Stat 432 Homework 8

Assigned: Oct 14, 2024; Due: 11:59 PM CT, Oct 24, 2024

Contents

Question 1: Discriminant Analysis (60 points)

We will be using the first 2500 observations of the MNIST dataset. You can use the following code, or the saved data from our previous homework.

```
# inputs to download file
fileLocation <- "https://pjreddie.com/media/files/mnist_train.csv"
numRowsToDownload <- 2500
localFileName <- paste0("mnist_first", numRowsToDownload, ".RData")

# download the data and add column names
mnist <- read.csv(fileLocation, nrows = numRowsToDownload)
numColsMnist <- dim(mnist)[2]
colnames(mnist) <- c("Digit", paste("Pixel", seq(1:(numColsMnist - 1)), sep = ""))

# save file
# in the future we can read in from the local copy instead of having to redownload
save(mnist, file = localFileName)

# you can load the data with the following code
load(file = localFileName)
```

a. [10 pts] Write you own code to fit a Linear Discriminant Analysis (LDA) model to the MNIST dataset. Use the first 1250 observations as the training set and the remaining observations as the test set. An issue with this dataset is that some pixels display little or no variation across all observations. This zero variance issue poses a problem when inverting the estimated covariance matrix. To address this issue, take digits 1, 7, and 9 from the training data, and perform a screening on the marginal variance of all 784 pixels. Take the top 300 pixels with the largest variance and use them to fit the LDA model. Remove the remaining ones from the training and test data.

```
library(MASS)  # For LDA
library(dplyr) # For data manipulation
```

```
##
## Attaching package: 'dplyr'
```

```
## The following object is masked from 'package:MASS':
##
##     select

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
load(file = "mnist_first2500.RData")

train_data <- mnist[1:1250, ]
test_data <- mnist[1251:2500, ]

train_filtered <- train_data %>% filter(Digit %in% c(1, 7, 9))

pixel_variances <- apply(train_filtered[, -1], 2, var)

top_pixels <- order(pixel_variances, decreasing = TRUE)[1:300]

train_selected <- train_filtered[, c(1, top_pixels + 1)]
test_selected <- test_data[, c(1, top_pixels + 1)]

lda_model <- lda(Digit ~ ., data = train_selected)

predictions <- predict(lda_model, newdata = test_selected)

accuracy <- mean(predictions$class == test_selected$Digit)
cat("LDA Model Accuracy:", accuracy * 100, "%\n")
```

```
## LDA Model Accuracy: 26.64 %
```

b. [30 pts] Write your own code to implement the LDA model. Remember that LDA requires the estimation of several parameters: Σ, μ_k, and π_k. Estimate these parameters and calculate the decision scores δ_k on the testing data to predict the class label. Report the accuracy and the confusion matrix based on the testing data.

```
library(caret)
```

```
## Loading required package: ggplot2

## Loading required package: lattice
```

```
LDA_custom <- function(train_data, test_data) {
  classes <- unique(train_data$Digit)
  num_classes <- length(classes)
  mu_k <- train_data %>%
    group_by(Digit) %>%
```

```r
    summarise_all(mean) %>%
    select(-Digit) %>%
    as.data.frame()
  cov_matrix <- cov(train_data[,-1])
  pi_k <- table(train_data$Digit) / nrow(train_data)
  delta_k <- function(x, k) {
    mu <- as.numeric(mu_k[k, ])
    pi <- pi_k[k]
    delta <- t(x) %*% solve(cov_matrix) %*% mu - 0.5 * t(mu) %*% solve(cov_matrix) %*% mu + log(pi)
    return(delta)
  }
  predict_LDA <- function(test_data) {
    predictions <- c()

    for (i in 1:nrow(test_data)) {
      x <- as.numeric(test_data[i, -1])
      delta_values <- sapply(1:num_classes, function(k) delta_k(x, k))
      predicted_class <- classes[which.max(delta_values)]
      predictions <- c(predictions, predicted_class)
    }

    return(predictions)
  }
  predictions <- predict_LDA(test_data)
  actual <- test_data$Digit
  accuracy <- mean(predictions == actual)
  conf_matrix <- confusionMatrix(factor(predictions), factor(actual))

  return(list(accuracy = accuracy, confusion_matrix = conf_matrix))
}

load(file = "mnist_first2500.RData")
train_data <- mnist[1:1250, ]
test_data <- mnist[1251:2500, ]

train_filtered <- train_data %>% filter(Digit %in% c(1, 7, 9))
test_filtered <- test_data %>% filter(Digit %in% c(1, 7, 9))

pixel_variances <- apply(train_filtered[, -1], 2, var)
top_pixels <- order(pixel_variances, decreasing = TRUE)[1:300]

train_selected <- train_filtered[, c(1, top_pixels + 1)]
test_selected <- test_filtered[, c(1, top_pixels + 1)]

results <- LDA_custom(train_selected, test_selected)

cat("Accuracy:", results$accuracy * 100, "%\n")
```

```
## Accuracy: 43.04813 %
```

```r
print(results$confusion_matrix)
```

```
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction   1   7   9
##          1 124   5   3
##          7   1  11  91
##          9   3 110  26
##
## Overall Statistics
##
##                Accuracy : 0.4305
##                  95% CI : (0.3797, 0.4824)
##     No Information Rate : 0.3422
##     P-Value [Acc > NIR] : 0.000244
##
##                   Kappa : 0.1464
##
##  Mcnemar's Test P-Value : 0.215643
##
## Statistics by Class:
##
##                      Class: 1 Class: 7 Class: 9
## Sensitivity            0.9688  0.08730  0.21667
## Specificity            0.9675  0.62903  0.55512
## Pos Pred Value         0.9394  0.10680  0.18705
## Neg Pred Value         0.9835  0.57565  0.60000
## Prevalence             0.3422  0.33690  0.32086
## Detection Rate         0.3316  0.02941  0.06952
## Detection Prevalence   0.3529  0.27540  0.37166
## Balanced Accuracy      0.9681  0.35817  0.38589
```

