Homework 5

Daniel Tshiani

2025-06-01

1

a

```r
library(tibble)

df <- data.frame(
  X1 = c(0, 2, 0, 0, 1, 1),
  X2 = c(3, 0, 1, 1, 0, 1),
  X3 = c(0, 0, 3, 2, 1, 1)
)

Y = c("Red", "Red", "Red", "Green", "Green", "Red")

euclidean_distances <- apply(df, 1, function(row) {
  sqrt(sum(row^2))
})

df$Y <- Y

df$DistanceFromOrigin <- euclidean_distances

print(df)
```

```
##   X1 X2 X3     Y DistanceFromOrigin
## 1  0  3  0   Red           3.000000
## 2  2  0  0   Red           2.000000
## 3  0  1  3   Red           3.162278
## 4  0  1  2 Green           2.236068
## 5  1  0  1 Green           1.414214
## 6  1  1  1   Red           1.732051
```

b

when K=1 the 5th observation has the smallest distance so we would predict green.

c

when K=3 the 5th, 6th, and 2nd observations have the smallest distances. there responses are green, red, and red. So we would predict red.

d

I would expect the best value for k to be small because a smalll K would accomadate more flexibility.

2

a

on the training set i would expect QDA is perform equally as good as LDA, if not I would expect QDA to preform better because its more flexible. so it will be able to caputre the linear patter plus some added noise in the training set.

However in the test set, I would expect LDA to preform better because if the Bayes decision boundary is linear, QDA would capture the noise in the training data.that would lead to higher variance in the test set for QDA.

b

if its non-linear, i would expect QDA to preform better on both training and testing data because LDA assumes linear boundries

c

I would expect QDA relative to LDA to improve because QDA is more flexible. when the number of observations is smaller than LDA would be better because its less prone to overfitting but when the number of observations gets larger LDA can be too restrictive.

d

False. expecially when the number of observations is small, QDA can capture too much of the noise and represent the training data too closely. this would increase the variance of the test error rate.

3

```
x <- 4
mu_yes <- 10
mu_no <- 0
sigma <- 6
p_yes <- 0.8
p_no <- 0.2

f_yes <- (1 / (sqrt(2 * pi) * sigma)) * exp(-((x - mu_yes)^2) / (2 * sigma^2))

f_no <- (1 / (sqrt(2 * pi) * sigma)) * exp(-((x - mu_no)^2) / (2 * sigma^2))

posterior_yes <- (f_yes * p_yes) / (f_yes * p_yes + f_no * p_no)

posterior_yes
```

```
## [1] 0.7518525
```

about 75%

4

```r
library(ISLR)
library(tibble)

Direction <- Weekly[["Direction"]]
df <- tibble(Direction)

df$Lag1 <- Weekly[["Lag1"]]
df$Lag2 <- Weekly[["Lag2"]]
df$Lag3 <- Weekly[["Lag3"]]
df$Lag4 <- Weekly[["Lag4"]]
df$Lag5 <- Weekly[["Lag5"]]
df$Year <- Weekly[["Year"]]
df$Volume <- Weekly[["Volume"]]
df$Today <- Weekly[["Today"]]
```

b

```r
logit_model <- glm(Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume, data = df, family = binomial)

summary(logit_model)
```

```
##
## Call:
## glm(formula = Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 +
##     Volume, family = binomial, data = df)
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.26686    0.08593   3.106   0.0019 **
## Lag1        -0.04127    0.02641  -1.563   0.1181
## Lag2         0.05844    0.02686   2.175   0.0296 *
## Lag3        -0.01606    0.02666  -0.602   0.5469
## Lag4        -0.02779    0.02646  -1.050   0.2937
## Lag5        -0.01447    0.02638  -0.549   0.5833
## Volume      -0.02274    0.03690  -0.616   0.5377
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1496.2  on 1088  degrees of freedom
## Residual deviance: 1486.4  on 1082  degrees of freedom
## AIC: 1500.4
##
## Number of Fisher Scoring iterations: 4
```

