STAT 432 Homework 8

Ram Goenka (rgoenka2)

2024-10-23

Question 1

(a) We begin by downloading and saving the data as mentioned in the problem:

```
#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)
```

I will now implement the LDA manually to the MNIST data. The first 1250 observations are used as training data and the rest of the data is used as the testing set. We take the digits 1, 7, 9 from the training data and perform a screen on the marginal variance of all 784 pixels, taking the top 300 pixels with the largest variance and use them to fit to the LDA model.

```
training_data <- mnist[1:1250, ]
test_data <- mnist[1251:2500, ]
digits_to_use <- c(1, 7, 9)
train_subset <- training_data[training_data$Digit %in% digits_to_use, ]
pixel_columns <- setdiff(names(train_subset), "Digit")
pixel_variances <- apply(train_subset[, pixel_columns], 2, var)
top_300_pixels <- names(sort(pixel_variances, decreasing = TRUE)[1:300])
train_reduced <- train_subset[, c("Digit", top_300_pixels)]
test_subset <- test_data[test_data$Digit %in% digits_to_use, ]
test_reduced <- test_subset[, c("Digit", top_300_pixels)]
```

(b) For this part I will be implementing the custom LDA model, the parameters mentioned are calculated as follows:

- π_k: Calculated by dividing the number of observations in each class by the total number of observations

- μ_k: Calculated by computing the mean of each pixel for each class
- Σ: Calculated by summing the within-class scatter matrices and dividing by $n - K$ where n is the total number of observations and K is the number of classes.

Now, we implement the custom LDA function:

```
classes <- sort(unique(train_reduced$Digit))
n_total <- nrow(train_reduced)
pi_k <- sapply(classes, function(k) sum(train_reduced$Digit == k) / n_total)
names(pi_k) <- classes
mu_k <- list()
for (k in classes) {
  mu_k[[as.character(k)]] <- colMeans(train_reduced[train_reduced$Digit == k, -1])
}

num_features <- length(top_300_pixels)
Sigma <- matrix(0, nrow = num_features, ncol = num_features)

for (k in classes) {
  X_k <- as.matrix(train_reduced[train_reduced$Digit == k, -1])
  X_k <- apply(X_k, 2, as.numeric)
  mu_k_vec <- mu_k[[as.character(k)]]
  X_centered <- sweep(X_k, 2, mu_k_vec, "-")
  Sigma <- Sigma + t(X_centered) %*% X_centered
}

n_minus_K <- n_total - length(classes)
Sigma <- Sigma / n_minus_K

Sigma_inv <- solve(Sigma)

X_test <- as.matrix(test_reduced[, -1])
X_test <- apply(X_test, 2, as.numeric)
n_test <- nrow(X_test)

delta <- matrix(0, nrow = n_test, ncol = length(classes))
colnames(delta) <- classes

for (i in 1:n_test) {
  x_i <- X_test[i, ]
  for (k in classes) {
    mu_k_vec <- mu_k[[as.character(k)]]
    delta[i, as.character(k)] <- t(x_i) %*% Sigma_inv %*%
      mu_k_vec - 0.5 * t(mu_k_vec) %*% Sigma_inv %*%
      mu_k_vec + log(pi_k[as.character(k)])
  }
}

predicted_classes <- as.numeric(apply(delta, 1, function(row) classes[which.max(row)]))
true_classes <- test_reduced$Digit
cm <- table(Predicted = predicted_classes, Actual = true_classes)
cm
```

```
##          Actual
```

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

```
acc <- mean(predicted_classes == true_classes)
acc
```

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

Based on the output of the custom LDA model, the accuracy on the testing data is 0.8903743

(c) Now using the `lda()` function from the `MASS` package we determine the accuracy and confusion matrix and compare the results with part (b)

```
library(MASS)
model_q1c <- lda(Digit ~ ., data = train_reduced)
model_pred <- predict(model_q1c, test_reduced)
predic_classes <- model_pred$class
cm2 <- table(Predicted = predic_classes, Actual = test_reduced$Digit)
cm2
```

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

```
acc2 <- mean(predic_classes == test_reduced$Digit)
acc2
```

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

Based on the output of the fitted `lda()` model using the `MASS` package and comparing it to the output of the model from (b), we see that the model output is identical for both which suggests that the custom implementation does a fair job of fitting an LDA model to the data.