c. [10 pts] Use the `lda()` function from MASS package to fit LDA. Report the accuracy and the confusion matrix based on the testing data. Compare your results with part b.

```r
library(MASS)

# Load the MNIST dataset and split into training and test sets
load(file = "mnist_first2500.RData")
train_data <- mnist[1:1250, ]
test_data <- mnist[1251:2500, ]

# Filter for digits 1, 7, and 9 from the training data
train_filtered <- train_data %>% filter(Digit %in% c(1, 7, 9))
test_filtered <- test_data %>% filter(Digit %in% c(1, 7, 9))

# Select the top 300 pixels with the largest variance
pixel_variances <- apply(train_filtered[, -1], 2, var)
top_pixels <- order(pixel_variances, decreasing = TRUE)[1:300]

# Subset the training and test data to retain only the selected pixels
train_selected <- train_filtered[, c(1, top_pixels + 1)]  # +1 to account for 'Digit' column
test_selected <- test_filtered[, c(1, top_pixels + 1)]

# Step 1: Fit the LDA model using the `lda()` function from MASS
lda_model <- lda(Digit ~ ., data = train_selected)
```

```r
# Step 2: Make predictions on the test data
lda_predictions <- predict(lda_model, newdata = test_selected)

# Step 3: Calculate the accuracy
lda_accuracy <- mean(lda_predictions$class == test_selected$Digit)

# Step 4: Generate the confusion matrix
lda_conf_matrix <- confusionMatrix(factor(lda_predictions$class), factor(test_selected$Digit))

# Report the accuracy and confusion matrix
cat("LDA Model Accuracy (using MASS lda()):", lda_accuracy * 100, "%\n")
```

```
## LDA Model Accuracy (using MASS lda()): 89.03743 %
```

```r
print(lda_conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction   1   7   9
##          1 125   3   2
##          7   2 111  21
##          9   1  12  97
##
## Overall Statistics
##
##                Accuracy : 0.8904
##                  95% CI : (0.8542, 0.9202)
##     No Information Rate : 0.3422
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8354
##
##  Mcnemar's Test P-Value : 0.3935
##
## Statistics by Class:
##
##                      Class: 1 Class: 7 Class: 9
## Sensitivity            0.9766   0.8810   0.8083
## Specificity            0.9797   0.9073   0.9488
## Pos Pred Value         0.9615   0.8284   0.8818
## Neg Pred Value         0.9877   0.9375   0.9129
## Prevalence             0.3422   0.3369   0.3209
## Detection Rate         0.3342   0.2968   0.2594
## Detection Prevalence   0.3476   0.3583   0.2941
## Balanced Accuracy      0.9781   0.8941   0.8786
```

d. [10 pts] Use the `qda()` function from MASS package to fit QDA. Does the code work directly? Why? If you are asked to modify your own code to perform QDA, what would you do? Discuss this issue and propose at least two solutions to address it. If relavent, provide mathematical reasoning (in latex) of your solution. You **do not** need to implement that with code. No, the code not work directly with the qda() function. One of the key reasons is that Quadratic Discriminant Analysis requires estimating a separate covariance matrix for each class. If there are issues such as zero or near-zero variance in the data, or if some features have very low variance, the QDA model might run into issues when attempting to invert the covariance matrix for each class. If I were asked to modify my own code, I would Introduce regularization by adding ridge parameter lambda.This helps avoid the singularity problem by making the covariance matrix invertible.

$$\Sigma'_k = \Sigma_k + \lambda I$$

Another way to address the singular covariance issue is to perform feature selection or dimensionality reduction by removing features (pixels) that have near-zero variance. Mathematically, this involves checking the determinant of each class-specific covariance matrix:

$$|\Sigma_k| > 0$$

In QDA, the class-specific covariance matrix Sigma_k is used to model the spread of the data for class k. The discriminant function for class k involves both the determinant Sigma_k and the inverse Sigma_k^{-1}:

$$\delta_k(x) = -\frac{1}{2}\log|\Sigma_k| - \frac{1}{2}(x-\mu_k)^\top \Sigma_k^{-1}(x-\mu_k) + \log(\pi_k)$$

If sigma_k = 0, the inverse Sigma_k^{-1} does not exist, and the discriminant function cannot be computed.

