Lab08

Channing_503128649

03/10/2021

Setup

```
library(MASS)
library(class)
library(gam)
```

```
## Loading required package: splines
```

```
## Loading required package: foreach
```

```
## Loaded gam 1.20
```

```
library(nnet)
library(e1071)
library(parallel)
```

Import

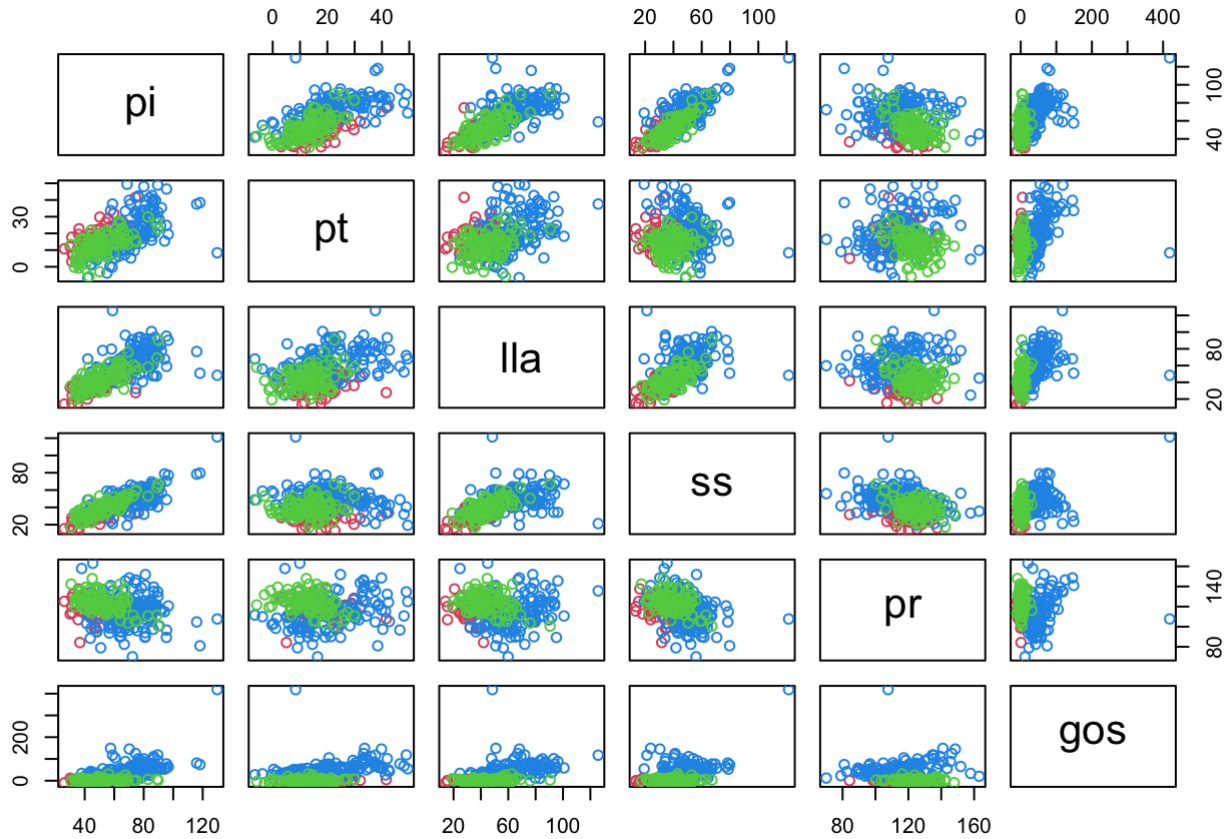
```
vc = read.csv("vertebral-column.csv", stringsAsFactors=TRUE)
head(vc)
```

```
##      pi    pt   lla    ss     pr   gos class
## 1 63.03 22.55 39.61 40.48  98.67 -0.25    DH
## 2 39.06 10.06 25.02 29.00 114.41  4.56    DH
## 3 68.83 22.22 50.09 46.61 105.99 -3.53    DH
## 4 69.30 24.65 44.31 44.64 101.87 11.21    DH
## 5 49.71  9.65 28.32 40.06 108.17  7.92    DH
## 6 40.25 13.92 25.12 26.33 130.33  2.23    DH
```