the intercept and lag2 appear to be statistically significant

c

```r
pred_probs <- predict(logit_model, type = "response")
pred_class <- ifelse(pred_probs > 0.5, "Up", "Down")
conf_matrix <- table(Predicted = pred_class, Actual = df$Direction)
conf_matrix
```

```
##          Actual
## Predicted Down  Up
##      Down   54  48
##      Up    430 557
```

```r
accuracy <- mean(pred_class == df$Direction)
accuracy
```

```
## [1] 0.5610652
```

the matrix is about 56% correct. its struggling because its predicting up whent he actual value is down very often.

d

```r
train <- df$Year <= 2008
test  <- df$Year > 2008

logit_train <- glm(Direction ~ Lag2, data = df, subset = train, family = binomial)
test_probs <- predict(logit_train, newdata = df[test, ], type = "response")

test_pred <- ifelse(test_probs > 0.5, "Up", "Down")

actual_test <- df$Direction[test]

conf_matrix <- table(Predicted = test_pred, Actual = actual_test)
conf_matrix
```

```
##          Actual
## Predicted Down Up
##      Down    9  5
##      Up     34 56
```

```r
accuracy <- mean(test_pred == actual_test)
accuracy
```

```
## [1] 0.625
```

i

```r
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

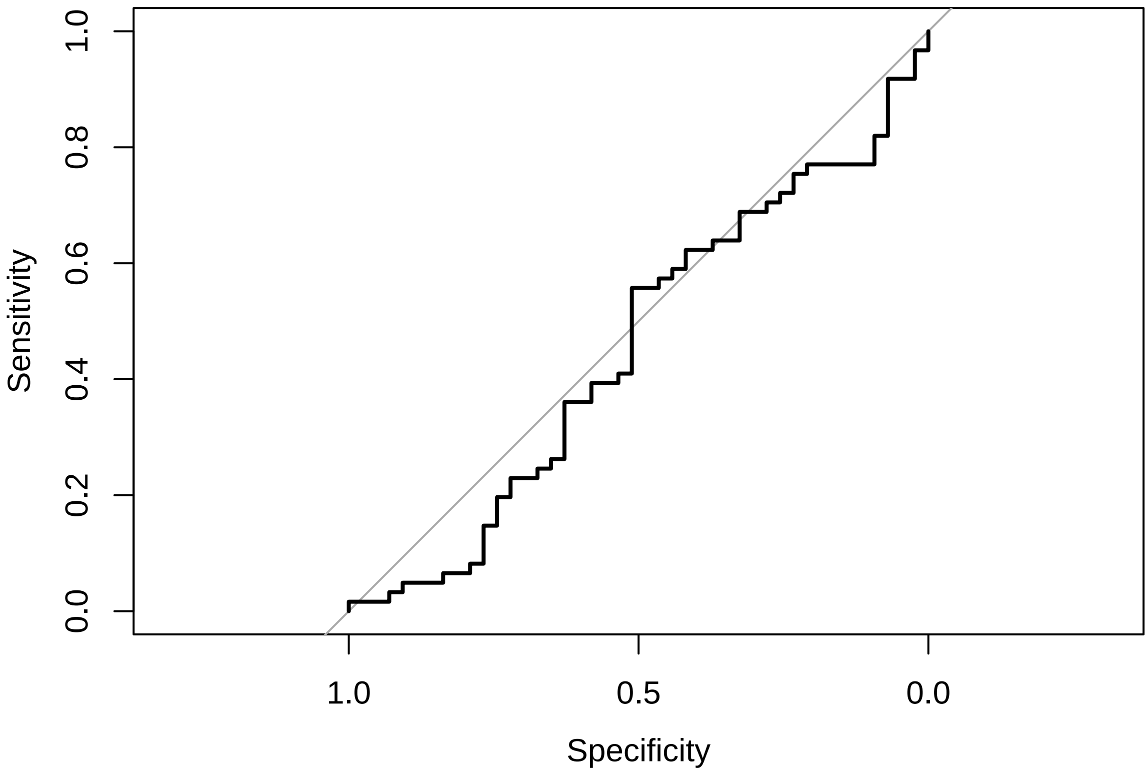
```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```r
actual_binary <- ifelse(actual_test == "Up", 1, 0)
roc_obj <- roc(actual_binary, test_probs)
```

