Special Assignments lab 2

Filip Cornell

2018-12-05

1 Task 1

In task 1, the task was to create one's own version of LDA, that will separate the data, and compare the misclassification rate to the one for the predefined function `lda()`.

1.1 Part a

First, we construct the function and get the misclassification rates. I am not sure I have understood the instructions perfectly, but I tried to follow them as far as I understood them. The code can be seen below.

```
data = read.csv("australian-crabs.csv")
DiscFunc = function(x, cov_k, mu_k, prior) {
  x = as.matrix(x)
  mu_k = as.matrix(mu_k)
  express = t(x)%*%solve(cov_k)%*%mu_k -
    (1/2)%*%t(mu_k)%*%solve(cov_k)%*%mu_k + log(prior)
  return(express)
}
classificationRate = function(confMatrix) {
  return(sum(diag(confMatrix))/sum(confMatrix))
}

# This function assumes only two dimensions of the data!
# Send in data as features as columns, data points as rows
LDA = function(x1, x2, prior1, prior2, test) {
  mu_1 = apply(x1, 2, mean)
  mu_2 = apply(x2, 2, mean)
  cov_1 = cov(x1)
  cov_2 = cov(x2)
  n1 = nrow(x1)
  n2 = nrow(x2)
  N = n1+n2
  covM = 1/N*(n1*cov_1 + n2*cov_2)
  classificationsTest = apply(test, 1, function(row) {
    class1 = DiscFunc(row, covM, mu_1, prior1)
    class2 = DiscFunc(row, covM, mu_2, prior2)
    return(which.max(c(class1,class2)))
  })
  classificationsTest = factor(ifelse(classificationsTest == 2, "Male", "Female"))

  # Is proportional to, so we do not get the constant needed.
  decisionBoundary = solve(covM)%*%(mu_2 - mu_1)

  return(list(classificationsTest, decisionBoundary))
}
```

```
x2 = matrix(c(data[data$sex == "Male",]$CL,
              data[data$sex == "Male",]$RW),
            nrow = nrow(data[data$sex == "Male",]), ncol = 2)
x1 = matrix(c(data[data$sex == "Female",]$CL,
              data[data$sex == "Female",]$RW),
            nrow = nrow(data[data$sex == "Female",]), ncol = 2)
test_m = matrix(c(data$CL, data$RW), nrow = nrow(data), ncol = 2)

result = LDA(x1,x2,prior1 = 0.5,prior2 = 0.5, test = test_m)
classificationsTest = result[[1]]
decisionB = result[[2]]

print("Misclassification rate test")
```

```
## [1] "Misclassification rate test"
```

```
table(data$sex, classificationsTest)
```

```
##         classificationsTest
##          Female Male
##   Female     97    3
##   Male        4   96
```

```
1 - classificationRate(table(data$sex, classificationsTest))
```

```
## [1] 0.035
```

Comparing the misclassification rates with the ones obtained in lab assignment 1, we see that the results are in fact identical.

```
library(MASS)

fitModel = lda(formula = sex ~ RW + CL,
               data = data,
               prior = c(length(data$sex[data$sex == "Male"])
                         /nrow(data),length(data$sex[data$sex == "Female"])
                         /nrow(data)))
fits = predict(fitModel, data)
confMatrix = table(data$sex, fits$class)
confMatrix
```

```
##
##          Female Male
##   Female     97    3
##   Male        4   96
```

```
print("misclassificionation rate test")
```

```
## [1] "misclassificionation rate test"
```

```
1 - classificationRate(confMatrix)
```

```
## [1] 0.035
```