Exploration

1. Create a pairwise scatterplot, with observations of different classes shown in different colors.

```
plot(vc[,-7], col=as.numeric(vc[,7])+1)
```



#

Classification Five classification methods are listed in the next 5 tasks. Use each of them to build a model for the data set vc and predict the class labels of the observations that are used to build the model. For each, compute the confusion matrix and (resubstitution) classification accuracy.

```
r = vector("list", 5)
A = vector("numeric", 5)
names(r) = c("LDA", "QDA", "NB", "MLR", "KNN")
names(A) = c("LDA", "QDA", "NB", "MLR", "KNN")
```

2. Linear discriminant analysis

```
r[[1]] = lda(class ~ ., data=vc)
yhat = predict(r[[1]], newdata=vc)$class
table(vc$class, yhat)
```

```
##     yhat
##       DH  NO  SL
##   DH  39  20   1
##   NO  12  79   9
##   SL   4  11 135
```

```
(A[1] = mean(vc$class == yhat))
```

```
## [1] 0.816129
```

3. Quadratic discriminant analysis

```
r[[2]] = qda(class ~ ., data=vc)
yhat = predict(r[[2]], newdata=vc)$class
table(vc$class, yhat)
```

```
##     yhat
##       DH  NO  SL
##   DH  42  16   2
##   NO  15  81   4
##   SL   1   1 148
```

```
(A[2] = mean(vc$class == yhat))
```

```
## [1] 0.8741935
```

4. Naive Bayes

```
r[[3]] = naiveBayes(class ~ ., data=vc)
yhat = predict(r[[3]], newdata=vc)
table(vc$class, yhat)
```

```
##     yhat
##       DH  NO  SL
##   DH  45  12   3
##   NO  21  69  10
##   SL   0   5 145
```

```
(A[3] = mean(vc$class == yhat))
```

```
## [1] 0.8354839
```

5. Multinomial logistic regression

```
r[[4]] = multinom(class ~ ., data=vc)
```

```
## # weights:  24 (14 variable)
## initial  value 340.569809
## iter  10 value 182.252739
## iter  20 value 91.500126
## iter  30 value 89.599802
## iter  40 value 89.590489
## iter  50 value 89.584980
## iter  60 value 89.544492
## iter  70 value 89.544275
## iter  70 value 89.544275
## final  value 89.544264
## converged
```

```
yhat = predict(r[[4]], newdata=vc)
table(vc$class, yhat)
```

```
##     yhat
##       DH  NO  SL
##   DH  40  19   1
##   NO  13  85   2
##   SL   1   3 146
```

```
(A[4] = mean(vc$class == yhat))
```

```
## [1] 0.8741935
```

6. K-nearest neighbours (with K=10)

```
yhat = knn(train=vc[,-7], test=vc[,-7], cl=vc[,7], k=10)  # K = 1
table(vc[,7], yhat)
```

```
##     yhat
##       DH  NO  SL
##   DH  43  16   1
##   NO  13  84   3
##   SL   1   3 146
```

```
(A[5] = mean(vc$class == yhat))
```

```
## [1] 0.8806452
```

Primary Performance Evaluation

7. Present your resulting classification accuracy for all five classification methods in a table (in whatever format)

```
names(A[which.max(A)])
```

```
## [1] "KNN"
```

```
length(unique(vc$class))
```

```
## [1] 3
```

All of them do a better job (Accuracy >80%) than the random guess (Accuracy = 33.33%), where KNN performed the best.

