HW10

Minh Luc, Devin Pham, Kyle Moore

Friday of Week 10, 06/03/2022

Contents

We all contributed equally for this homework.

Question 0

Member 1:

- Name: Minh Luc
- Student ID: A17209607

Member 2:

- Name: Kyle Moore
- Student ID: A14271413

Member 3:

- Name: Devin Pham
- Student ID: A17198936

Question 1

- **(a)**

```r
library(ISLR2)
library(MASS)
```

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:ISLR2':
##
##     Boston
```

```r
auto_data <- Auto

mpg01 <- ifelse(auto_data$mpg > median(auto_data$mpg), 1, 0)

auto_data <- data.frame(auto_data, mpg01)

head(auto_data)
```

```
##   mpg cylinders displacement horsepower weight acceleration year origin
## 1  18         8          307        130   3504         12.0   70      1
## 2  15         8          350        165   3693         11.5   70      1
## 3  18         8          318        150   3436         11.0   70      1
## 4  16         8          304        150   3433         12.0   70      1
## 5  17         8          302        140   3449         10.5   70      1
## 6  15         8          429        198   4341         10.0   70      1
##                        name mpg01
## 1 chevrolet chevelle malibu     0
## 2         buick skylark 320     0
## 3        plymouth satellite     0
## 4             amc rebel sst     0
## 5               ford torino     0
## 6          ford galaxie 500     0
```

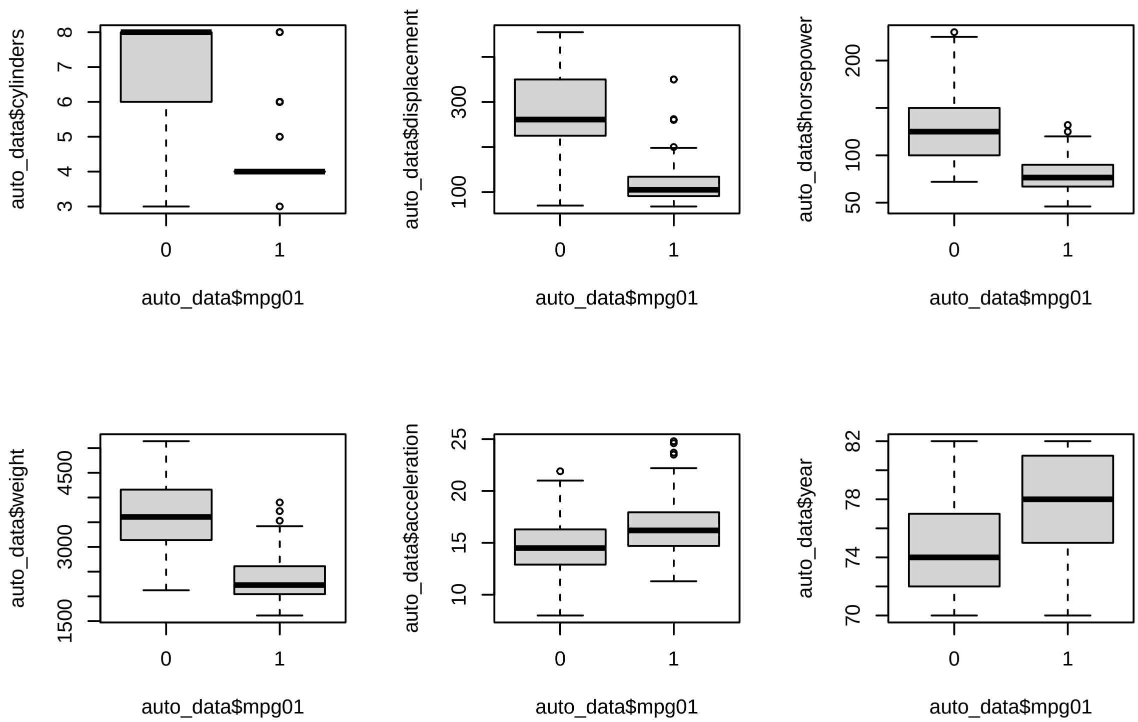
- **(b)**

