : 0.6795
##
##        'Positive' Class : M
##
```

NB:

```
ggplot(nb.fit)
```




```
nb.fit
```

```
## Naive Bayes
##
## 166 samples
##  60 predictor
##   2 classes: 'M', 'R'
##
## Pre-processing: centered (60), scaled (60)
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 149, 150, 150, 149, 149, 149, ...
## Resampling results across tuning parameters:
##
##   fL   usekernel  adjust  ROC        Sens       Spec
##   0.0  FALSE      0.5     0.7772222  0.6148889  0.7642857
##   0.0  FALSE      1.0     0.7772222  0.6148889  0.7642857
##   0.0  FALSE      1.5     0.7772222  0.6148889  0.7642857
##   0.0   TRUE      0.5     0.7752262  0.8242222  0.6117857
##   0.0   TRUE      1.0     0.8029365  0.8415556  0.6596429
##   0.0   TRUE      1.5     0.8152063  0.7951111  0.6807143
##   0.5  FALSE      0.5     0.7772222  0.6148889  0.7642857
##   0.5  FALSE      1.0     0.7772222  0.6148889  0.7642857
##   0.5  FALSE      1.5     0.7772222  0.6148889  0.7642857
##   0.5   TRUE      0.5     0.7752262  0.8242222  0.6117857
##   0.5   TRUE      1.0     0.8029365  0.8415556  0.6596429
##   0.5   TRUE      1.5     0.8152063  0.7951111  0.6807143
##   1.0  FALSE      0.5     0.7772222  0.6148889  0.7642857
```

```
##   1.0  FALSE      1.0     0.7772222  0.6148889  0.7642857
##   1.0  FALSE      1.5     0.7772222  0.6148889  0.7642857
##   1.0   TRUE      0.5     0.7752262  0.8242222  0.6117857
##   1.0   TRUE      1.0     0.8029365  0.8415556  0.6596429
##   1.0   TRUE      1.5     0.8152063  0.7951111  0.6807143
##
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were fL = 0, usekernel = TRUE and adjust
##  = 1.5.
```

```
nb.pre = predict(nb.fit, test)
confusionMatrix(nb.pre, test$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  M  R
##          M 20  6
##          R  0 16
##
##                Accuracy : 0.8571
##                  95% CI : (0.7146, 0.9457)
##     No Information Rate : 0.5238
##     P-Value [Acc > NIR] : 5.736e-06
##
##                   Kappa : 0.7175
##
##  Mcnemar's Test P-Value : 0.04123
##
##             Sensitivity : 1.0000
##             Specificity : 0.7273
##          Pos Pred Value : 0.7692
##          Neg Pred Value : 1.0000
##              Prevalence : 0.4762
##          Detection Rate : 0.4762
##    Detection Prevalence : 0.6190
##       Balanced Accuracy : 0.8636
##
##        'Positive' Class : M
##
```

Finally, our goal is to compare the models and find the best model, so we have:

```
resamp = resamples(list(PLS = pls.fit, RDA = rda.fit, LDA = lda.fit, QDA = qda.fit, NB = nb.fit))
summary(resamp)
```

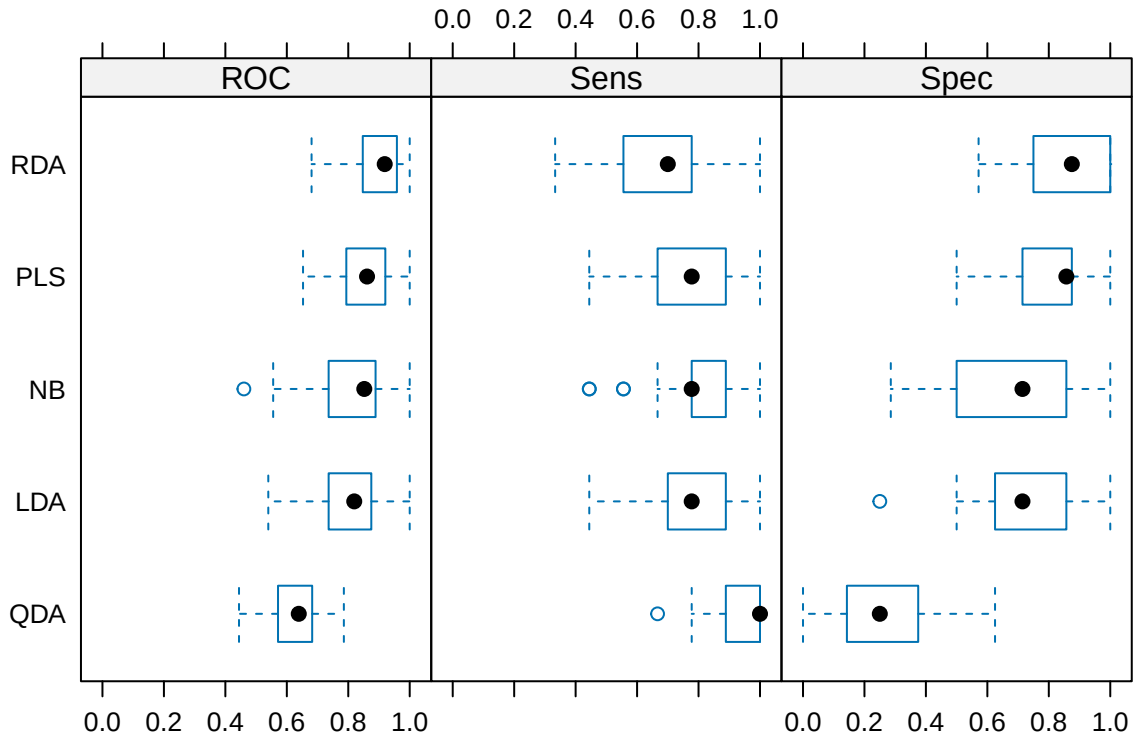
```
##
## Call:
## summary.resamples(object = resamp)
##
## Models: PLS, RDA, LDA, QDA, NB
## Number of resamples: 50
##
## ROC
##          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## PLS 0.6527778 0.7952381 0.8611111 0.8564127 0.9196429 1.0000000    0
```

```
## RDA 0.6805556 0.8497024 0.9186508 0.8999802 0.9583333 1.0000000    0
## LDA 0.5396825 0.7425595 0.8194444 0.8139921 0.8705357 1.0000000    0
## QDA 0.4444444 0.5714286 0.6388889 0.6293115 0.6825397 0.7857143    0
## NB  0.4603175 0.7385913 0.8521825 0.8152063 0.8888889 1.0000000    0
##
## Sens
##          Min.   1st Qu.    Median      Mean   3rd Qu. Max. NA's
## PLS 0.4444444 0.6666667 0.7777778 0.7646667 0.8888889    1    0
## RDA 0.3333333 0.5555556 0.7000000 0.7013333 0.7777778    1    0
## LDA 0.4444444 0.7194444 0.7777778 0.7893333 0.8888889    1    0
## QDA 0.6666667 0.9166667 1.0000000 0.9582222 1.0000000    1    0
## NB  0.4444444 0.7777778 0.7777778 0.7951111 0.8888889    1    0
##
## Spec
##          Min.   1st Qu.    Median      Mean   3rd Qu.  Max. NA's
## PLS 0.5000000 0.7142857 0.8571429 0.7796429 0.8705357 1.000    0
## RDA 0.5714286 0.7767857 0.8750000 0.8760714 1.0000000 1.000    0
## LDA 0.2500000 0.6250000 0.7142857 0.7285714 0.8571429 1.000    0
## QDA 0.0000000 0.1428571 0.2500000 0.2625000 0.3750000 0.625    0
## NB  0.2857143 0.5000000 0.7142857 0.6807143 0.8571429 1.000    0
```

And for graphical comparing we have:

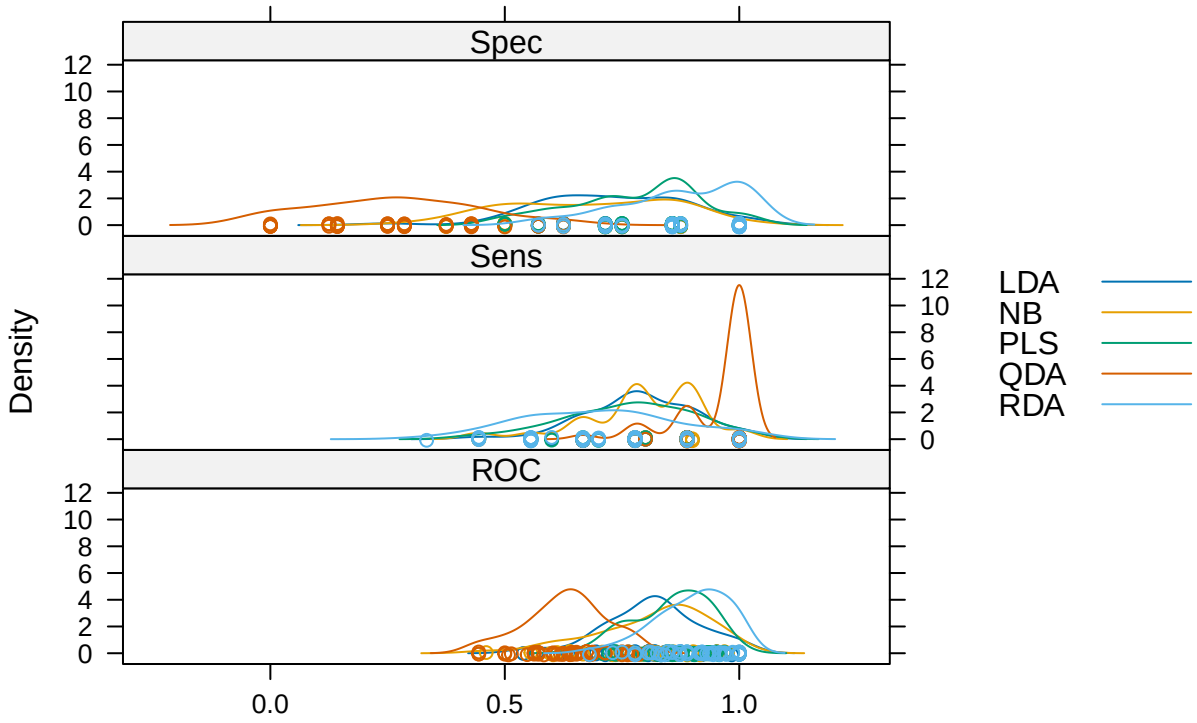
Box-Whisker Plot:

```
bwplot(resamp, layout = c(3,1))
```



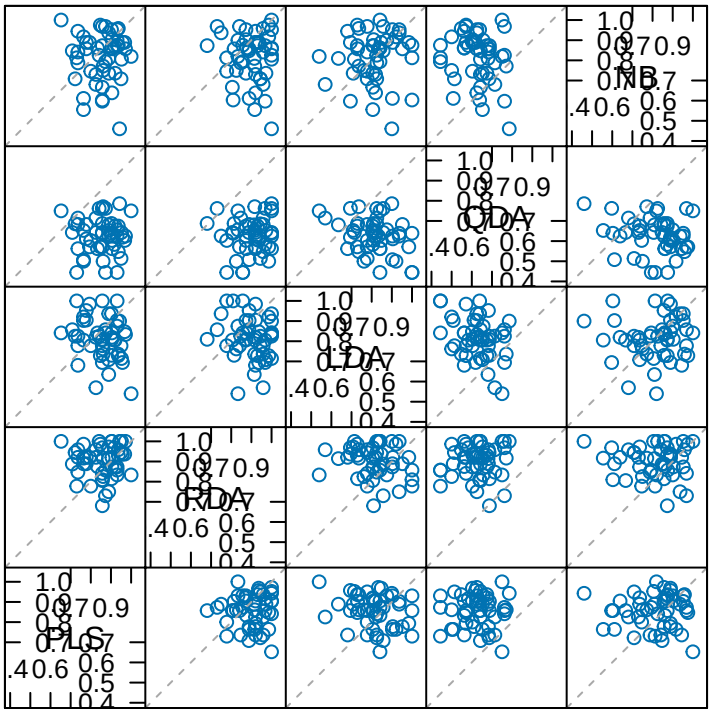
Density Plot:

```
densityplot(resamp, layout = c(1,3), auto.key = list(column = 1))
```



Scatter Plot Matrix:

```
splom(resamp)
```



After that, for testing the differences between proposed models we have:

```
difresamp = diff(resamp)
summary(difresamp)
```

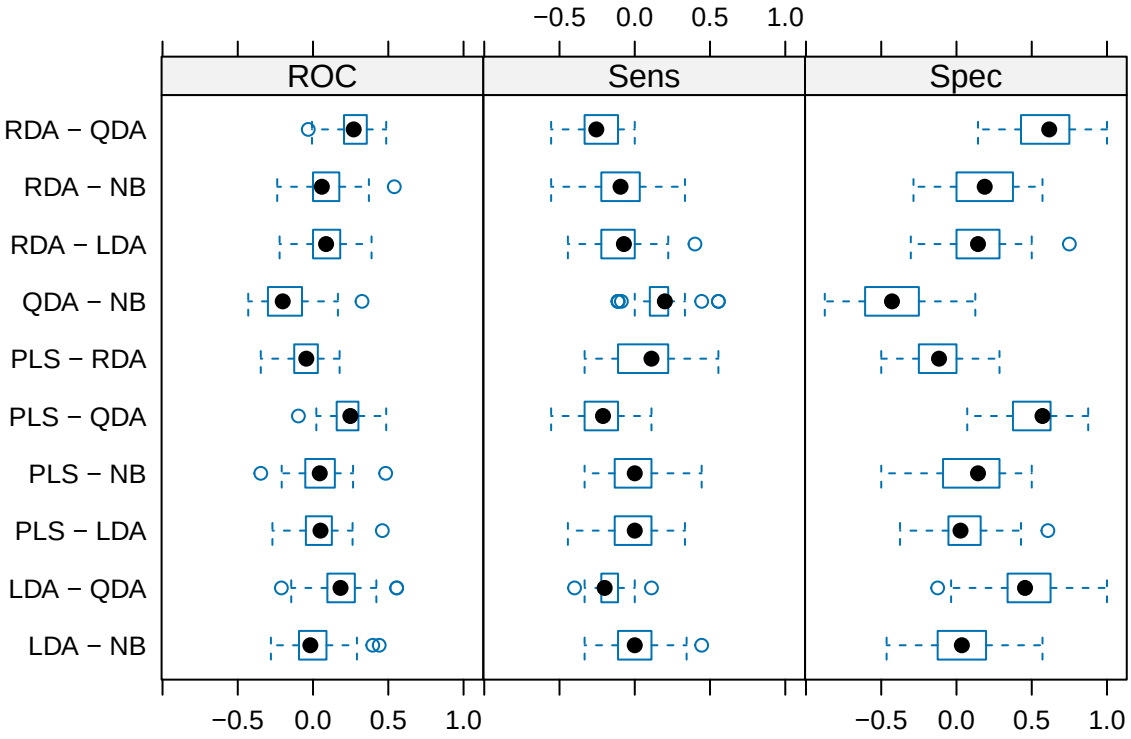
```
##
## Call:
## summary.diff.resamples(object = difresamp)
##
## p-value adjustment: bonferroni
## Upper diagonal: estimates of the difference
## Lower diagonal: p-value for H0: difference = 0
##
## ROC
##     PLS        RDA        LDA        QDA        NB
## PLS           -0.043567  0.042421  0.227101  0.041206
## RDA 0.07761               0.085988  0.270669  0.084774
## LDA 0.43754   0.00069               0.184681 -0.001214
## QDA < 2.2e-16 < 2.2e-16 4.046e-10           -0.185895
## NB  0.49715   0.00184   1.00000   1.226e-09
##
## Sens
##     PLS        RDA        LDA        QDA        NB
## PLS            0.063333 -0.024667 -0.193556 -0.030444
## RDA 0.54672               -0.088000 -0.256889 -0.093778
## LDA 1.00000   0.01697               -0.168889 -0.005778
## QDA 2.765e-10 4.305e-14 2.733e-12            0.163111
```

