HW4 ML

Bijesh Mishra

2/24/2021

Machine Learning Homework 4

Due date: March 01, 2021

Variables: * predictor: eprobe scores * eprobes: Ep1, Ep2, Ep3, Ep4 Ep5, Total.

```
rm(list = ls()) #Clear environment.
setwd("~/Dropbox/OSU/PhD/SemVISp2021/STAT5063ML/Homeworks/hw4")
# Load libraries:
library(MASS)
library(ISLR)
library(class)
library(readxl)
library(readr)
eprobe = read.csv("/Users/bmishra/Dropbox/OSU/PhD/SemVISp2021/STAT5063ML/Data/Eprobe.csv")
eprobe = setNames(eprobe,
                  tolower(names(eprobe)))
attach(eprobe, pos = 2L,
       name = deparse(substitute(eprobe), backtick=FALSE),
       warn.conflicts = F)
eprobe[c(1, 13), ]
```

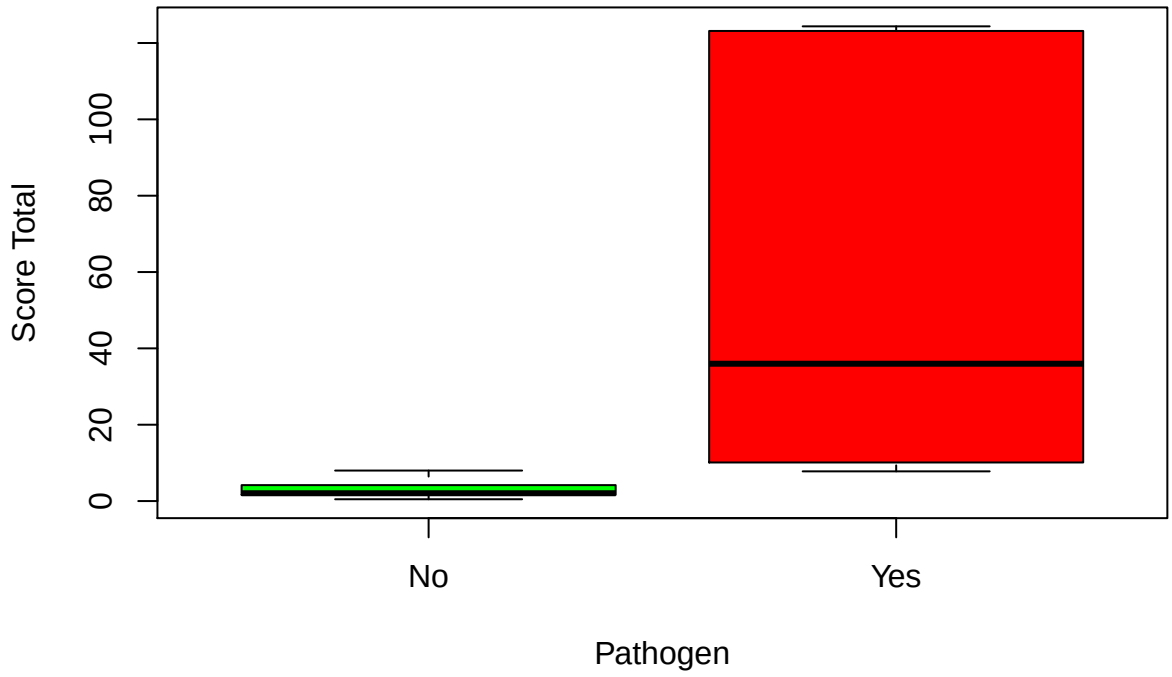
```
##     x pathogen run     ep1      ep2     ep3     ep4      ep5    total
## 1   1        Y   1 3.36751 1.196677 1.99751 2.00001 1.768343 10.33005
## 13 13        N   1 0.82251 0.200010 0.36501 0.67001 0.375010  2.43255
```

```
# View(eprobe)
```

1: Plotting and preliminary analysis:

Q1.a) Box plot: Total score vs. Pathogen. Which would work better: LDA or QDA?

```
boxplot(total ~ pathogen,
        data = eprobe,
        main = "Pathogen Count Score",
        col = c("green", "red"),
        names = c("No", "Yes"),
        ann = T,
        xlab = "Pathogen",
        ylab = "Score Total")
```



Answer: Quadratic Discriminant analysis(QDA) works better. Because, the Linear Discriminant analysis (LDA) assumes that the distribution function (probability densities) for each class is normally distributed (Gaussian distribution) and different classes share same variance-covariance matrix and, thus, we can use same variance-covariance matrix for each class. But QDA assumes that variance-covariance matrix can be differnet for each class. So, we estimate the convariance matrix seperately for each K classes. The covariance matrix $[\sum_k]$ is not identical so we have to keep quadratic term. In the figure above, this is not normally distributed. Thus we have to estimate variance-covariance matrix seperately for each classes k; K = 2.