```r
cor(subset(auto_data, select = -c(name) ))
```

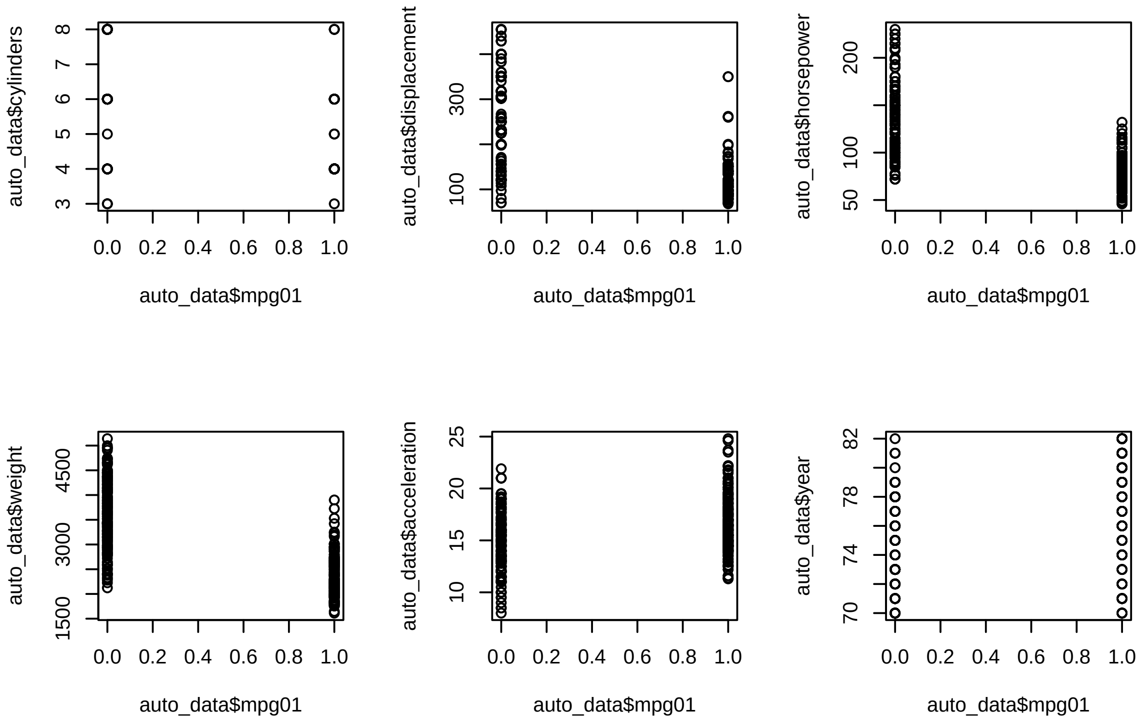
```
##                     mpg  cylinders displacement horsepower     weight
## mpg           1.0000000 -0.7776175   -0.8051269 -0.7784268 -0.8322442
## cylinders    -0.7776175  1.0000000    0.9508233  0.8429834  0.8975273
## displacement -0.8051269  0.9508233    1.0000000  0.8972570  0.9329944
## horsepower   -0.7784268  0.8429834    0.8972570  1.0000000  0.8645377
## weight       -0.8322442  0.8975273    0.9329944  0.8645377  1.0000000
## acceleration  0.4233285 -0.5046834   -0.5438005 -0.6891955 -0.4168392
## year          0.5805410 -0.3456474   -0.3698552 -0.4163615 -0.3091199
## origin        0.5652088 -0.5689316   -0.6145351 -0.4551715 -0.5850054
## mpg01         0.8369392 -0.7591939   -0.7534766 -0.6670526 -0.7577566
##              acceleration       year     origin      mpg01
## mpg             0.4233285  0.5805410  0.5652088  0.8369392
## cylinders      -0.5046834 -0.3456474 -0.5689316 -0.7591939
## displacement   -0.5438005 -0.3698552 -0.6145351 -0.7534766
## horsepower     -0.6891955 -0.4163615 -0.4551715 -0.6670526
## weight         -0.4168392 -0.3091199 -0.5850054 -0.7577566
```