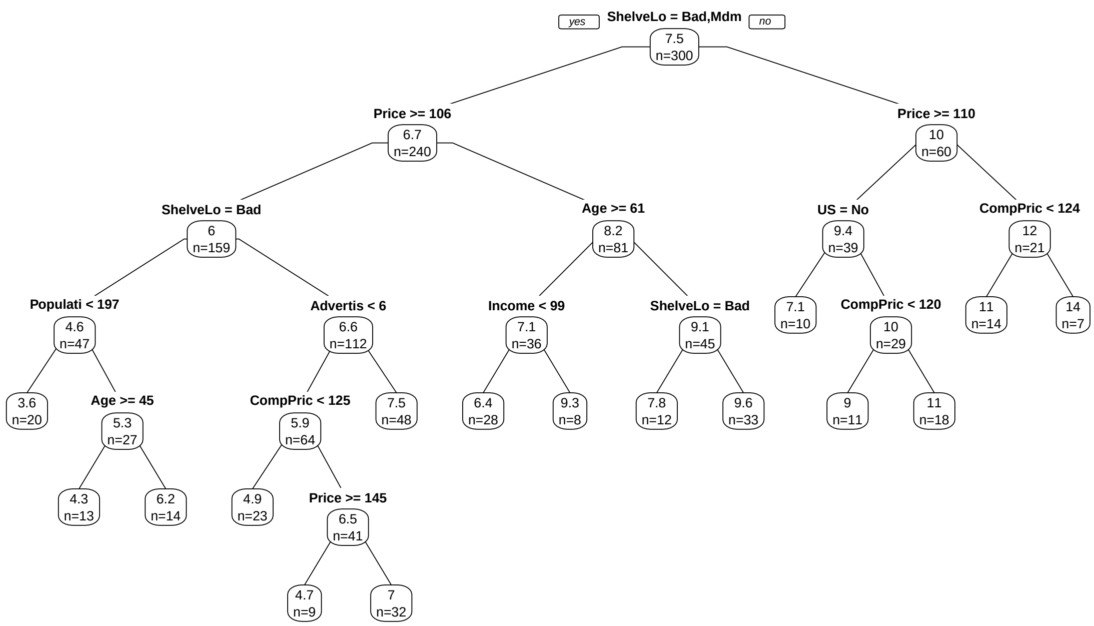
Question 2: Regression Trees (40 points)

Load data `Carseats` from the `ISLR` package. Use the following code to define the training and test sets.

```r
# load library
library(ISLR)

# load data
data(Carseats)

# set seed
set.seed(7)

# number of rows in entire dataset
n_Carseats <- dim(Carseats)[1]

# training set parameters
train_percentage <- 0.75
train_size <- floor(train_percentage*n_Carseats)
train_indices <- sample(x = 1:n_Carseats, size = train_size)

# separate dataset into train and test
train_Carseats <- Carseats[train_indices,]
test_Carseats <- Carseats[-train_indices,]
```