(d) Now using the `qda()` function from the `MASS` package we fit a QDA and determine if the code works directly or not, and discuss possible implementations in the case that the code does not work:

```
train_reduced$Digit <- as.factor(train_reduced$Digit)
test_reduced$Digit <- as.factor(test_reduced$Digit)
qda_model <- qda(Digit ~ ., data = train_reduced)
```

```
## Error in qda.default(x, grouping, ...): some group is too small for 'qda'
```

As can be seen, running this code leads to the error that suggests that some classes in the dataset are too small, which leads to an inability to estimate the class-specific covariance matrices. This means that the sample size of some class must at least be equal to the number of features. Some solutions that might be viable to implement:

- Reducinng the number of features p so that it is elss than the smallest number of observations in any class n_k this will ensure that the covariance matrices estimated for each class are invertible. One way to do this would be through feature selection. We begin by selecting a smaller number of pixels (perhaps top 100) with the highest variance. Then we choose the features with the strongest relationship to the target variable by testing. We could also use PCA by computing the principle components of the training data and selecting the top d components that capture most of the variance where $d < n_k$ for all classes. PCA reduces the dimension by transforming the original data into a new set of variables. The covariance matrix now has dimensions $d \times d$ which is a square matrix and can be inverted.
- Another method could be the regularization of the covariance matrix. We add a small value to the diagonal performing ridge regularization:

  $$\Sigma_k' = \Sigma_k + \lambda \cdot I_n$$

  Where Σ_k is the covariance matrix for class k, $\lambda > 0$ is a small regularization constant and I_n is an $n \times n$ identity matrix. This ensures that all eigenvalues of the new regularized matrix are positive making it invertible.

These would be two solutions to address the issues from the QDA that I would implement in my own code.

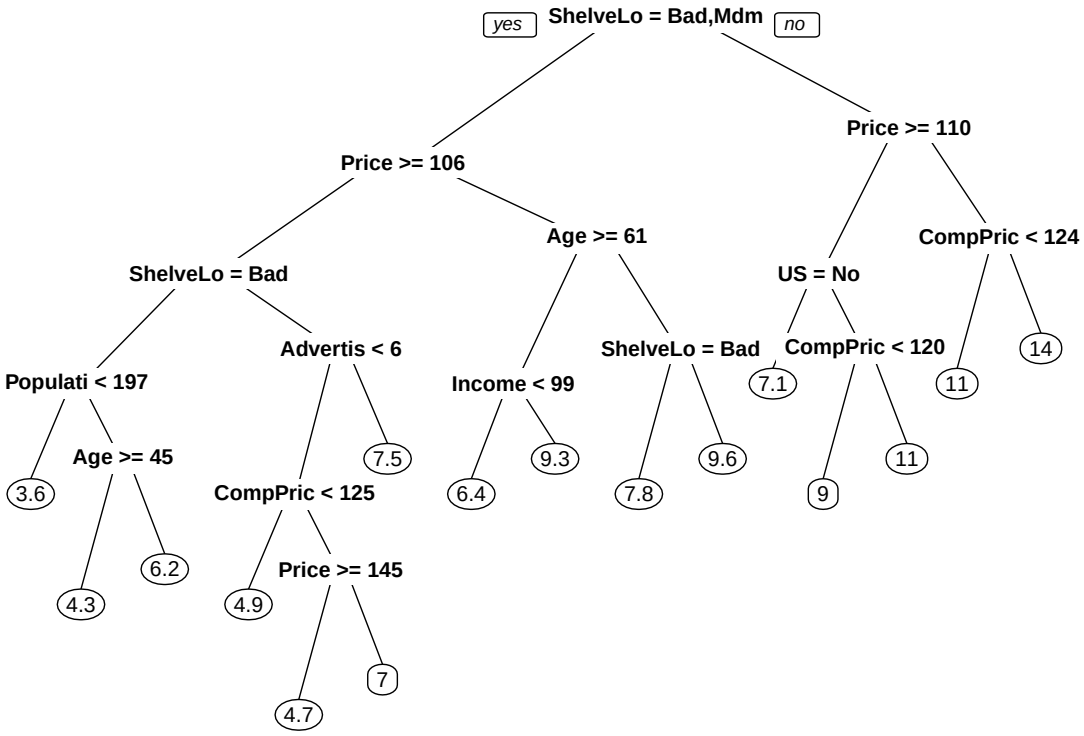
Question 2

(a) We begin by loading the `Carseats` data from the `ISLR` package and use the code provided in the question to define the training and test set:

```
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,]
```