```
## acceleration    1.0000000  0.2903161  0.2127458  0.3468215
## year            0.2903161  1.0000000  0.1815277  0.4299042
## origin          0.2127458  0.1815277  1.0000000  0.5136984
## mpg01           0.3468215  0.4299042  0.5136984  1.0000000
```

```r
par(mfrow=c(2,3))
boxplot(auto_data$cylinders~auto_data$mpg01)
boxplot(auto_data$displacement~auto_data$mpg01)
boxplot(auto_data$horsepower~auto_data$mpg01)
boxplot(auto_data$weight~auto_data$mpg01)
boxplot(auto_data$acceleration~auto_data$mpg01)
boxplot(auto_data$year~auto_data$mpg01)
```




```r
par(mfrow=c(2,3))
plot(auto_data$cylinders~auto_data$mpg01)
plot(auto_data$displacement~auto_data$mpg01)
plot(auto_data$horsepower~auto_data$mpg01)
plot(auto_data$weight~auto_data$mpg01)
plot(auto_data$acceleration~auto_data$mpg01)
plot(auto_data$year~auto_data$mpg01)
```



– The boxplots with the most separation appear to be cylinders, displacement, horsepower, and weight. Acceleration the least useful and year is middling but does show some separation. Scatterplots show similar but less easy to see results. By then confirming with the correlation table, we can see that our intuitions are confirmed, and the order of the useful correlations are cylinders, weight, displacement, then horsepower. I will exclude year and origin in the models.

- **(c)**

```
set.seed(1)

n <- nrow(auto_data)

train_index <- sample(1:n, size = n / 2)

train <- auto_data[train_index,]
test <- auto_data[-train_index,]
```

- **(d)**

```
lda.fit <- lda(mpg01 ~ cylinders + displacement + horsepower + weight, data = train)
lda.fit
```

```
## Call:
## lda(mpg01 ~ cylinders + displacement + horsepower + weight, data = train)
##
## Prior probabilities of groups:
##         0         1
## 0.4795918 0.5204082
##
## Group means:
```

```
##   cylinders displacement horsepower   weight
## 0  6.872340     276.8404  129.40426 3620.723
## 1  4.137255     113.6275   77.92157 2319.118
##
## Coefficients of linear discriminants:
##                        LD1
## cylinders    -0.4495275445
## displacement -0.0080629902
## horsepower    0.0100691240
## weight       -0.0005339524
```

```
lda.pred <- predict(lda.fit, test)
lda.class <- lda.pred$class
table(lda.class, test$mpg01)
```

```
##
## lda.class  0  1
##         0 83  6
##         1 19 88
```

```
accuracy <- mean(lda.class == test$mpg01); accuracy
```

```
## [1] 0.872449
```

```
1 - accuracy
```

```
## [1] 0.127551
```

- Test error for LDA was 12.7551%

- **(e)**

```
qda.fit <- qda(mpg01 ~ cylinders + displacement + horsepower + weight, data = train)
qda.fit
```

```
## Call:
## qda(mpg01 ~ cylinders + displacement + horsepower + weight, data = train)
##
## Prior probabilities of groups:
##         0         1
## 0.4795918 0.5204082
##
## Group means:
##   cylinders displacement horsepower   weight
## 0  6.872340     276.8404  129.40426 3620.723
## 1  4.137255     113.6275   77.92157 2319.118
```

```
qda.pred <- predict(qda.fit, test)
qda.class <- qda.pred$class
table(qda.class, test$mpg01)
```