a. [20 pts] We seek to predict the variable `Sales` using a regression tree. Load the library `rpart`. Fit a regression tree to the training set using the `rpart()` function, all hyperparameter arguments should be left as default. Load the library `rpart.plot()`. Plot the tree using the `prp()` function. Based on this model, what type of observations has the highest or lowest sales? Predict using the tree onto the test set, calculate and report the MSE on the testing data.

```r
library(rpart)
library(rpart.plot)
tree_model <- rpart(Sales ~ ., data = train_Carseats)
prp(tree_model, type = 1, extra = 1)
```




```r
predictions <- predict(tree_model, newdata = test_Carseats)
mse <- mean((predictions - test_Carseats$Sales)^2)
mse
```

```
## [1] 4.51347
```

```r
summary(tree_model)
```

```
## Call:
## rpart(formula = Sales ~ ., data = train_Carseats)
##   n= 300
##
##            CP nsplit rel error    xerror       xstd
## 1  0.26171490      0 1.0000000 1.0051876 0.08088926
## 2  0.10841398      1 0.7382851 0.7483700 0.06096581
## 3  0.05518323      2 0.6298711 0.6454034 0.05096855
## 4  0.04143279      3 0.5746879 0.6030383 0.05081197
## 5  0.03531304      4 0.5332551 0.6020229 0.04912763
## 6  0.03028131      5 0.4979421 0.5922682 0.04772484
```

```
## 7  0.02958110      6 0.4676607 0.5692657 0.04720330
## 8  0.02173028      7 0.4380796 0.5286490 0.04343673
## 9  0.01529893      8 0.4163494 0.5378804 0.04271219
## 10 0.01479043     10 0.3857515 0.5393631 0.04234195
## 11 0.01175257     11 0.3709611 0.5409597 0.04384293
## 12 0.01156122     12 0.3592085 0.5357537 0.04283274
## 13 0.01016828     13 0.3476473 0.5220956 0.04102937
## 14 0.01010924     14 0.3374790 0.5225807 0.04123589
## 15 0.01000000     15 0.3273698 0.5220202 0.04121369
##
## Variable importance
##   ShelveLoc       Price   CompPrice Advertising         Age          US
##          38          22           9           8           6           6
##  Population      Income   Education
##           5           5           1
##
## Node number 1: 300 observations,    complexity param=0.2617149
##   mean=7.474567, MSE=8.148088
##   left son=2 (240 obs) right son=3 (60 obs)
##   Primary splits:
##       ShelveLoc   splits as  LRL,       improve=0.26171490, (0 missing)
##       Price       < 106   to the right, improve=0.14025620, (0 missing)
##       Age         < 61.5  to the right, improve=0.09934454, (0 missing)
##       Advertising < 6.5   to the left,  improve=0.08477633, (0 missing)
##       US          splits as  LR,        improve=0.02956584, (0 missing)
##
## Node number 2: 240 observations,    complexity param=0.108414
##   mean=6.744417, MSE=6.036325
##   left son=4 (159 obs) right son=5 (81 obs)
##   Primary splits:
##       Price       < 106   to the right, improve=0.18292720, (0 missing)
##       ShelveLoc   splits as  L-R,       improve=0.11581490, (0 missing)
##       Advertising < 7.5   to the left,  improve=0.09288208, (0 missing)
##       Age         < 64.5  to the right, improve=0.08140715, (0 missing)
##       Income      < 98.5  to the left,  improve=0.04343076, (0 missing)
##   Surrogate splits:
##       CompPrice  < 113.5 to the right, agree=0.750, adj=0.259, (0 split)
##       Population < 487   to the left,  agree=0.675, adj=0.037, (0 split)
##
## Node number 3: 60 observations,    complexity param=0.04143279
##   mean=10.39517, MSE=5.932758
##   left son=6 (39 obs) right son=7 (21 obs)
##   Primary splits:
##       Price       < 109.5 to the right, improve=0.28452030, (0 missing)
##       Age         < 61.5  to the right, improve=0.12150940, (0 missing)
##       Advertising < 13.5  to the left,  improve=0.11127150, (0 missing)
##       US          splits as  LR,        improve=0.06955854, (0 missing)
##       Population  < 77    to the right, improve=0.05692815, (0 missing)
##   Surrogate splits:
##       Population < 77    to the right, agree=0.733, adj=0.238, (0 split)
##       Income     < 29.5  to the right, agree=0.700, adj=0.143, (0 split)
##       Age        < 26.5  to the right, agree=0.683, adj=0.095, (0 split)
##       Education  < 17.5  to the left,  agree=0.683, adj=0.095, (0 split)
##       CompPrice  < 102   to the right, agree=0.667, adj=0.048, (0 split)
```

```
##
## Node number 4: 159 observations,    complexity param=0.05518323
##   mean=5.994403, MSE=4.930182
##   left son=8 (47 obs) right son=9 (112 obs)
##   Primary splits:
##       ShelveLoc   splits as  L-R,       improve=0.17207750, (0 missing)
##       CompPrice   < 124.5 to the left,  improve=0.12690100, (0 missing)
##       Advertising < 7.5   to the left,  improve=0.11223830, (0 missing)
##       Age         < 64.5  to the right, improve=0.09310875, (0 missing)
##       Price       < 136.5 to the right, improve=0.07773729, (0 missing)
##   Surrogate splits:
##       Population < 18.5  to the left,  agree=0.717, adj=0.043, (0 split)
##
## Node number 5: 81 observations,    complexity param=0.03531304
##   mean=8.216667, MSE=4.935916
##   left son=10 (36 obs) right son=11 (45 obs)
##   Primary splits:
##       Age       < 60.5  to the right, improve=0.2159033, (0 missing)
##       CompPrice < 123.5 to the left,  improve=0.1891027, (0 missing)
##       Income    < 102.5 to the left,  improve=0.1505879, (0 missing)
##       Price     < 88    to the right, improve=0.1089945, (0 missing)
##       ShelveLoc splits as  L-R,       improve=0.1030606, (0 missing)
##   Surrogate splits:
##       CompPrice  < 124.5 to the left,  agree=0.605, adj=0.111, (0 split)
##       Income     < 73.5  to the right, agree=0.593, adj=0.083, (0 split)
##       Price      < 94.5  to the right, agree=0.593, adj=0.083, (0 split)
##       Population < 234.5 to the right, agree=0.580, adj=0.056, (0 split)
##       ShelveLoc  splits as  L-R,       agree=0.568, adj=0.028, (0 split)
##
## Node number 6: 39 observations,    complexity param=0.0295811
##   mean=9.441795, MSE=4.976507
##   left son=12 (10 obs) right son=13 (29 obs)
##   Primary splits:
##       US          splits as  LR,        improve=0.3725650, (0 missing)
##       Advertising < 0.5   to the left,  improve=0.3623589, (0 missing)
##       Price       < 135   to the right, improve=0.2207263, (0 missing)
##       Age         < 61.5  to the right, improve=0.2077134, (0 missing)
##       Income      < 35.5  to the left,  improve=0.1377684, (0 missing)
##   Surrogate splits:
##       Advertising < 0.5   to the left,  agree=0.949, adj=0.8, (0 split)
##       Price       < 142.5 to the right, agree=0.795, adj=0.2, (0 split)
##       CompPrice   < 141   to the right, agree=0.769, adj=0.1, (0 split)
##       Income      < 35.5  to the left,  agree=0.769, adj=0.1, (0 split)
##       Population  < 117   to the left,  agree=0.769, adj=0.1, (0 split)
##
## Node number 7: 21 observations,    complexity param=0.01016828
##   mean=12.16571, MSE=2.885824
##   left son=14 (14 obs) right son=15 (7 obs)
##   Primary splits:
##       CompPrice  < 124   to the left,  improve=0.41014310, (0 missing)
##       Age        < 55.5  to the right, improve=0.13420960, (0 missing)
##       Urban      splits as  LR,        improve=0.07910907, (0 missing)
##       Education  < 11.5  to the right, improve=0.07143769, (0 missing)
##       Population < 187.5 to the right, improve=0.06700131, (0 missing)
```