We now use the `rpart` and `rpart.plot` libraries to plot a tree using `prp()` function and using the model to determine what type of observations have the highest and loswest sales. We also calculate and report the MSE on the testing data:

```
library(rpart)
library(rpart.plot)

tree_q2a <- rpart(Sales ~ ., data = train_Carseats)
prp(tree_q2a)
```




```
predictions <- predict(tree_q2a, test_Carseats)
mse <- mean((test_Carseats$Sales - predictions)^2)
mse
```

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

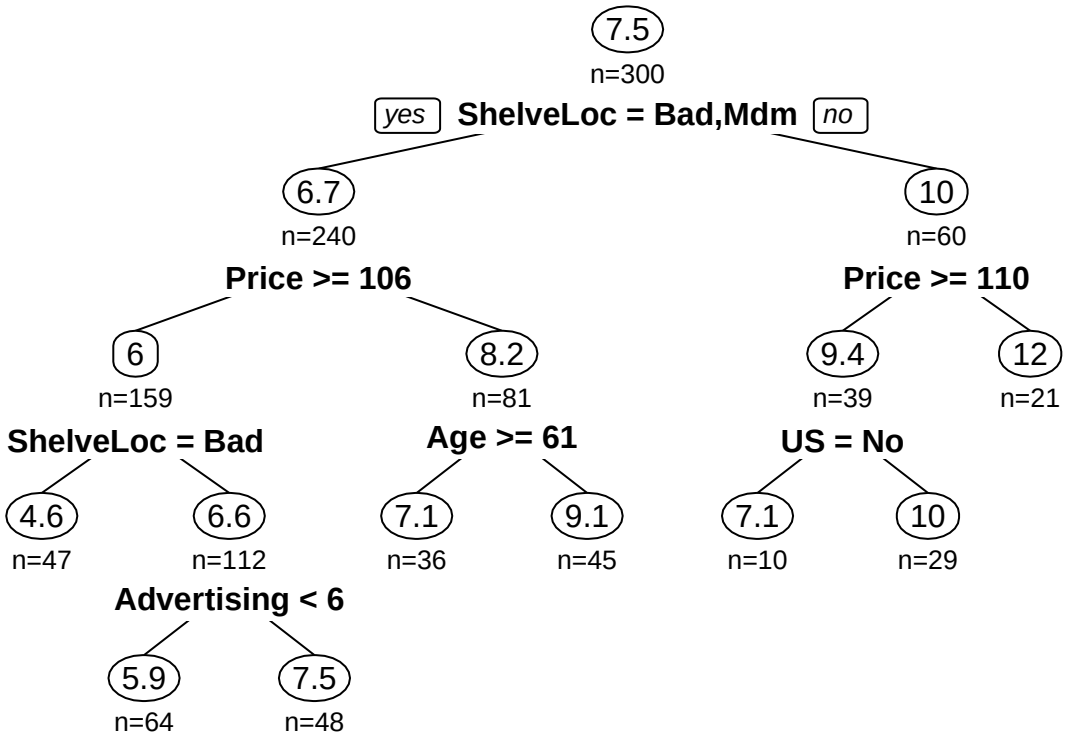
From the output see that 4.51347 is the MSE. Based on the tree, the observations with the highest predicted sales are when the shelf location is good (neither bad nor medium), the product is price is high (above $110), and the competitor's price is lower than $124. On the other hand, the lowest predicted sales are when the shelf location is bad, the price is high (above $106), the population is smaller, and the advertising budget is below 6. Another interesting observation is that while advertising and competitor price influence sales, a good shelf location and a higher price are the most important factors leading to higher sales and a bad shelf location and reduced advertising lead to lower sales.

(b) Now we find the largest complexity parameter value for the tree in part (a) and report it. And then we prune the tree using the found complexity parameter on the test set and report the test MSE:

```
cv_results <- tree_q2a$cptable
min_xerror <- min(cv_results[,"xerror"])
min_xstd <- cv_results[which.min(cv_results[,"xerror"]), "xstd"]
threshold <- min_xerror + min_xstd
optimal_cp <- max(cv_results[cv_results[,"xerror"] < threshold, "CP"])
optimal_cp
```

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

```
pruned_tree <- prune(tree_q2a, cp = optimal_cp)
prp(pruned_tree, type = 2, extra = 1, under = TRUE, varlen = 0)
```




```
prune_pred <- predict(pruned_tree, test_Carseats)
prune_mse <- mean((test_Carseats$Sales - prune_pred)^2)
prune_mse
```

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

The largest complexity parameter value for the tree grown in part (a) to ensure that the cross-validation error $<$ min(cross-validation error) + cross-validation standard deviation is 0.02173028, and the test MSE is 4.543565