```
##
## qda.class  0  1
##         0 89 10
##         1 13 84
```

```
accuracy <- mean(qda.class == test$mpg01); accuracy
```

```
## [1] 0.8826531
```

```r
1 - accuracy
```

```
## [1] 0.1173469
```

  - Test error for QDA was 11.73469%

- **(f)**

```r
glm.fit <- glm(mpg01 ~ cylinders + displacement + horsepower + weight, data = train, family = binom
summary(glm.fit)
```

```
##
## Call:
## glm(formula = mpg01 ~ cylinders + displacement + horsepower +
##     weight, family = binomial, data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5768  -0.1043   0.1291   0.3476   2.2602
##
## Coefficients:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)  11.3034129  2.6683207   4.236 2.27e-05 ***
## cylinders    -0.0517713  0.5239294  -0.099   0.9213
## displacement -0.0194725  0.0124725  -1.561   0.1185
## horsepower   -0.0395012  0.0224692  -1.758   0.0787 .
## weight       -0.0013873  0.0009885  -1.403   0.1605
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 271.387  on 195  degrees of freedom
## Residual deviance:  94.438  on 191  degrees of freedom
## AIC: 104.44
##
## Number of Fisher Scoring iterations: 7
```

```r
glm.probs <- predict(glm.fit, newdata = test,type = "response")
glm.pred <- ifelse(glm.probs > 0.5, 1, 0)

table(glm.pred, test$mpg01)
```

```
##
## glm.pred  0  1
##        0 86  8
##        1 16 86
```

```r
accuracy <- mean(glm.pred == test$mpg01); accuracy
```

```
## [1] 0.877551
```

```r
1 - accuracy
```

```
## [1] 0.122449
```

  - Test error for logistic regression was 12.2449%

- **(g)**

```r
summary(glm.fit)$coef
```

```
##                  Estimate   Std. Error     z value     Pr(>|z|)
## (Intercept)  11.303412946 2.6683207382  4.23615227 2.273826e-05
## cylinders    -0.051771323 0.5239294308 -0.09881354 9.212863e-01
## displacement -0.019472458 0.0124724981 -1.56123158 1.184691e-01
## horsepower   -0.039501199 0.0224691580 -1.75801866 7.874433e-02
## weight       -0.001387313 0.0009885384 -1.40339797 1.604982e-01
```

- The β_1 coefficient of cylinders is zero, the test statistic(z value) is -0.0988, and the p-value is .9213. Since our p-value is very large, we do not reject the null, and that is why we cannot assume that β_1 is not 0.

Question 2

- **(a)**

```
oj_data <- OJ

n <- nrow(oj_data)

train_index <- sample(1:n, size = 800)

train <- oj_data[train_index,]
test <- oj_data[-train_index,]
```

- **(b)**

```
library(e1071)

svm.fit <- svm(Purchase ~ ., data = train, kernel = "linear", cost = .01)

summary(svm.fit)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train, kernel = "linear", cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  433
##
##  ( 216 217 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

  – Linearn kernel has a total of 433 support vectors, 216 from CH and 217 from MM

- **(c)**

```
train_pred <- predict(svm.fit, train)
test_pred <- predict(svm.fit, test)

table(train_pred, train$Purchase)
```

```
##
## train_pred  CH  MM
##         CH 426  73
##         MM  64 237
```

```
table(test_pred, test$Purchase)
```

```
##
```

```
## test_pred  CH  MM
##        CH 151  39
##        MM  12  68
```

```
# print training accuracy
train_accuracy <- mean(train_pred == train$Purchase); train_accuracy
```

```
## [1] 0.82875
```

```
# print training error
1 - train_accuracy
```

```
## [1] 0.17125
```

```
# print test accuracy
test_accuracy <- mean(test_pred == test$Purchase); test_accuracy
```

```
## [1] 0.8111111
```

```
# print test error
1 - test_accuracy
```

```
## [1] 0.1888889
```