Q1.b: Get t-test p-value for LDA and a t-test p-value for QDA to determine if $H_o : \mu_1 = \mu_2$ where μ_1 and μ_2 are total scores among pathogens and non-pathogen populations.

```
t.test(total ~ pathogen,
       var.equal = T, # Variances equal: LDA
       conf.level = 0.95,
       alternative = c("two.sided"),
       data = eprobe)
```

```
##
##  Two Sample t-test
##
## data:  total by pathogen
## t = -3.1807, df = 24, p-value = 0.004024
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -102.05469  -21.73182
## sample estimates:
## mean in group N mean in group Y
##        3.017029       64.910281
```

Answer: LDA: $H_o : \mu_1 = \mu_2$ and $H_a : \mu_1 \neq \mu_2$. Conclusion: Since p = 0.004024, we reject H_o in favor of H_a and concluded that the true differences in score means is not equal to zero between pathogen and

non-pathogen population.

```
t.test(total ~ pathogen,
       var.equal = F, # Variances not equal: LDA
       conf.level = 0.95,
       alternative = c("two.sided"),
       data = eprobe)
```

```
##
##  Welch Two Sample t-test
##
## data:  total by pathogen
## t = -4.8162, df = 17.168, p-value = 0.000157
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -88.9864 -34.8001
## sample estimates:
## mean in group N mean in group Y
##        3.017029       64.910281
```

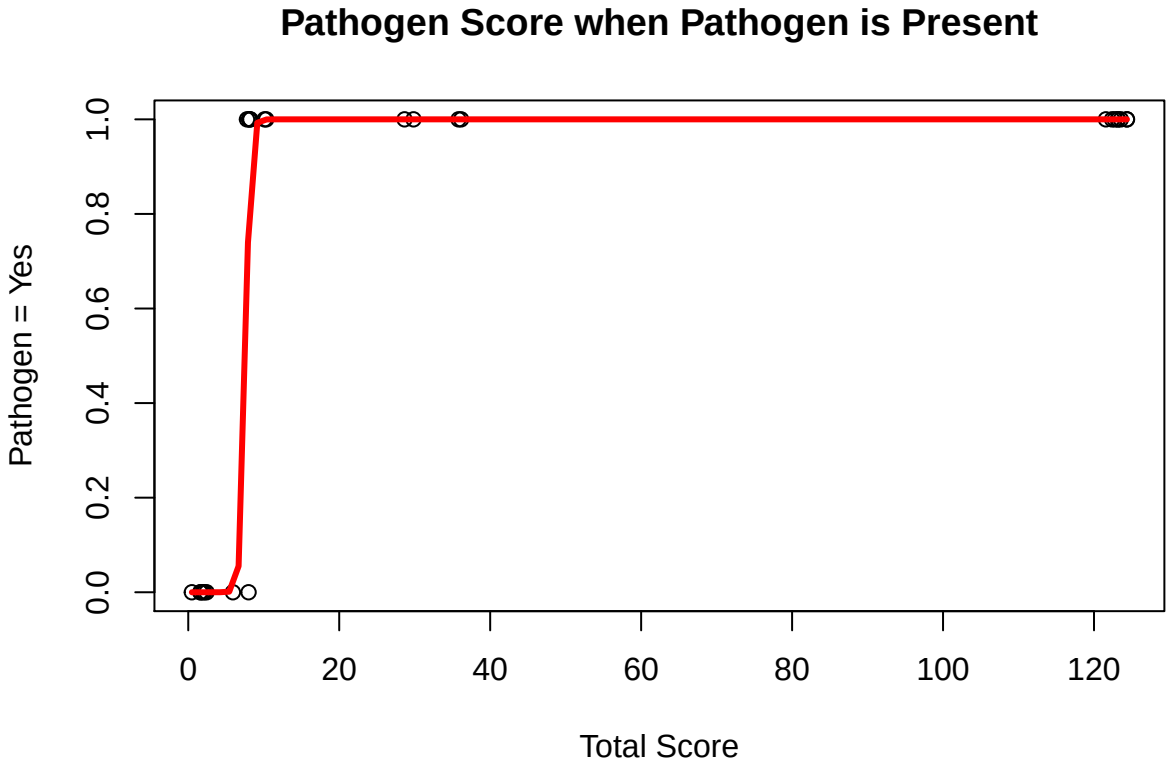
Answer: QDA: $H_o : \mu_1 = \mu_2$ and $H_a : \mu_1 \neq \mu_2$. Conclusion: Since p = 0.000157, we reject H_o in favor of H_a and concluded that the true differences in score means is not equal to zero between pathogen and non-pathogen population.

