Stat 432 Homework 8

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

Instruction

Please remove this section when submitting your homework.

Students are encouraged to work together on homework and/or utilize advanced AI tools. However, **sharing, copying, or providing any part of a homework solution or code to others** is an infraction of the University's rules on Academic Integrity (https://studentcode.illinois.edu/article1/part4/1-401/). Any violation will be punished as severely as possible. Final submissions must be uploaded to Gradescope (https://www.gradescope.com/courses/570816). No email or hard copy will be accepted. For **late submission policy and grading rubrics** (https://teazrq.github.io/stat432/syllabus.html), please refer to the course website.

- You are required to submit the rendered file `HWx_yourNetID.pdf`. For example, `HW01_rqzhu.pdf`. Please note that this must be a `.pdf` file. `.html` format **cannot** be accepted. Make all of your `R` code chunks visible for grading.
- Include your Name and NetID in the report.
- If you use this file or the example homework `.Rmd` file as a template, be sure to **remove this instruction** section.
- Make sure that you **set seed** properly so that the results can be replicated if needed.
- For some questions, there will be restrictions on what packages/functions you can use. Please read the requirements carefully. As long as the question does not specify s uch restrictions, you can use anything.
- **When using AI tools**, you are encouraged to document your comment on your experience with AI tools especially when it's difficult for them to grasp the idea of the question.
- **On random seed and reproducibility**: Make sure the version of your `R` is ≥ $4.0.0$. This will ensure your random seed generation is the same as everyone else. Please note that updating the `R` version may require you to reinstall all of your packages.

Question 1: Discriminant Analysis

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.

Solution:

```
# construct the training and test data
train <- mnist[1:1250,]
train <- train[train$Digit == 1 | train$Digit == 7 | train$Digit == 9,]
test <- mnist[1251:2500,]
test <- test[test$Digit == 1 | test$Digit == 7| test$Digit == 9,]

# variance screening
var = apply(train[, -1], 2, var)
varuse = order(var, decreasing = TRUE)[1:300]
train = train[, c(1, varuse + 1)]
test = test[, c(1, varuse + 1)]
```

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.

Solution:

```
# get digit 1, 7, 9
digit1 = train[train$Digit == 1, -1]
digit7 = train[train$Digit == 7, -1]
digit9 = train[train$Digit == 9, -1]

# calculate the mean and covariance matrix
mu1 = colMeans(digit1)
mu7 = colMeans(digit7)
mu9 = colMeans(digit9)

digit1_centered = scale(digit1, center = TRUE, scale = FALSE)
digit7_centered = scale(digit7, center = TRUE, scale = FALSE)
digit9_centered = scale(digit9, center = TRUE, scale = FALSE)

# pooled covariance matrix
S = (t(digit1_centered) %*% digit1_centered +
     t(digit7_centered) %*% digit7_centered +
     t(digit9_centered) %*% digit9_centered) / (nrow(train) - 3)

# the prior
pi1 = nrow(digit1) / nrow(train)
pi7 = nrow(digit7) / nrow(train)
pi9 = nrow(digit9) / nrow(train)

# calculate the decision scores
test_pixel = data.matrix(test[, -1])
delta1 = rowSums(test_pixel %*% solve(S) %*% as.matrix(mu1)) +
         as.numeric(- 0.5 * t(mu1) %*% solve(S) %*% mu1 + log(pi1))
delta7 = rowSums(test_pixel %*% solve(S) %*% as.matrix(mu7)) +
         as.numeric(- 0.5 * t(mu7) %*% solve(S) %*% mu7 + log(pi7))
delta9 = rowSums(test_pixel %*% solve(S) %*% as.matrix(mu9)) +
         as.numeric(- 0.5 * t(mu9) %*% solve(S) %*% mu9 + log(pi9))

# predict the class label
pred = c(1, 7, 9)[apply(cbind(delta1, delta7, delta9), 1, which.max)]
conf_tab = table(pred, test[, 1])
conf_tab
```

```
##
## pred   1   7   9
##    1 125   3   2
##    7   2 111  21
##    9   1  12  97
```

```
# prediction accuracy
sum(diag(conf_tab))/nrow(test)
```

```
## [1] 0.8903743
```

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.

Solution:

```
library(MASS)
Digit.lda<-lda(Digit~., data=train)
pred.lda <- predict(Digit.lda, newdata=test)

# confusion matrix
conf_tab <- table(pred.lda$class, test$Digit)
conf_tab
```

```
##
##       1   7   9
##   1 125   3   2
##   7   2 111  21
##   9   1  12  97
```

```
# testing accuracy
sum(diag(conf_tab))/nrow(test)
```

```
## [1] 0.8903743
```

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 provide mathematical reasoning (in latex if needed) of your solution. You do not need to implement that with code.

Solution:

```
Digit.qda<-qda(Digit~., data=train)
pred.qda <- predict(Digit.qda, newdata=test)
table(pred.qda$class, test$Digit)
```

So the code does not work because there are not enough observations to support the inverse of the covariance matrix with 300 variables. There are several possible ways to address this:

1. Reduce the number of variables used in the model. For example, we can take just the top 100 instead of top 300.
2. Regularize the covariance matrix. For example, we can add a small diagonal matrix, just like the ridge regression penalty on the covariance matrix. $\widehat{\Sigma}_k(\delta) = \widehat{\Sigma}_k + \delta I$ for some small δ.