- **(d)**

```
# tuning to select cost for linear SVM

tune.out <- tune(svm, Purchase ~ ., data = train, kernel = "linear",
                 ranges = list(cost = c(0.01, 0.1, 1, 5, 10)))

summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.1675
##
## - Detailed performance results:
##    cost   error dispersion
## 1  0.01 0.17250 0.03670453
## 2  0.10 0.17250 0.03809710
## 3  1.00 0.16875 0.04177070
## 4  5.00 0.16875 0.03596391
## 5 10.00 0.16750 0.03545341
```

```
bestmod <- tune.out$best.model
summary(bestmod)
```

```
##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = train, ranges = list(cost = c(0.01,
##     0.1, 1, 5, 10)), kernel = "linear")
```

```
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
##
## Number of Support Vectors:  328
##
##  ( 164 164 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

- **(e)**

```
train_pred <- predict(bestmod, train)
test_pred <- predict(bestmod, test)

table(train_pred, train$Purchase)
```

```
##
## train_pred  CH  MM
##         CH 428  65
##         MM  62 245
```

```
table(test_pred, test$Purchase)
```

```
##
## test_pred  CH  MM
##        CH 152  36
##        MM  11  71
```

```
# print training accuracy
train_accuracy <- mean(train_pred == train$Purchase); train_accuracy
```

```
## [1] 0.84125
```

```
# print training error
1 - train_accuracy
```

```
## [1] 0.15875
```

```
# print test accuracy
test_accuracy <- mean(test_pred == test$Purchase); test_accuracy
```

```
## [1] 0.8259259
```

```
# print test error
1 - test_accuracy
```

```
## [1] 0.1740741
```

- **(f)**

```
svm.fit <- svm(Purchase ~ ., data = train, kernel = "radial", cost = .01)
```

```r
summary(svm.fit)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train, kernel = "radial", cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.01
##
## Number of Support Vectors:  623
##
##  ( 313 310 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
train_pred <- predict(svm.fit, train)
test_pred <- predict(svm.fit, test)

table(train_pred, train$Purchase)
```

```
##
## train_pred  CH  MM
##         CH 490 310
##         MM   0   0
```

```r
table(test_pred, test$Purchase)
```

```
##
## test_pred  CH  MM
##        CH 163 107
##        MM   0   0
```

```r
# print training accuracy
train_accuracy <- mean(train_pred == train$Purchase); train_accuracy
```

```
## [1] 0.6125
```

```r
# print training error
1 - train_accuracy
```

```
## [1] 0.3875
```

```r
# print test accuracy
test_accuracy <- mean(test_pred == test$Purchase); test_accuracy
```

```
## [1] 0.6037037
```

```r
# print test error
1 - test_accuracy
```

```
## [1] 0.3962963
```

– Radial basis kernel has a total of 623 support vectors, 313 from CH and 310 from MM

```
# tuning to select cost for radial basis SVM

tune.out <- tune(svm, Purchase ~ ., data = train, kernel = "radial",
                 ranges = list(cost = c(0.01, 0.1, 1, 5, 10)))

summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     5
##
## - best performance: 0.17625
##
## - Detailed performance results:
##    cost   error dispersion
## 1  0.01 0.38750 0.07021791
## 2  0.10 0.18000 0.04005205
## 3  1.00 0.17625 0.04505013
## 4  5.00 0.17625 0.04185375
## 5 10.00 0.17875 0.04126894
```

```
bestmod <- tune.out$best.model
summary(bestmod)
```

```
##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = train, ranges = list(cost = c(0.01,
##     0.1, 1, 5, 10)), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  5
##
## Number of Support Vectors:  331
##
##  ( 171 160 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
train_pred <- predict(bestmod, train)
test_pred <- predict(bestmod, test)

table(train_pred, train$Purchase)
```

```
##
## train_pred  CH  MM
##         CH 448  69
##         MM  42 241
```

```
table(test_pred, test$Purchase)
```

```
##
## test_pred  CH  MM
##        CH 146  44
##        MM  17  63
```

```
# print training accuracy
train_accuracy <- mean(train_pred == train$Purchase); train_accuracy
```