Q1.c: Construct a plot with Y = I(Pathogens = "Yes") on y-axis and total score on x-axis with logistic regression curve superimposed. Does it appears that pathogens can be discriminated from controls using total scores based on this plot? Interpret p-value for logistic model.

```
logitq1c = glm(pathogen ~ total, family = "binomial", data = eprobe)
```

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

```
plot(total, I(pathogen == "Y"),
     xlab = "Total Score",
     ylab = "Pathogen = Yes",
     pch = 1, col = 1,
     main = "Pathogen Score when Pathogen is Present")
b0 = -23.738
b1 = 3.134
curve(exp(b0 + b1*x)/(1+exp(b0 + b1*x)), add = TRUE, col = "red", lwd = 3, lty = 1)
```



Answer: When total score is close to zero, it is hard to discreminate as there is some overlapping in the chart between pathogen ~ 0 and pathogen ~ 1.

```
summary(logitq1c)
```

```
##
## Call:
## glm(formula = pathogen ~ total, family = "binomial", data = eprobe)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.75339  -0.00009   0.00000   0.00000   0.92044
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -23.738     37.528  -0.633    0.527
## total          3.134      4.700   0.667    0.505
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 32.0966  on 25  degrees of freedom
## Residual deviance:  4.9002  on 24  degrees of freedom
## AIC: 8.9002
##
## Number of Fisher Scoring iterations: 15
```

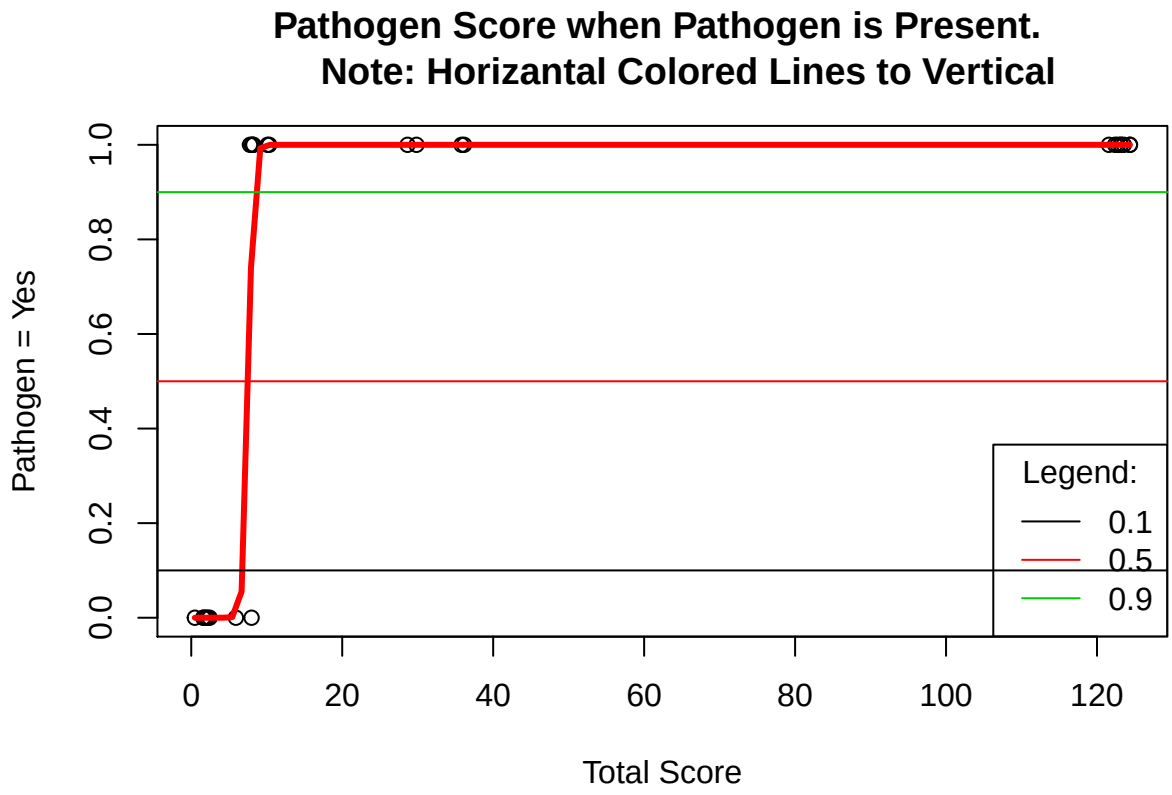
Answer: Since p-value > 0.05 for total, total is not associated with the pathogen count score at 95% CI.

2: Logistic Regression on total score:

Q2.a) Find values for X such that p(X) = 0.1, 0.5, 0.9 and add them to your plot. Hint p(X) = c $\Longleftrightarrow$

logit(p(X)) = logit(c). Use a different line type and for each line and add a key to your plot that identifies each line.

```
logit2a = logitq1c
l2acf = coef(logit2a)
l2a.prob = exp(l2acf[1] + l2acf[2]*total)/(1 + exp(l2acf[1] + l2acf[2]*total))
plot(total, I(pathogen == "Y"),
     xlab = "Total Score",
     ylab = "Pathogen = Yes",
     pch = 1, col = 1,
     main = "Pathogen Score when Pathogen is Present.
     Note: Horizantal Colored Lines to Vertical")
# b0 = -23.738
# b1 = 3.134
curve(exp(b0 + b1*x)/(1+exp(b0 + b1*x)), add = TRUE, col = "red", lwd = 3, lty = 1)
abline(h = 0.1, lwd = 1, lty = 1, col = 1)
abline(h = 0.5, lwd = 1, lty = 1, col = 2)
abline(h = 0.9, lwd = 1, lty = 1, col = 3)
legend("bottomright",
       title = "Legend:",
       col = c( 1, 2, 3),
       lty = c(1, 1, 1),
       legend = c("0.1", "0.5", "0.9"))
```