Question 2: Regression Trees (35 points)

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

```
# 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.

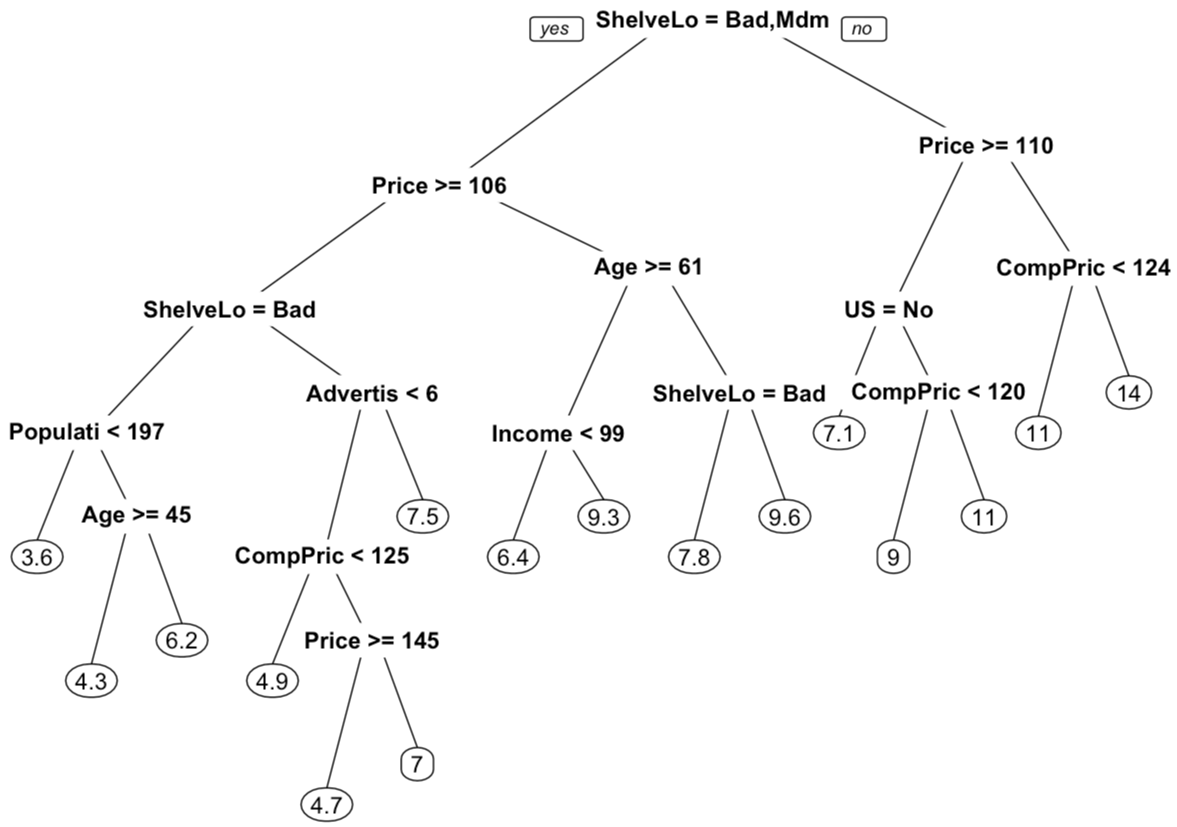
```
# load library
library(rpart)

# set seed
# set.seed(432)

# fit regression tree to training set
train_Carseats_reg_tree <- rpart(formula = Sales ~ ., data = train_Carseats)

# load library
library(rpart.plot)

# plot the tree
prp(train_Carseats_reg_tree)
```



It seems like high quality of the shelving location for the car seats, a price below $110, and a competitor price above $124 gets you the highest sales.

```
# predict on test set and calculate mse
test_Carseats_predicted <- predict(object = train_Carseats_reg_tree,
                                   newdata = test_Carseats)
test_Carseats_predicted_mse <- mean((test_Carseats$Sales - test_Carseats_predicted)^2)
print(paste0("The test MSE is ", test_Carseats_predicted_mse, "."))
```

```
## [1] "The test MSE is 4.5134702708434."
```

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
  set.seed(7)

  # select best Cp
  train_Carseats_reg_tree_Cp_table <- as.data.frame(train_Carseats_reg_tree$cptable)
   train_Carseats_reg_tree_Cp_table  <-  train_Carseats_reg_tree_Cp_table[,c("CP",  "xerror",  "xst
d")]
  min_xerror <- min(train_Carseats_reg_tree_Cp_table$xerror)
  train_Carseats_reg_tree_Cp_table$Cp_satisfies_cond <-
    ifelse(train_Carseats_reg_tree_Cp_table$xerror <
           min_xerror + train_Carseats_reg_tree_Cp_table$xstd,
           TRUE, FALSE)

  train_Carseats_reg_tree_Cp_table <- train_Carseats_reg_tree_Cp_table[
    which(train_Carseats_reg_tree_Cp_table$Cp_satisfies_cond),]
  train_Carseats_reg_tree_Cp_table <- train_Carseats_reg_tree_Cp_table[
    order(train_Carseats_reg_tree_Cp_table$CP, decreasing = TRUE),]
  optimal_complexity_parameter <- train_Carseats_reg_tree_Cp_table[1, "CP"]
  print(paste0("The optimal complexity parameter is ", optimal_complexity_parameter, "."))
```

```
## [1] "The optimal complexity parameter is 0.021730277228566."
```

```r
  # prune tree
  train_Carseats_reg_tree_pruned <- prune(tree = train_Carseats_reg_tree,
                                          cp = optimal_complexity_parameter)

  # predict on test set and calculate mse
  test_Carseats_predicted_pruned <- predict(object = train_Carseats_reg_tree_pruned,
                                            newdata = test_Carseats)
  test_Carseats_predicted_pruned_mse <-
    mean((test_Carseats$Sales - test_Carseats_predicted_pruned)^2)
  print(paste0("The test MSE is ", test_Carseats_predicted_pruned_mse, "."))
```

```
## [1] "The test MSE is 4.54356484357657."
```