1.2 Part b

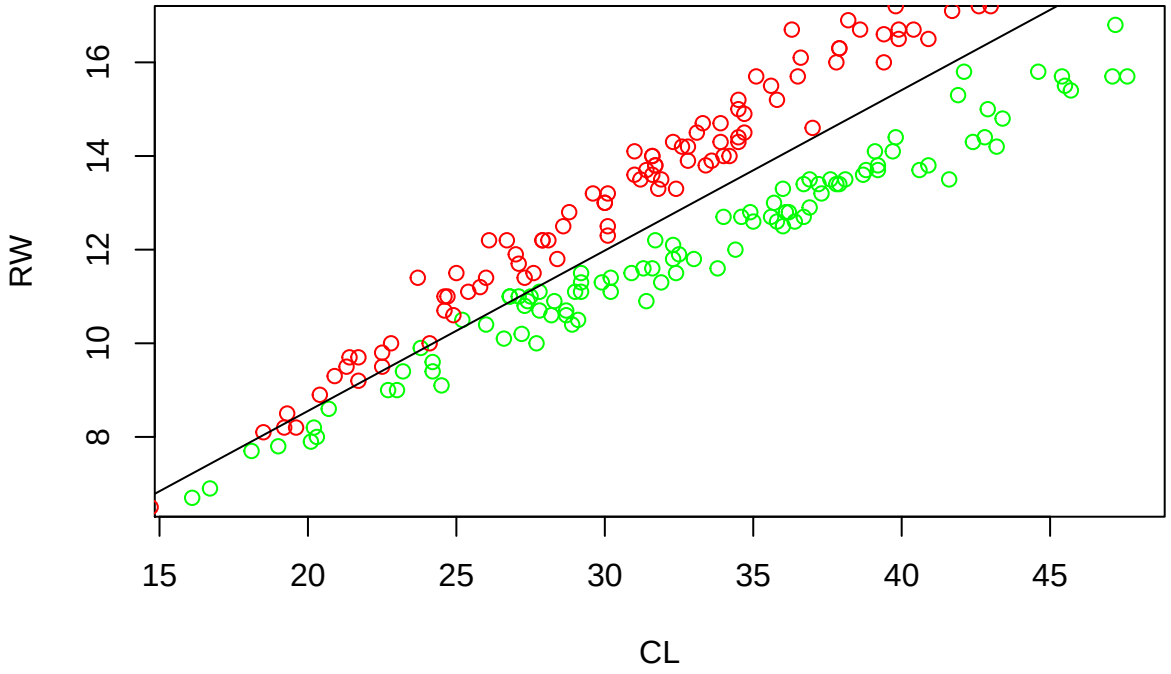
In part b), we draw the decision boundary. Using the result from Pattern Recognition and Machine Learning, the course book, page x, we see that

$$w \propto S_w(\mu_2 - \mu_1), \text{ where } S_w = S_1 + S_2$$

This is the line onto which we can project the points and get a clear separation. Thus, I quickly found the separating line, but without the constant c. Thus, I had to guess a bit to make the decision line pass correctly in a visualizing way. I am very curious to hear suggestion on how to find the appropriate constant c, even though it algebraically does not really matter.

```
plot(data[data$sex == "Male",]$CL,
     data[data$sex == "Male",]$RW,
     col = "green", xlab = "CL", ylab = "RW")
points(data[data$sex == "Female",]$CL,
       data[data$sex == "Female",]$RW, col = "red")


xSeq = seq(min(data$CL), max(data$CL))
c = 1.7 # How find constant c?
RWvals = -(decisionB[1,]*xSeq/decisionB[2,]) + c
lines(x=xSeq, y = RWvals)
```