Q2.b) Get a table that contains the training error, the training TPR and the training FPR for a rule that classifies Y as a pathogen if p(x) > c for each value of c above.

```
prediction1 = rep("N", length(pathogen))
prediction1[predict(logit2a) > log(.1/(1-.1))] = "Y" # p(x) = 0.1
table1 = table(prediction1, pathogen) # p(x) = 0.1
```

```
proportion1 = round(prop.table(table(prediction1, pathogen),2),3) # p(x) = 0.1
trr0.1 = mean(prediction1 != pathogen) # p(x) = 0.1 #Training Error Rate:
TPR.2b1 = proportion1[2,2]/ sum(proportion1[1,2], proportion1[2,2]) #TPR
FPR.2b1 = proportion1[2,1]/ sum(proportion1[1,1], proportion1[2,1]) #FPR
proportion1
```

```
##            pathogen
## prediction1     N     Y
##           N 0.875 0.000
##           Y 0.125 1.000
```

```
trr0.1
```

```
## [1] 0.03846154
```

```
TPR.2b1
```

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

```
FPR.2b1
```

```
## [1] 0.125
```

```
prediction2 = rep("N", length(pathogen))
prediction2[predict(logit2a) > log(.5/(1-.5))] = "Y" # p(x) = 0.5
table2 =  table(prediction2, pathogen) # p(x) = 0.5
proportion2 = round(prop.table(table(prediction2, pathogen),2),3) # p(x) = 0.5
trr0.5 = mean(prediction2 != pathogen) # p(x) = 0.5 #Training Error Rate:
TPR.2b2 = proportion2[2,2]/ sum(proportion2[1,2], proportion2[2,2]) #TPR
FPR.2b2 = proportion2[2,1]/ sum(proportion2[1,1], proportion2[2,1]) #FPR
proportion2
```

```
##            pathogen
## prediction2     N     Y
##           N 0.875 0.000
##           Y 0.125 1.000
```

```
trr0.5
```

```
## [1] 0.03846154
```

```
TPR.2b2
```

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

```
FPR.2b2
```

```
## [1] 0.125
```

```
prediction3 = rep("N", length(pathogen))
prediction3[predict(logit2a) > log(.9/(1-.9))] = "Y" # p(x) = 0.9
table3 = table(prediction3, pathogen) # p(x) = 0.9
proportion3 = round(prop.table(table(prediction3, pathogen),2),3) # p(x) = 0.9
trr0.9 = mean(prediction3 != pathogen) # p(x) = 0.9 #Training Error Rate
TPR.2b3 = proportion3[2,2]/ sum(proportion3[1,2], proportion3[2,2]) #TPR
FPR.2b3 = proportion3[2,1]/ sum(proportion3[1,1], proportion3[2,1]) #FPR
trr0.9
```

```
## [1] 0.1538462
```

```
proportion3
```

```
##            pathogen
## prediction3     N     Y
##           N 1.000 0.222
##           Y 0.000 0.778
```

```
TPR.2b3
```

```
## [1] 0.778
```

```
FPR.2b3
```

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

```
probability = c(0.1, 0.5, 0.9)
Train.Err = c((table1[1,2] + table1[2,1]),
              (table2[1,2] + table2[2,1]),
              (table3[1,2] + table3[2,1]))
TPR = c(TPR.2b1, TPR.2b2, TPR.2b3)
FPR = c(FPR.2b1, FPR.2b2, FPR.2b3)
TRR = c(trr0.1, trr0.5, trr0.9)
as.data.frame(cbind(probability, Train.Err, TPR, FPR, TRR))
```

```
##   probability Train.Err   TPR   FPR        TRR
## 1         0.1         1 1.000 0.125 0.03846154
## 2         0.5         1 1.000 0.125 0.03846154
## 3         0.9         4 0.778 0.000 0.15384615
```

3: Linear Discriminant Analysis on total:

Q3.a) Get the estimated Bayes decision boundary (you may assume $\pi_1 = \pi_2$) and interpret the coefficient in L(x).

```
lda.q3 = lda(pathogen~total, data = eprobe)
lda.q3
```

```
## Call:
## lda(pathogen ~ total, data = eprobe)
##
## Prior probabilities of groups:
##         N         Y
## 0.3076923 0.6923077
##
## Group means:
##       total
## N  3.017029
## Y 64.910281
##
## Coefficients of linear discriminants:
##              LD1
## total 0.02183658
```

```
# Bayes Boundary:
q3.b1 = lda.q3$scaling[1]
var.q3 = (lda.q3$means[2] - lda.q3$means[1])/q3.b1
q3.b0 = log(lda.q3$prior[2]) -
  log(lda.q3$prior[1]) +
  (lda.q3$means[1]^2/(2*var.q3)) -
  (lda.q3$means[2]^2/(2*var.q3))
```

```
bayes.boundary = -(q3.b0/q3.b1)
data.frame(q3.b0, q3.b1, bayes.boundary)
```

```
##        q3.b0      q3.b1 bayes.boundary
## Y 0.06928026 0.02183658       -3.17267
```