```
## NB  1.00000   0.02323   1.00000   1.022e-08
##
## Spec
##     PLS       RDA       LDA       QDA       NB
## PLS           -0.09643   0.05107   0.51714   0.09893
## RDA 0.0066390            0.14750   0.61357   0.19536
## LDA 0.8438075 0.0002986            0.46607   0.04786
## QDA < 2.2e-16 < 2.2e-16 < 2.2e-16           -0.41821
## NB  0.0500783 7.708e-07 1.0000000 1.915e-14
```

Now, the graphical summary of this is:

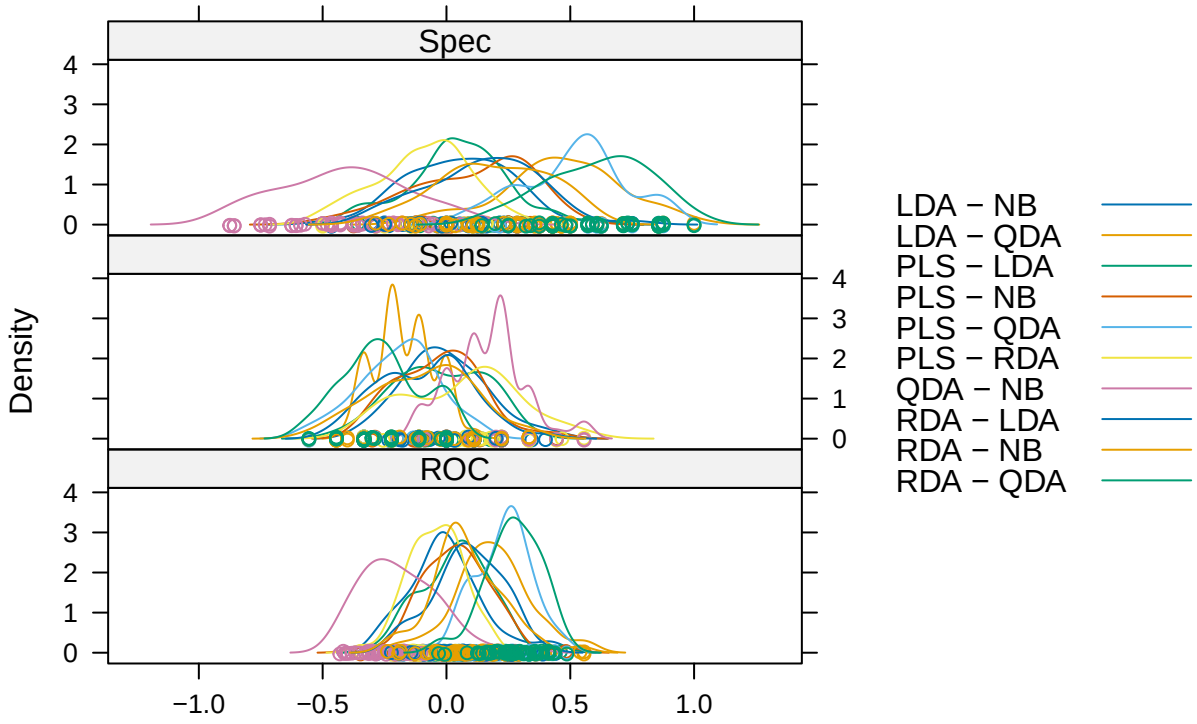
Box-Whisker Plot for difference:

```
bwplot(difresamp, layout = c(3,1))
```



Density Plot for difference:

```
densityplot(difresamp, layout = c(1,3), auto.key = list(column = 1))
```



We observe that `PLS`, `NB`, and `RDA` have same accuracy. We choose `PLS` because it is used dimension reduction for modeling. In `RDA` has not dimension reduction property and if we have multicollinearity, interpretation of coefficients are difficult. For `NB` we have to use kernel estimation for 60 variables and mutually exclusive independence between variables. It is surprising that, we assume mutually exclusive independence between variables! So our final model is `PLS` with the best tune parameter. `(ncomp = 4)`

```
pls.fit$bestTune
```

```
##   ncomp
## 4     4
```

```
pls.final = train(
  Class ~ .,
  data = Sonar,
  method = 'pls',
  preProcess = c('center','scale'),
  tuneGrid = data.frame(ncomp = 4),
  metric = 'ROC',
  trControl = trainControl(
    method = 'none',
    classProbs = T,
    summaryFunction = twoClassSummary,
    verboseIter = F,
    savePredictions = T
    )
)
```

And for summary of final model we have:

```r
pls.final.pred = predict(pls.final, Sonar)
confusionMatrix(pls.final.pred, Sonar$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  M  R
##          M 96 13
##          R 15 84
##
##                Accuracy : 0.8654
##                  95% CI : (0.8114, 0.9086)
##     No Information Rate : 0.5337
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.7299
##
##  Mcnemar's Test P-Value : 0.8501
##
##             Sensitivity : 0.8649
##             Specificity : 0.8660
##          Pos Pred Value : 0.8807
##          Neg Pred Value : 0.8485
##              Prevalence : 0.5337
##          Detection Rate : 0.4615
##    Detection Prevalence : 0.5240
##       Balanced Accuracy : 0.8654
##
##        'Positive' Class : M
##
```

Our model accuracy is 82.54% (Sens = 86.49% and Spec = 86.60%), which is good result.

Thank you for your attention.

Best regards

email: mohammadali.mirzaie2000@gmail.com