```
## [1] 0.86125
```

```
# print training error
1 - train_accuracy
```

```
## [1] 0.13875
```

```
# print test accuracy
test_accuracy <- mean(test_pred == test$Purchase); test_accuracy
```

```
## [1] 0.7740741
```

```
# print test error
1 - test_accuracy
```

```
## [1] 0.2259259
```

- **(g)**

```
svm.fit <- svm(Purchase ~ ., data = train, kernel = "polynomial",
               degree = 2, cost = .01)

summary(svm.fit)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train, kernel = "polynomial",
##     degree = 2, cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  0.01
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  625
##
##  ( 315 310 )
##
##
## Number of Classes:  2
##
```

```
## Levels:
##  CH MM
```

```r
train_pred <- predict(svm.fit, train)
test_pred <- predict(svm.fit, test)

table(train_pred, train$Purchase)
```

```
##
## train_pred  CH  MM
##         CH 487 288
##         MM   3  22
```

```r
table(test_pred, test$Purchase)
```

```
##
## test_pred  CH  MM
##        CH 159 100
##        MM   4   7
```

```r
# print training accuracy
train_accuracy <- mean(train_pred == train$Purchase); train_accuracy
```

```
## [1] 0.63625
```

```r
# print training error
1 - train_accuracy
```

```
## [1] 0.36375
```

```r
# print test accuracy
test_accuracy <- mean(test_pred == test$Purchase); test_accuracy
```

```
## [1] 0.6148148
```

```r
# print test error
1 - test_accuracy
```

```
## [1] 0.3851852
```

– Polynomial kernel has a total of 625 support vectors, 315 from CH and 310 from MM

```r
# tuning to select cost for polynomial SVM

tune.out <- tune(svm, Purchase ~ ., data = train, kernel = "polynomial",
                 degree = 2, ranges = list(cost = c(0.01, 0.1, 1, 5, 10)))

summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.17125
##
```

```
## - Detailed performance results:
##    cost   error dispersion
## 1  0.01 0.37625 0.08259044
## 2  0.10 0.30875 0.06375136
## 3  1.00 0.19000 0.05737305
## 4  5.00 0.17500 0.04930066
## 5 10.00 0.17125 0.04966904
```

```r
bestmod <- tune.out$best.model
summary(bestmod)
```

```
##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = train, ranges = list(cost = c(0.01,
##     0.1, 1, 5, 10)), kernel = "polynomial", degree = 2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  10
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  340
##
##  ( 173 167 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
train_pred <- predict(bestmod, train)
test_pred <- predict(bestmod, test)

table(train_pred, train$Purchase)
```

```
##
## train_pred  CH  MM
##         CH 451  79
##         MM  39 231
```

```r
table(test_pred, test$Purchase)
```

```
##
## test_pred  CH  MM
##        CH 150  47
##        MM  13  60
```

```r
# print training accuracy
train_accuracy <- mean(train_pred == train$Purchase); train_accuracy
```

```
## [1] 0.8525
```

```r
# print training error
1 - train_accuracy
```

```
## [1] 0.1475
```

```r
# print test accuracy
test_accuracy <- mean(test_pred == test$Purchase); test_accuracy
```

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

```r
# print test error
1 - test_accuracy
```

```
## [1] 0.2222222
```

- **(h)**
  - Linear
    - training error = 15.875%
    - testing error = 17.40741%
  - Radial
    - training error = 13.875%
    - testing error = 22.59259%
  - Polynomial
    - training error = 14.75%
    - testing error = 22.22222%
  - The models all performed similarly on training accuracy, ~14-16% error, but the linear kernel had ~5% less error than the polynomial or radial SVMs. Therefore the linear model had the best results on this data for generalizability.