Answer: Bayes decision boundary is -0.3152. LDA output indicates that $\hat{\pi}_{no} = 0.3077$ and $\hat{\pi}_{yes} = 0.6923$. i.e. 30.77 % of the training observations corresponds to no total pathogen counts and 69.23% corresponds to the some amount of pathonge counts. Group means are means of class Y and Class N in total and used by LDA as estimates of μ_k. The coefficient of linear determinants [L(x)] is a linear combination of variables (Here, only total), which is used to form decision rule. If 0.022*total is large then pathogen count is Y, otherwise N.

Q3.b) Get the training error and the training TPR and FPR.

```
lda.predict = predict(lda.q3, new.data = eprobe)
names(lda.predict)
```

```
## [1] "class"     "posterior" "x"
```

```
# cbind(lda.predict$x, lda.predict$posterior, lda.predict$class, eprobe$pathogen)
ldamat = table(lda.predict$class, eprobe$pathogen)
lda.trr = mean(lda.predict$class != eprobe$pathogen) ##training error rate
lda.prop = round(prop.table(table(predicted = lda.predict$class,
                                  truth = eprobe$pathogen), 2), 3)
TPR.lda = ldamat[2,2]/(ldamat[1,2]+ldamat[2,2])
FPR.lda = ldamat[2,1]/(ldamat[1,1]+ldamat[2,1])

TPR.lda ##True Positive Rate
```

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

```
FPR.lda ##False Positive Rate
```

```
## [1] 0.125
```

```
ldamat #LDA Matrix
```

```
##
##      N  Y
##   N  7  0
##   Y  1 18
```

```
TrErr.lda = c((ldamat[1,2] + ldamat[2,1]))
lda.prop #LDA Proportion.
```

```
##           truth
## predicted     N     Y
##         N 0.875 0.000
##         Y 0.125 1.000
```

```
TrErr.lda # Training Error
```

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

```
lda.trr # Training Error Rate
```

```
## [1] 0.03846154
```

- True Positive Rate = 1
- False Positive Rate = 0.125

- Training Error = 1
- Training Error Rate = 0.03846

Q4: Quadratic discriminant analysis on total: Get the training error and training TPR and FPR.

```
qda.q4 = qda(pathogen~total, data = eprobe)
table(predict(qda.q4)$class, pathogen)
```

```
##    pathogen
##      N  Y
##   N  8  4
##   Y  0 14
```

```
qda.predict = predict(qda.q4, new.data = eprobe)
# names(qda.predict)
# cbind(qda.predict$x, qda.predict$posterior, qda.predict$class, eprobe$pathogen)
qdamat = table(qda.predict$class, eprobe$pathogen)
qda.trr = mean(qda.predict$class != eprobe$pathogen) ##training error rate
qda.prop = round(prop.table(table(predicted = qda.predict$class,
                                  truth = eprobe$pathogen), 2), 3)
TPR.qda = qdamat[2,2]/(qdamat[1,2]+qdamat[2,2])
FPR.qda = qdamat[2,1]/(qdamat[1,1]+qdamat[2,1])
# qdamatrix #QDA Matrix
qda.q4
```

```
## Call:
## qda(pathogen ~ total, data = eprobe)
##
## Prior probabilities of groups:
##         N         Y
## 0.3076923 0.6923077
##
## Group means:
##       total
## N  3.017029
## Y 64.910281
```

```
qdamat
```

```
##
##      N  Y
##   N  8  4
##   Y  0 14
```

```
TrErr.qda = c((qdamat[1,2] + qdamat[2,1])) #Training Error
TrErr.qda
```

```
## [1] 4
```

```
qda.prop #QDA Proportion.
```

```
##          truth
## predicted     N     Y
##         N 1.000 0.222
##         Y 0.000 0.778
```

```
TPR.qda ##True Positive Rate
```

```
## [1] 0.7777778
```

```
FPR.qda ##False Positive Rate
```

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

```
qda.trr # Training Error Rate
```

```
## [1] 0.1538462
```

- True Positive Rate = 0.778
- False Positive Rate = 0
- Training Error = 4
- Training Error Rate = 0.154

5: LDA on Ep1, Ep2, . . . , Ep5.