```
##   Surrogate splits:
##       Price < 90.5  to the left,  agree=0.714, adj=0.143, (0 split)
##
## Node number 8: 47 observations,    complexity param=0.01479043
##   mean=4.572553, MSE=4.407142
##   left son=16 (20 obs) right son=17 (27 obs)
##   Primary splits:
##       Population  < 196.5 to the left,  improve=0.1745432, (0 missing)
##       Age         < 61.5  to the right, improve=0.1529259, (0 missing)
##       CompPrice   < 120   to the left,  improve=0.1483448, (0 missing)
##       Price       < 132.5 to the right, improve=0.1382951, (0 missing)
##       Advertising < 10.5  to the left,  improve=0.1165734, (0 missing)
##   Surrogate splits:
##       Advertising < 1.5   to the left,  agree=0.745, adj=0.40, (0 split)
##       Age         < 67.5  to the right, agree=0.681, adj=0.25, (0 split)
##       Education   < 15.5  to the right, agree=0.660, adj=0.20, (0 split)
##       Price       < 132.5 to the right, agree=0.638, adj=0.15, (0 split)
##       US          splits as  LR,        agree=0.638, adj=0.15, (0 split)
##
## Node number 9: 112 observations,    complexity param=0.03028131
##   mean=6.591071, MSE=3.945285
##   left son=18 (64 obs) right son=19 (48 obs)
##   Primary splits:
##       Advertising < 5.5   to the left,  improve=0.16751560, (0 missing)
##       CompPrice   < 124.5 to the left,  improve=0.14107040, (0 missing)
##       Age         < 64.5  to the right, improve=0.12783510, (0 missing)
##       Price       < 135.5 to the right, improve=0.11821850, (0 missing)
##       Income      < 61.5  to the left,  improve=0.08229926, (0 missing)
##   Surrogate splits:
##       US         splits as  LR,        agree=0.804, adj=0.542, (0 split)
##       Population < 208.5 to the left,  agree=0.643, adj=0.167, (0 split)
##       Income     < 117   to the left,  agree=0.607, adj=0.083, (0 split)
##       CompPrice  < 97.5  to the right, agree=0.589, adj=0.042, (0 split)
##       Age        < 47.5  to the right, agree=0.589, adj=0.042, (0 split)
##
## Node number 10: 36 observations,    complexity param=0.02173028
##   mean=7.0625, MSE=4.58883
##   left son=20 (28 obs) right son=21 (8 obs)
##   Primary splits:
##       Income      < 99    to the left,  improve=0.32154210, (0 missing)
##       Price       < 92.5  to the right, improve=0.21867710, (0 missing)
##       Advertising < 11.5  to the left,  improve=0.13232740, (0 missing)
##       CompPrice   < 118.5 to the left,  improve=0.12965100, (0 missing)
##       Education   < 11.5  to the left,  improve=0.06211541, (0 missing)
##   Surrogate splits:
##       Advertising < 11.5  to the left,  agree=0.833, adj=0.25, (0 split)
##       Price       < 80.5  to the right, agree=0.833, adj=0.25, (0 split)
##
## Node number 11: 45 observations,    complexity param=0.01156122
##   mean=9.14, MSE=3.29536
##   left son=22 (12 obs) right son=23 (33 obs)
##   Primary splits:
##       ShelveLoc   splits as  L-R,       improve=0.19057470, (0 missing)
##       CompPrice   < 131.5 to the left,  improve=0.12963420, (0 missing)
```

```
##        Income      < 57    to the left,  improve=0.10707800, (0 missing)
##        Advertising < 9.5   to the left,  improve=0.09865769, (0 missing)
##        Age         < 35    to the right, improve=0.07840298, (0 missing)
##    Surrogate splits:
##        Education < 10.5  to the left,  agree=0.778, adj=0.167, (0 split)
##
## Node number 12: 10 observations
##    mean=7.123, MSE=1.844981
##
## Node number 13: 29 observations,    complexity param=0.01175257
##    mean=10.24138, MSE=3.562936
##    left son=26 (11 obs) right son=27 (18 obs)
##    Primary splits:
##        CompPrice   < 120   to the left,  improve=0.27803770, (0 missing)
##        Advertising < 13.5  to the left,  improve=0.23932930, (0 missing)
##        Age         < 46.5  to the right, improve=0.13795240, (0 missing)
##        Price       < 128.5 to the right, improve=0.10803730, (0 missing)
##        Income      < 64.5  to the left,  improve=0.06173821, (0 missing)
##    Surrogate splits:
##        Income      < 111.5 to the right, agree=0.690, adj=0.182, (0 split)
##        Advertising < 18    to the right, agree=0.655, adj=0.091, (0 split)
##        Population  < 445.5 to the right, agree=0.655, adj=0.091, (0 split)
##        Price       < 119   to the left,  agree=0.655, adj=0.091, (0 split)
##
## Node number 14: 14 observations
##    mean=11.39643, MSE=1.68878
##
## Node number 15: 7 observations
##    mean=13.70429, MSE=1.72911
##
## Node number 16: 20 observations
##    mean=3.5535, MSE=3.757753
##
## Node number 17: 27 observations,    complexity param=0.01010924
##    mean=5.327407, MSE=3.54913
##    left son=34 (13 obs) right son=35 (14 obs)
##    Primary splits:
##        Age         < 44.5  to the right, improve=0.2578754, (0 missing)
##        Price       < 127.5 to the right, improve=0.1396096, (0 missing)
##        CompPrice   < 137.5 to the left,  improve=0.1385838, (0 missing)
##        Population  < 337.5 to the right, improve=0.1272430, (0 missing)
##        Advertising < 10.5  to the left,  improve=0.1189134, (0 missing)
##    Surrogate splits:
##        Price       < 112.5 to the left,  agree=0.667, adj=0.308, (0 split)
##        Income      < 77.5  to the right, agree=0.630, adj=0.231, (0 split)
##        Advertising < 8.5   to the left,  agree=0.630, adj=0.231, (0 split)
##        Population  < 340.5 to the right, agree=0.630, adj=0.231, (0 split)
##        US          splits as  RL,        agree=0.630, adj=0.231, (0 split)
##
## Node number 18: 64 observations,    complexity param=0.01529893
##    mean=5.887031, MSE=3.733699
##    left son=36 (23 obs) right son=37 (41 obs)
##    Primary splits:
##        CompPrice  < 124.5 to the left,  improve=0.15593990, (0 missing)
```