Multiple Logistic Regression and Generalised Additive Models

8. Create a response variable from variable class so that both classes DH and SL are relabelled as AB (Abnormal).

```
vc$newclass = as.factor(ifelse(vc$class=="DH" | vc[, 7]=="SL","AB", "NO"))
```

Use glm() to build a multiple logistic regression model for this new class variable, and compute the confusion matrix and resubstitution classification accuracy.

```
r1 = glm(newclass ~ .-class, data=vc, family=binomial)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
p = predict(r1, newdata=vc, type="response")
yhat = as.numeric(p > 0.5) + 1
table(vc$newclass, levels(vc$newclass)[yhat])
```

```
##
##       AB  NO
##   AB 188  22
##   NO  22  78
```

```
(A1 = mean(vc$newclass == levels(vc$newclass)[yhat]))
```

```
## [1] 0.8580645
```

(@)Use step() to find the AIC-selected model, with backward selection. Which variables are removed by the AIC?

```
r1 = step(r1, type="backward")
```

```
## Start:  AIC=192.13
## newclass ~ (pi + pt + lla + ss + pr + gos + class) - class
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
##        Df Deviance    AIC
## - ss    1   178.27 190.27
## - pi    1   178.27 190.27
## - pt    1   178.27 190.27
## - lla   1   178.86 190.86
## <none>      178.13 192.13
## - pr    1   209.31 221.31
## - gos   1   302.31 314.31
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
##
## Step:  AIC=190.27
## newclass ~ pi + pt + lla + pr + gos
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
##        Df Deviance    AIC
## - lla   1   178.95 188.95
## <none>      178.27 190.27
## - pi    1   189.08 199.08
## - pt    1   202.23 212.23
## - pr    1   209.36 219.36
## - gos   1   303.45 313.45
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
##
## Step:  AIC=188.95
## newclass ~ pi + pt + pr + gos
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
##        Df Deviance    AIC
## <none>      178.95 188.95
## - pt    1   208.55 216.55
## - pr    1   212.18 220.18
## - pi    1   212.53 220.53
## - gos   1   310.54 318.53
```

```
summary(r1)
```

```
##
## Call:
## glm(formula = newclass ~ pi + pt + pr + gos, family = binomial,
##     data = vc)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.21933  -0.37927  -0.03193   0.40184   2.79266
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -15.45723    3.27273  -4.723 2.32e-06 ***
## pi            0.11495    0.02266   5.072 3.94e-07 ***
## pt           -0.18140    0.03784  -4.794 1.63e-06 ***
## pr            0.10895    0.02279   4.780 1.75e-06 ***
## gos          -0.16538    0.02288  -7.227 4.92e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 389.86  on 309  degrees of freedom
## Residual deviance: 178.95  on 305  degrees of freedom
## AIC: 188.95
##
## Number of Fisher Scoring iterations: 8
```