Q5.a): Report the coefficients for the linear discriminant function if discriminating between populations with Ep1, . . . , Ep5 and determine which Eprobe best discriminates pathogens from non-pathogens by commenting on the coefficients.

```
lda5a = lda(pathogen ~ ep1 + ep2 + ep3 + ep4 + ep5)
lda5a
```

```
## Call:
## lda(pathogen ~ ep1 + ep2 + ep3 + ep4 + ep5)
##
## Prior probabilities of groups:
##         N         Y
## 0.3076923 0.6923077
##
## Group means:
##          ep1        ep2      ep3      ep4      ep5
## N  0.8453225  0.3466767  0.63376  0.71751  0.47376
## Y 12.3613989 16.3611211 12.00682 12.29149 11.88945
##
## Coefficients of linear discriminants:
##             LD1
## ep1  0.92266622
## ep2  0.05688198
## ep3  2.07598126
## ep4 -0.88491256
## ep5 -1.96940218
```

```
# plot(lda5a)
```

Answer: The coefficients for respective Eprobes are given above. Looking at the coefficients, ep3 best descriminates the pathogen from non-pathogen as the coefficents is much higher and thus makes pathogen counts much higher which can help to classify count as Y.

Q5.b): Get the training error and the training TPR and FPR.

```
table(predict(lda5a)$class, pathogen)
```

```
##    pathogen
##      N  Y
##   N  6  2
##   Y  2 16
```

```
lda5a.predict = predict(lda5a, new.data = eprobe)
# names(lda5a.predict)
```

```
# cbind(lda5a.predict$x, lda5a.predict$posterior, lda5a.predict$class, eprobe$pathogen)
lda5amat = table(lda5a.predict$class, eprobe$pathogen)
lda5a.trr = mean(lda5a.predict$class != eprobe$pathogen) ##training error rate
lda5a.prop = round(prop.table(table(predicted = lda5a.predict$class,
                                    truth = eprobe$pathogen), 2), 3)
TPR.lda5a = lda5amat[2,2]/(lda5amat[1,2] + lda5amat[2,2])
FPR.lda5a = lda5amat[2,1]/(lda5amat[1,1] + lda5amat[2,1])
lda5amat
```

```
##
##      N  Y
##   N  6  2
##   Y  2 16
```

```
TrErr.lda5a = c((lda5amat[1,2] + lda5amat[2,1])) #Training Error
TrErr.lda5a #Training Error
```

```
## [1] 4
```

```
lda5a.prop #LDA Proportion.
```

```
##          truth
## predicted     N     Y
##         N 0.750 0.111
##         Y 0.250 0.889
```

```
TPR.lda5a ##True Positive Rate
```

```
## [1] 0.8888889
```

```
FPR.lda5a ##False Positive Rate
```

```
## [1] 0.25
```

```
lda5a.trr # Training Error Rate
```

```
## [1] 0.1538462
```

- True Positive Rate = 0.889
- False Positive Rate = 0.25
- Training Error = 4
- Training Error Rate = 0.154

Q6: KNN on total:

Q6.a): Get the training error and training TPR and FPR for k = 5 nearest neighbors using total as a predictor.

```
knnq6 = knn(scale(total), scale(total), pathogen, k = 5)
knnq6mat = table(knnq6, pathogen)
TPR.knnq6 = knnq6mat[2,2]/(knnq6mat[1,2] + knnq6mat[2,2])
FPR.knnq6 = knnq6mat[2,1]/(knnq6mat[1,1] + knnq6mat[2,1])
TrErr.knn = c((knnq6mat[1,2] + knnq6mat[2,1])) #Training Error
trr.knn = mean((knnq6 != eprobe$pathogen)) #Training Error Rate
TPR.knnq6 ##True Positive Rate
```

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

```
FPR.knnq6 ##False Positive Rate
```

```
## [1] 0.25
```

```
knnq6mat #KNN Matrix
```

```
##      pathogen
## knnq6  N  Y
##     N  6  0
##     Y  2 18
```

```
TrErr.knn #Training Error
```

```
## [1] 2
```

```
trr.knn
```