2 Task 2

In task 2, the task was to create a Naive Bayes implementation using the density function, and predict species.

```
## Special task 4

classificationRate = function(confMatrix) {
```

```
  return(sum(diag(confMatrix))/sum(confMatrix))
}

data = read.csv("australian-crabs.csv")
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data[id,]
test=data[-id,]

NaiveBayesImpl = function(train, test) {

  # Divide into subsets of the classes
  trainBlue = train[train$species == "Blue",]
  trainOrange = train[train$species == "Orange",]

  # Set priors proportional to the class sizes
  priorBlue = nrow(trainBlue)/(nrow(trainBlue) + nrow(trainOrange))
  priorOrange = 1 - priorBlue

  # Estimate the densities for the training sets
  # through density() function for each class for each measurement variable.
  # Could definitely be done smoother than this, but this works too.

  BlueCLDens = density(trainBlue$CL)
  OrangeCLDens = density(trainOrange$CL)
  BlueRWDens = density(trainBlue$RW)
  OrangeRWDens = density(trainOrange$RW)
  BlueFLDens = density(trainBlue$FL)
  OrangeFLDens = density(trainOrange$FL)
  BlueCWDens = density(trainBlue$CW)
  OrangeCWDens = density(trainOrange$CW)

  BlueBDDens = density(trainBlue$BD)
  OrangeBDDens = density(trainOrange$BD)

  trainFeats = as.matrix(train[,-c(1,2,3)])
  testFeats = as.matrix(test[,-c(1,2,3)])

  # Classify them according to unnormalized probabilities
  #obtained through densities calculated above.
  # This assumes conditional
  #independence of the variables, just like Naive Bayes
  trainClassifications = factor(apply(trainFeats, 1, function(row) {
    blue = approx(BlueFLDens$x, BlueFLDens$y, xout = row[1])$y*
      approx(BlueRWDens$x, BlueRWDens$y, xout = row[2])$y*
      approx(BlueCLDens$x,BlueCLDens$y,xout=row[3])$y*
      approx(BlueCWDens$x, BlueCWDens$y, xout = row[4])$y*
      approx(BlueBDDens$x, BlueBDDens$y, xout = row[5])$y*
      priorBlue
    orange = approx(OrangeFLDens$x, OrangeFLDens$y, xout = row[1])$y*
      approx(OrangeRWDens$x, OrangeRWDens$y, xout = row[2])$y*
      approx(OrangeCLDens$x,OrangeCLDens$y,xout=row[3])$y*
```

```
      approx(OrangeCWDens$x, OrangeCWDens$y, xout = row[4])$y*
      approx(OrangeBDDens$x, OrangeBDDens$y, xout = row[5])$y*
      priorOrange
    return(ifelse(blue >= orange, "blue", "orange"))
  }))
  print("Confusion matrix train")
  print(table(train$species, trainClassifications))
  print("Misclassification rate train")
  print(1 - classificationRate(table(train$species, trainClassifications)))
  testClassifications = factor(apply(testFeats, 1, function(row) {
    blue = approx(BlueFLDens$x, BlueFLDens$y, xout = row[1])$y*
      approx(BlueRWDens$x, BlueRWDens$y, xout = row[2])$y*
      approx(BlueCLDens$x,BlueCLDens$y,xout=row[3])$y*
      approx(BlueCWDens$x, BlueCWDens$y, xout = row[4])$y*
      approx(BlueBDDens$x, BlueBDDens$y, xout = row[5])$y*
      priorBlue
    orange = approx(OrangeFLDens$x, OrangeFLDens$y, xout = row[1])$y*
      approx(OrangeRWDens$x, OrangeRWDens$y, xout = row[2])$y*
      approx(OrangeCLDens$x,OrangeCLDens$y,xout=row[3])$y*
      approx(OrangeCWDens$x, OrangeCWDens$y, xout = row[4])$y*
      approx(OrangeBDDens$x, OrangeBDDens$y, xout = row[5])$y*
      priorOrange
    return(ifelse(blue >= orange, "blue", "orange"))
  }))
  print("Confusion matrix test")
  print(table(test$species, trainClassifications))
  print("Misclassification rate test")
  print(1 - classificationRate(table(test$species, trainClassifications)))

}

NaiveBayesImpl(train, test)
```

```
## [1] "Confusion matrix train"
##         trainClassifications
##          blue orange
##   Blue     41      8
##   Orange   23     27
## [1] "Misclassification rate train"
## [1] 0.3131313
## [1] "Confusion matrix test"
##         trainClassifications
##          blue orange
##   Blue     26     24
##   Orange   38     11
## [1] "Misclassification rate test"
## [1] 0.6262626
```

The result is very poor, not to any surprise however. Naive Bayes makes the strong assumption of conditional independence, which in other terms assumes a diagonal covariance matrix. If we look at the covariance matrix, we see that this is clearly not the case. Thus, for this specific problem, it is not surprising at all that Naive Bayes performs poorly. We see a similar behaviour if we use the predefined function `naiveBayes` from the package `e1071`, as seen below.

```
##           FL        RW       CL        CW        BD
```

```
## FL 12.217297  8.158045 24.35668 26.55080 11.822581
## RW  8.158045  6.622078 16.35466 18.23964  7.836659
## CL 24.356677 16.354662 50.67992 55.76138 23.971389
## CW 26.550801 18.239640 55.76138 61.96768 26.091867
## BD 11.822581  7.836659 23.97139 26.09187 11.729065
```

```
library(e1071)
fit = naiveBayes(species ~ FL + RW + CL + CW + BD,data=train)
trainPreds = predict(fit, train)
table(train$species, trainPreds)
```

```
##         trainPreds
##          Blue Orange
##   Blue     36     13
##   Orange   20     31
```

```
1 - classificationRate(table(train$species, trainPreds))
```

```
## [1] 0.33
```

```
testPreds = predict(fit, test)

table(test$species, testPreds)
```

```
##         testPreds
##          Blue Orange
##   Blue     33     18
##   Orange   23     26
```

```
print("Misclassification rate test")
```

```
## [1] "Misclassification rate test"
```

```
1 - classificationRate(table(test$species, testPreds))
```

```
## [1] 0.41
```