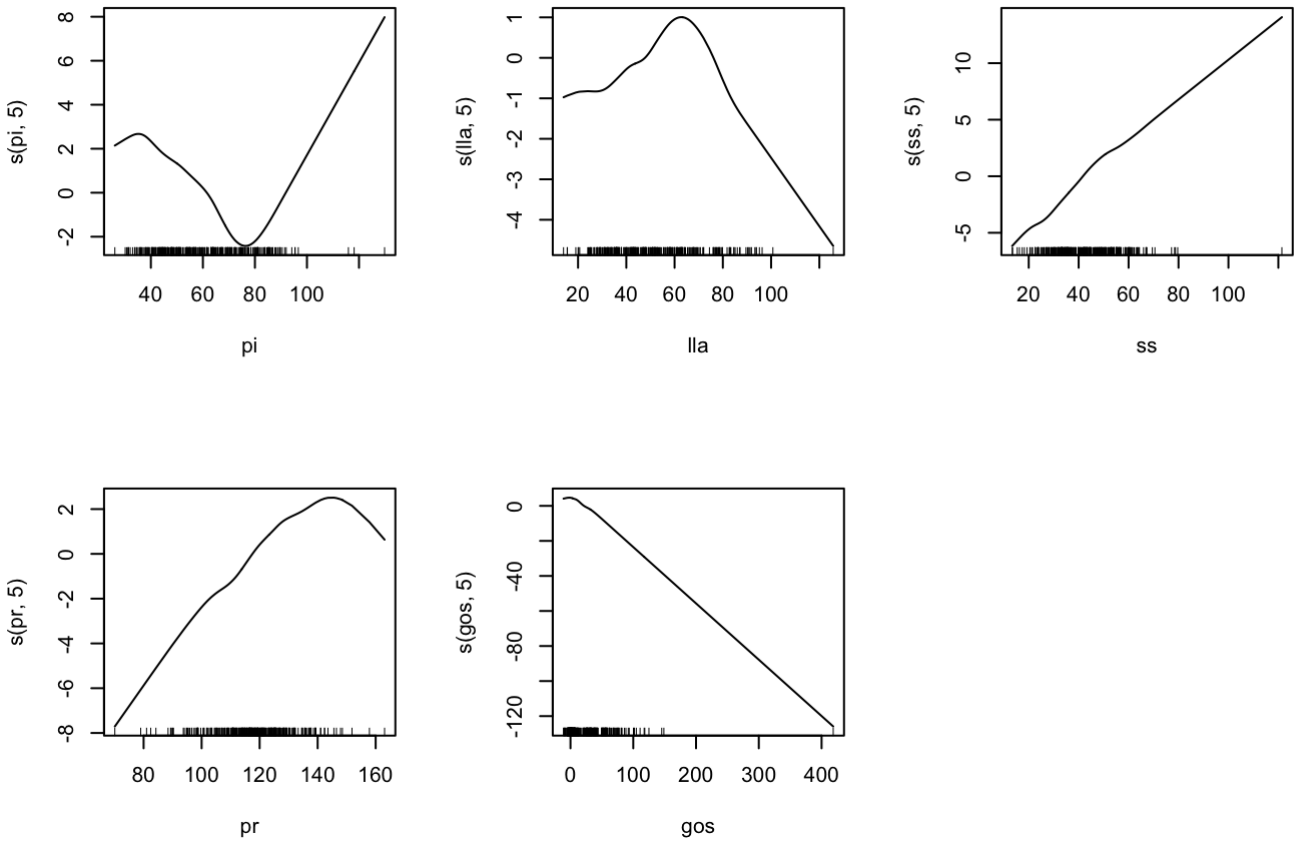
llr, ss are removed by AIC

9. Extend the AIC-selected model with gam() so that each linear term is replaced with a smoothing spline of 5 degrees of freedom. Take a visual approach to reasonably lower the degrees of freedom in each term. For your chosen model, compute the confusion matrix and classification accuracy.

```
par(mfrow=c(2,3))
r1 = gam(newclass ~ s(pi, 5) + s(lla, 5) + s(ss, 5) + s(pr,5) + s(gos, 5), data=vc, f
amily=binomial())
summary(r1)
```

```
##
## Call: gam(formula = newclass ~ s(pi, 5) + s(lla, 5) + s(ss, 5) + s(pr,
##     5) + s(gos, 5), family = binomial(), data = vc)
## Deviance Residuals:
##       Min        1Q    Median        3Q       Max
## -2.054408 -0.248374 -0.004163  0.312835  2.596147
##
## (Dispersion Parameter for binomial family taken to be 1)
##
##     Null Deviance: 389.8556 on 309 degrees of freedom
## Residual Deviance: 143.8526 on 284 degrees of freedom
## AIC: 195.8526
##
## Number of Local Scoring Iterations: NA
##
## Anova for Parametric Effects
##            Df  Sum Sq Mean Sq F value    Pr(>F)
## s(pi, 5)    1   0.020   0.020  0.0375 0.8466535
## s(lla, 5)   1   1.041   1.041  1.9317 0.1656579
## s(ss, 5)    1   7.138   7.138 13.2452 0.0003246 ***
## s(pr, 5)    1  10.002  10.002 18.5593 2.272e-05 ***
## s(gos, 5)   1  39.617  39.617 73.5117 6.557e-16 ***
## Residuals 284 153.054   0.539
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##             Npar Df Npar Chisq  P(Chi)
## (Intercept)
## s(pi, 5)          4     8.4470 0.07650 .
## s(lla, 5)         4     5.4449 0.24463
## s(ss, 5)          4     2.8070 0.59055
## s(pr, 5)          4     4.5488 0.33683
## s(gos, 5)         4    11.4353 0.02209 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
plot(r1)
```



pi, lla, pr seem to be quadratic regressions and ss, gos seem to linear regression

```
r1 = gam(newclass ~ s(pi, 2) + s(lla, 2) + ss + s(pr,2) + gos, data=vc, family=binomi
al())
p = predict(r1, vc)
yhat = as.numeric(p > 0.5) + 1
table(vc$newclass, levels(vc$newclass)[yhat])
```

```
##
##        AB  NO
##    AB 199  11
##    NO  28  72
```

```
(A1 = mean(vc$newclass == levels(vc$newclass)[yhat]))
```

```
## [1] 0.8741935
```