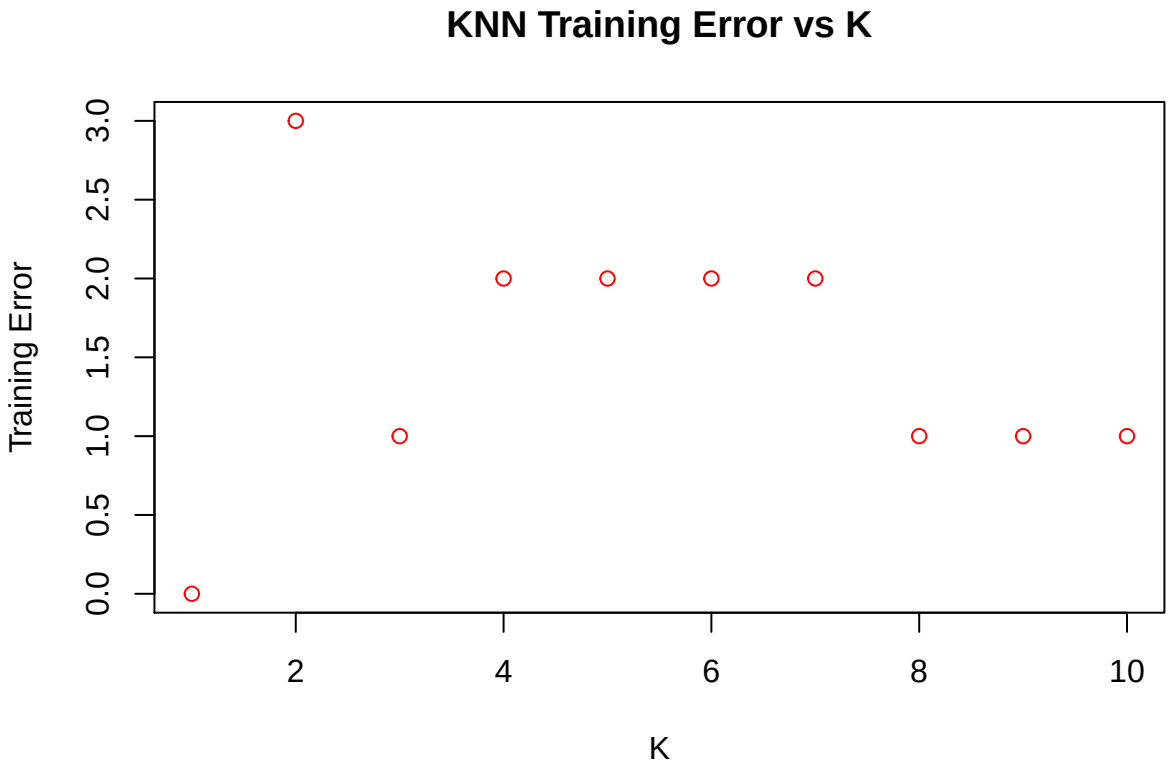
```
## [1] 0.07692308
```

- True Positive Rate = 1
- False Positive Rate = 0.25
- Training Error = 2.
- Training Error Rate = 0.07692

Q6.b): Get a plot of training error vs. k for k = 1, 2, . . . , 10 with k on the x axis. What value(s) of k would you anticipate the most bias in the training error estimate? Justify your answer in a sentence or two.

```
set.seed(1)
knnq6b1 = knn(scale(total), scale(total), pathogen, k = 1)
knnq6b1mat = table(knnq6b1, pathogen)
knnq6b2 = knn(scale(total), scale(total), pathogen, k = 2)
knnq6b2mat = table(knnq6b2, pathogen)
knnq6b3 = knn(scale(total), scale(total), pathogen, k = 3)
knnq6b3mat = table(knnq6b3, pathogen)
knnq6b4 = knn(scale(total), scale(total), pathogen, k = 4)
knnq6b4mat = table(knnq6b4, pathogen)
knnq6b5 = knn(scale(total), scale(total), pathogen, k = 5)
knnq6b5mat = table(knnq6b5, pathogen)
knnq6b6 = knn(scale(total), scale(total), pathogen, k = 6)
knnq6b6mat = table(knnq6b6, pathogen)
knnq6b7 = knn(scale(total), scale(total), pathogen, k = 7)
knnq6b7mat = table(knnq6b7, pathogen)
knnq6b8 = knn(scale(total), scale(total), pathogen, k = 8)
knnq6b8mat = table(knnq6b8, pathogen)
knnq6b9 = knn(scale(total), scale(total), pathogen, k = 9)
knnq6b9mat = table(knnq6b9, pathogen)
knnq6b10 = knn(scale(total), scale(total), pathogen, k = 10)
knnq6b10mat = table(knnq6b10, pathogen)
training.error = c((knnq6b1mat[1,2] + knnq6b1mat[2,1]),
                   (knnq6b2mat[1,2] + knnq6b2mat[2,1]),
                   (knnq6b3mat[1,2] + knnq6b3mat[2,1]),
                   (knnq6b4mat[1,2] + knnq6b4mat[2,1]),
                   (knnq6b5mat[1,2] + knnq6b5mat[2,1]),
                   (knnq6b6mat[1,2] + knnq6b6mat[2,1]),
                   (knnq6b7mat[1,2] + knnq6b7mat[2,1]),
                   (knnq6b8mat[1,2] + knnq6b8mat[2,1]),
                   (knnq6b9mat[1,2] + knnq6b9mat[2,1]),
                   (knnq6b10mat[1,2] + knnq6b10mat[2,1]))
k = c(1:10)
plot(k, training.error, col = "red", xlab = "K",
     ylab = "Training Error",
```

```
     main = "KNN Training Error vs K")
```