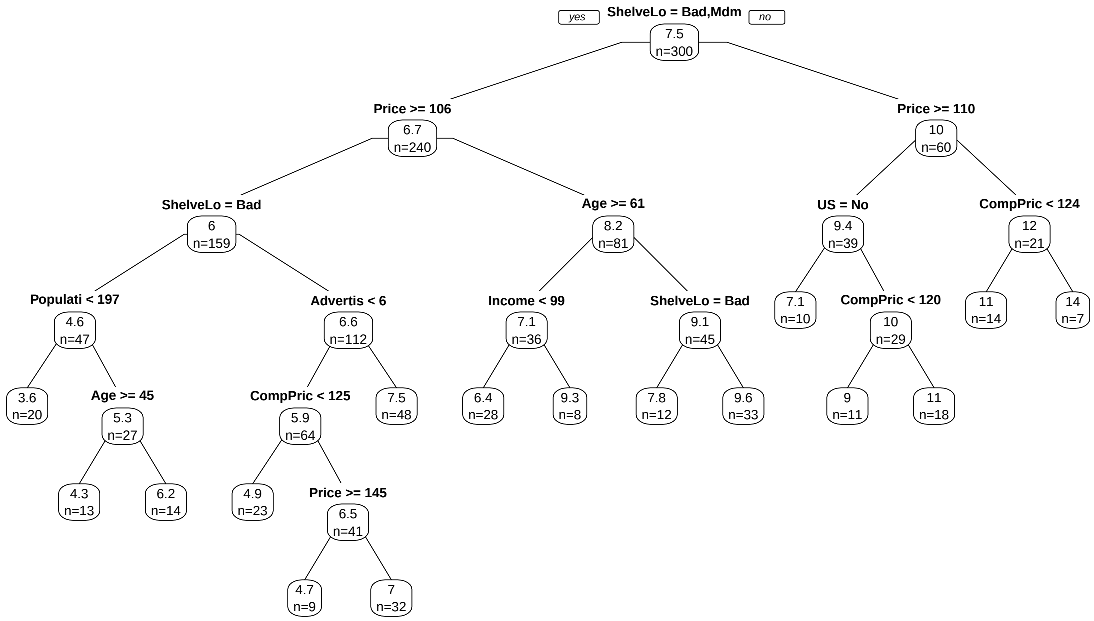
```
##       Price      < 127   to the right, improve=0.14244230, (0 missing)
##       Age        < 61.5  to the right, improve=0.11635990, (0 missing)
##       Income     < 77.5  to the left,  improve=0.10920480, (0 missing)
##       Population < 310.5 to the right, improve=0.04100289, (0 missing)
##   Surrogate splits:
##       Price      < 111.5 to the left,  agree=0.781, adj=0.391, (0 split)
##       Age        < 30.5  to the left,  agree=0.703, adj=0.174, (0 split)
##       Population < 53    to the left,  agree=0.688, adj=0.130, (0 split)
##       Income     < 29    to the left,  agree=0.656, adj=0.043, (0 split)
##
## Node number 19: 48 observations
##   mean=7.529792, MSE=2.685306
##
## Node number 20: 28 observations
##   mean=6.413214, MSE=2.700343
##
## Node number 21: 8 observations
##   mean=9.335, MSE=4.558775
##
## Node number 22: 12 observations
##   mean=7.825833, MSE=2.055808
##
## Node number 23: 33 observations
##   mean=9.617879, MSE=2.889726
##
## Node number 26: 11 observations
##   mean=8.968182, MSE=2.075851
##
## Node number 27: 18 observations
##   mean=11.01944, MSE=2.875694
##
## Node number 34: 13 observations
##   mean=4.334615, MSE=1.641425
##
## Node number 35: 14 observations
##   mean=6.249286, MSE=3.555478
##
## Node number 36: 23 observations
##   mean=4.868261, MSE=2.484675
##
## Node number 37: 41 observations,    complexity param=0.01529893
##   mean=6.458537, MSE=3.52552
##   left son=74 (9 obs) right son=75 (32 obs)
##   Primary splits:
##       Price     < 144.5 to the right, improve=0.25964910, (0 missing)
##       Age       < 61.5  to the right, improve=0.20009240, (0 missing)
##       Income    < 34    to the left,  improve=0.17883590, (0 missing)
##       CompPrice < 147.5 to the left,  improve=0.06104972, (0 missing)
##       Urban     splits as  LR,        improve=0.04165624, (0 missing)
##   Surrogate splits:
##       Income     < 24.5  to the left,  agree=0.829, adj=0.222, (0 split)
##       CompPrice  < 151   to the right, agree=0.805, adj=0.111, (0 split)
##       Population < 63.5  to the left,  agree=0.805, adj=0.111, (0 split)
##
```

```
## Node number 74: 9 observations
##   mean=4.654444, MSE=3.867269
##
## Node number 75: 32 observations
##   mean=6.965937, MSE=2.256549
```