Cross-validation and Parallel Computing

10. Reconsider the 3-class problem studied in Questions 2-7. Use 10 repetitions of 10-fold cross-validation to evaluate the performance of the 5 classification methods. Present your resulting classification accuracy of all five classification methods in a table. Comment on the results.

```
vc = read.csv("vertebral-column.csv", stringsAsFactors=TRUE)

n = nrow(vc)       # total number of observations
c = 1:5  # all the classifiers for comparison
R = 5                   # number of repetitions
K = 10                  # K-fold CV
a = matrix(nrow=R*K, ncol=length(c))   # pre-allocate space

# indies of a test test in CV
# i -- which fold
# n -- sample size
# K -- total number of folds

test.set = function(i, n, K=10) {
  if(i < 1 || i > K)
    stop(" i out of range (1, K)")
  start = round(1 + (i-1) * n / K)
  end = ifelse(i == K, n, round(i * n / K))
  start:end
}

set.seed(769)           # set a random seed

for(i in 1:R) {                 # for each repetition
  ind = sample(n)
  for(k in 1:K) {               # for each fold
    index = ind[test.set(k, n, K)]
    test = vc[index,]
    train = vc[-index,]
    for(j in 1:length(c)) {     # for each classifier
      if (j ==1 || j ==2) yhat = predict(r[[1]], newdata=test)$class
      if (j ==3 || j ==4) yhat = predict(r[[j]], newdata=test)
      if (j == 5) yhat = knn(train=train[,-7], test=test[,-7], cl=train[,7], k=10)
      a[K*(i-1)+k,j] = mean(test$class == yhat)
    }
  }
}

head(a)
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] 0.8387097 0.8387097 0.7741935 0.8709677 0.8064516
## [2,] 0.8709677 0.8709677 0.8387097 0.9677419 0.9032258
## [3,] 0.6451613 0.6451613 0.7096774 0.7419355 0.7741935
## [4,] 0.7741935 0.7741935 0.8064516 0.8387097 0.8064516
## [5,] 0.7741935 0.7741935 0.7741935 0.8064516 0.8064516
## [6,] 0.7741935 0.7741935 0.8387097 0.8064516 0.7741935
```

```
(pe = cbind(c=c, a=colMeans(a)))
```

```
##      c         a
## [1,] 1 0.8161290
## [2,] 2 0.8161290
## [3,] 3 0.8354839
## [4,] 4 0.8741935
## [5,] 5 0.8283871
```

During the cross validation, MLR is the best out of the first 5 classifiers.

11. Modify your code so that parallel computing can be used, in which each job running in parallel is only for the computation about one fold out of 10 (i.e., one training set and one test set). Make sure that the same subsamples are used by different methods and that the results are reproducible. Compare timings with 1, 5, 10, 20 cores.

Encapsulate the function for the each fold, and then compute the accurracy in parallel. 5 cores should be 5 times faster than 1 core, 10 cores should be 2 times faster than 5 cores, but 20 cores wouldn't see a significant improvement as the performance is plateaued by folds rather than the actual computing inside the fold.

Summary

In this lab, we look into the classification problems.

Firstly, we compared among Linear discriminant analysis, Quadratic discriminant analysis, Naive Bayes, Multinomial logistic regression, K-nearest neighbours (with K=10), and found that KNN performed the best. The result is trustworthy as in KNN, we don't assume normal distributions for all predictors, but we do have to take computational cost into considerations if the data set is large as KNN is memory based method that needs more resources as the data set grows.

Secondly, we transformed the reponse variables into 2 classes ('NO' and 'AB') rather than 3 classes ('NO', 'DH', 'SL') and then compared between Multiple Logistic Regression and Generalised Additive Models. We can see that GAM performed slightly better, but it needs an extra step to tune the pynonomial degrees in each smoothing splines.

Lastly, we took advantage of the computational power to do the cross validation and the parallel computing more efficiently to better our models.