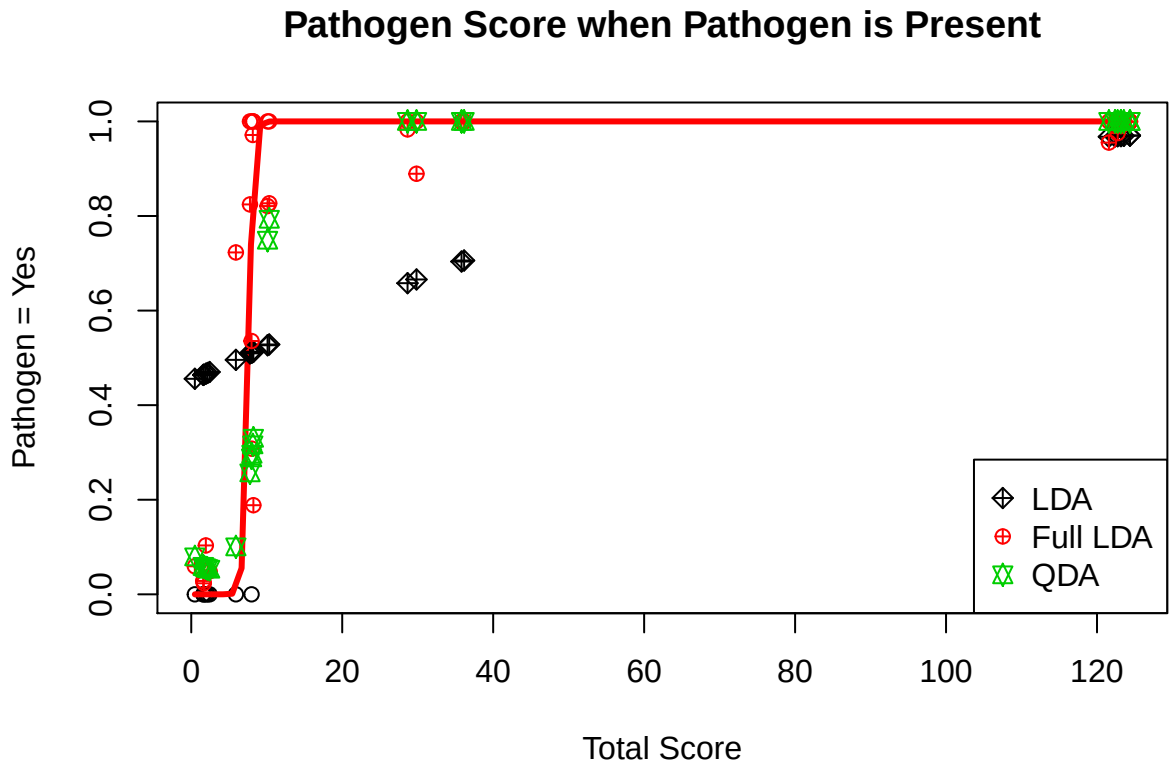
Answer: In this case, K = 2 has highest in the training error estimate. This is because the training error (ie. sum of miscalssification) with K = 2 is highest in the chart above. However, only looking at the K, does not says which has less and more bias because the total bias is related to bias and variance and there is always bias and variances tradeoff in the modeling process. With the increase in K, model become less flexible, the bias increases, variance decreases. Thus I expect 10 to be most biased.

```
#looping:
# set.seed(1)
# errors = NULL
# for(i in 1:10){
#   eprobe_KNN = knn(scale(total),
#                    scale(total),
#                    eprobe$pathogen, k = i)
#   errors[i] = round(mean(eprobe_KNN != pathogen), 3)
# }
# k <- 1:10
# plot(k, errors)
```

Q7: (Grad Students Only). Reconstruct the plot with Total score on the X axis and Pathogen on Y axis with the logistic regression curve superimposed. Add $(x_i, p_2(x_i))$ to the plot using the points function, where $p_2(x_i)$ is the probability of default for the LDA on the total scores, full LDA, and QDA on total using different plotting characters and colors for each set of probabilities. Add a legend to the plot to identify probabilities. Hint: You'll want to run something like the code below to get one set of points: pred.fullLDA<-predict(my.full.LDA.model) points(total, preds.fullLDA$posterior[,2], pch = 2, col = 2)

```
# Logistic Regression:
plot(total, I(pathogen == "Y"),
     xlab = "Total Score",
     ylab = "Pathogen = Yes",
```

```
     pch = 1,
     col = as.factor(pathogen),
     main = "Pathogen Score when Pathogen is Present")
b0 = -23.738
b1 = 3.134
curve(exp(b0 + b1*x)/(1+exp(b0 + b1*x)),
      add = TRUE, col = "red",
      lwd = 3, lty = 1)
points(total, lda.predict$posterior[,2],
       pch=9, col=9) # LDA
points(total, lda5a.predict$posterior[,2],
       pch=10, col=10) # Full LDA with 5 Eps.
points(total, qda.predict$posterior[,2],
       pch=11, col=11) #QDA
legend("bottomright", col = c(9, 10, 11),
       pch = c(9, 10, 11),
       legend = c("LDA", "Full LDA", "QDA"))
```



Q8: (Grad Students Only) Consider an LDA for 2 predictors. Denote the estimated covariance matrix by S = (s_{ij}) and denote the mean vectors by $\bar{x}_1^T = (\bar{x}_{11}, \bar{x}_{12})$ and $\bar{x}_2^T = (\bar{x}_{21}, \bar{x}_{22})$. Find expressions for $\hat{\beta}_1 and \hat{\beta}_2$ in the linear discriminant function $L(x) = \hat{\beta}_o + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2$. How does $\hat{\beta}_1$ compare to the univariate estimate of $\hat{\beta}_1$ when $s_{12} = 0$?

```
# Theoritical question: Next Page:
```