Price >= 110 and Compric<124 has the highest sales. Price>=106, SheiveLo=Bad,Populati<197,age<45 has the lowest sale.

b. [20 pts] Set the seed to 7 at the beginning of the chunk and do this question in a single chunk so the seed doesn't get switched. Find the largest complexity parameter value of the tree you grew in part a) that will ensure that the cross-validation error < min(cross-validation error) + cross-validation standard deviation. Print that complexity parameter value. Prune the tree using that value. Predict using the pruned tree onto the test set, calculate the test Mean-Squared Error, and print it.

```r
set.seed(7)

library(rpart)
library(rpart.plot)

data(Carseats)
n_Carseats <- dim(Carseats)[1]
train_percentage <- 0.75
train_size <- floor(train_percentage * n_Carseats)
train_indices <- sample(1:n_Carseats, size = train_size)
train_Carseats <- Carseats[train_indices,]
test_Carseats <- Carseats[-train_indices,]

tree_model <- rpart(Sales ~ ., data = train_Carseats)

# Plot the tree
prp(tree_model, type = 1, extra = 1)
```




```r
predictions <- predict(tree_model, newdata = test_Carseats)
mse_original <- mean((predictions - test_Carseats$Sales)^2)
mse_original
```

```
## [1] 4.51347
```

```r
printcp(tree_model)
```