```
## Setting levels: control = 0, case = 1

## Setting direction: controls > cases
```

```r
plot(roc_obj)
```



j

```r
library(class)
library(caret)
```

```
## Loading required package: ggplot2

## Loading required package: lattice
```

```r
lag_vars <- c("Lag1", "Lag2", "Lag3", "Lag4", "Lag5")

train_index <- df$Year <= 2008
test_index  <- df$Year > 2008

train_X <- scale(df[train_index, lag_vars])
test_X  <- scale(df[test_index, lag_vars])

train_Y <- df$Direction[train_index]
test_Y  <- df$Direction[test_index]
```

```r
set.seed(1)
ks <- 1:10
```

```r
acc <- numeric(length(ks))

for (i in ks) {
  knn_pred <- knn(train = train_X, test = test_X, cl = train_Y, k = i)
  acc[i] <- mean(knn_pred == test_Y)
}

data.frame(k = ks, Accuracy = round(acc, 4))
```

```
##     k Accuracy
## 1   1   0.5096
## 2   2   0.5096
## 3   3   0.5192
## 4   4   0.5385
## 5   5   0.5481
## 6   6   0.5865
## 7   7   0.5000
## 8   8   0.5673
## 9   9   0.5769
## 10 10   0.5769
```

```r
best_k <- ks[which.max(acc)]
best_k
```

```
## [1] 6
```

```r
knn_final <- knn(train = train_X, test = test_X, cl = train_Y, k = best_k)

table(Predicted = knn_final, Actual = test_Y)
```

```
##          Actual
## Predicted Down Up
##      Down   12 14
##      Up     31 47
```

```r
mean(knn_final == test_Y)
```

```
## [1] 0.5673077
```

e

```r
library(MASS)

lda_model <- lda(Direction ~ Lag2, data = df, subset = train_index)

lda_pred <- predict(lda_model, df[test_index, ])

conf_matrix <- table(Predicted = lda_pred$class, Actual = df$Direction[test_index])
print(conf_matrix)
```

```
##          Actual
## Predicted Down Up
##      Down    9  5
##      Up     34 56
```

```r
accuracy <- mean(lda_pred$class == df$Direction[test_index])
print(accuracy)
```

```
## [1] 0.625
```

f

```r
qda_model <- qda(Direction ~ Lag2, data = df, subset = train_index)

qda_pred <- predict(qda_model, df[test_index, ])

conf_matrix <- table(Predicted = qda_pred$class, Actual = df$Direction[test_index])
print(conf_matrix)
```

```
##          Actual
## Predicted Down Up
##      Down    0  0
##      Up     43 61
```

```r
accuracy <- mean(qda_pred$class == df$Direction[test_index])
print(accuracy)
```

```
## [1] 0.5865385
```

g

```r
set.seed(1)
knn_pred <- knn(train = train_X, test = test_X, cl = train_Y, k = 1)

conf_matrix_knn1 <- table(Predicted = knn_pred, Actual = test_Y)
print(conf_matrix_knn1)
```

```
##          Actual
## Predicted Down Up
##      Down   21 29
##      Up     22 32
```

```r
accuracy_knn1 <- mean(knn_pred == test_Y)
print(accuracy_knn1)
```

```
## [1] 0.5096154
```

h

the logistic model, and the lda model had a prediction aaccuracy of 62.5 so they provided the best fit.