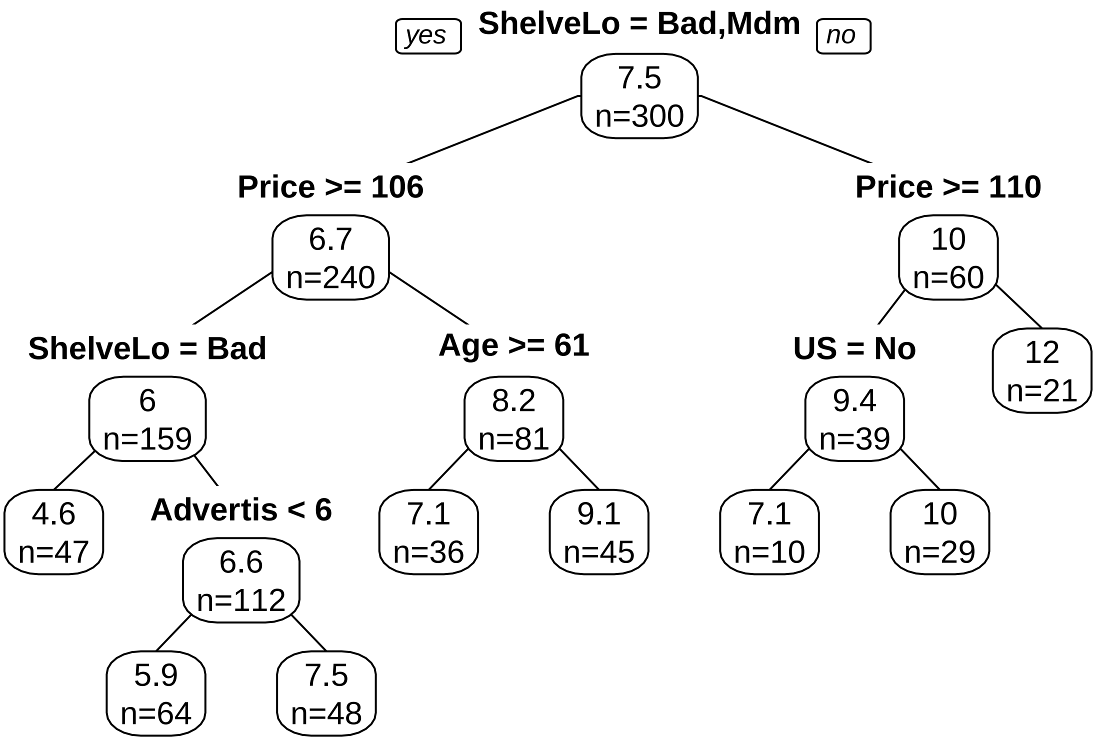
```
##
## Regression tree:
## rpart(formula = Sales ~ ., data = train_Carseats)
##
## Variables actually used in tree construction:
## [1] Advertising Age         CompPrice   Income      Population  Price
## [7] ShelveLoc   US
##
## Root node error: 2444.4/300 = 8.1481
##
## n= 300
##
##          CP nsplit rel error  xerror     xstd
## 1  0.261715      0   1.00000 1.00519 0.080889
## 2  0.108414      1   0.73829 0.74837 0.060966
## 3  0.055183      2   0.62987 0.64540 0.050969
```

```
## 4  0.041433      3   0.57469 0.60304 0.050812
## 5  0.035313      4   0.53326 0.60202 0.049128
## 6  0.030281      5   0.49794 0.59227 0.047725
## 7  0.029581      6   0.46766 0.56927 0.047203
## 8  0.021730      7   0.43808 0.52865 0.043437
## 9  0.015299      8   0.41635 0.53788 0.042712
## 10 0.014790     10   0.38575 0.53936 0.042342
## 11 0.011753     11   0.37096 0.54096 0.043843
## 12 0.011561     12   0.35921 0.53575 0.042833
## 13 0.010168     13   0.34765 0.52210 0.041029
## 14 0.010109     14   0.33748 0.52258 0.041236
## 15 0.010000     15   0.32737 0.52202 0.041214
```

```r
cp_table <- tree_model$cptable
min_xerror <- min(cp_table[,"xerror"])
xerror_threshold <- min_xerror + cp_table[which.min(cp_table[,"xerror"]),"xstd"]
optimal_cp <- max(cp_table[cp_table[,"xerror"] <= xerror_threshold,"CP"])
optimal_cp
```

```
## [1] 0.02173028
```

```r
pruned_tree <- prune(tree_model, cp = optimal_cp)

prp(pruned_tree, type = 1, extra = 1)
```



```r
pruned_predictions <- predict(pruned_tree, newdata = test_Carseats)
mse_pruned <- mean((pruned_predictions - test_Carseats$Sales)^2)
mse_pruned
```

```
## [1] 4.543565
```