Math for Data Science: Problem Set 4

Name: Giorgio Coppola, Group: Giorgio Coppola, Lino Hans Julian Zurmuehl, Luis Fernando Ramirez

2023-27-11

Due Date: Friday, December 8 by the end of the day. (The Moodle submission link will become inactive at midnight of December 9.)

Instructions: Please submit one solution set per person and include your name and your group members' names at the top. This time, please write your solutions within this Rmd file, under the relevant question. Please submit the knitted output as a pdf. Make sure to show all code you used to arrive at the answer. However, please provide a brief, clear answer to every question rather than making us infer it from your output, and please avoid printing unnecessary output.

1. More Cat Compression

(25 points.) I have another cat named Lev. He was upset that Laszlo got to be in my course materials and he didn't. In this exercise we will explain to Lev why my initial choice of Laszlo was nothing personal.

Figure 1: My other cat, Lev.

a. (3 points) Load the image of Lev. Extract its red, green, and blue matrices and store them as separate objects called `r`, `g`, and `b`. Center the blue matrix and compute its variance-covariance matrix. What is the dimensionality of this variance-covariance matrix and why?

```
# upload and plot Lev
lev <- readJPEG("lev.jpg")

# store rgb components in separate matrices
r_lev <- lev[,,1]
g_lev <- lev[,,2]
b_lev <- lev[,,3]

# center the data
b_centered_lev <- scale(b_lev, center = TRUE, scale = FALSE)

# check that it worked
# round(apply(b_centered_lev, 2, mean), 5)
```

```r
# make variance-covariance matrix
vc_lev <- cov(b_centered_lev)

# check dimensions
dim(vc_lev)
```

```
## [1] 200 200
```

Answer:

The variance-covariance matrix is n x n (200 x 200). A variance-covariance (vc) matrix is a systematic way to describe the variance and covariance of relationships between multiple variables in a data set. More precisely, a vc matrix is the sum of the matrices derived from the multiplication of a vector representing a row of the original matrix with its transpose (the sum is then divided by the total number of observations). Therefore, a vc matrix for a dataset is a sum of the vc matrices for each observation (or row) in the dataset. As said, each vc matrix for each row is derived by the multiplication of the vector representing the entries for each row, and its transpose. Therefore, we will have a nxp pxn multiplication. In this case, the two vectors are 1x200 and 200x1. The multiplication gives a pxp matrix, namely 200x200.

b. (3 points) Find the total variance of this image. Compare this to the total variance of the centered blue matrix for Laszlo.

```r
# Lev's variance
var_total_lev <- sum(apply(b_centered_lev, 2,
                           function(x) sum(x^2))) / nrow(b_centered_lev)

# Lazlo's variance
laszlo <- readJPEG("laszlo.jpg")
b_lazlo <- laszlo[,,3]
b_centered_lazlo <- scale(b_lazlo, center = TRUE, scale = FALSE)
var_total_lazlo <- sum(apply(b_centered_lazlo, 2,
                             function(x) sum(x^2))) / nrow(b_centered_lazlo)

# comparison
comparison_var <- data.frame(Lev_variance = var_total_lev,
                             Lazlo_variance = var_total_lazlo)
comparison_var
```

```
##   Lev_variance Lazlo_variance
## 1     9.441113       12.88591
```

Answer:

The total variance of the blue is less for Lev (9.441113) compared to Lazlo's (12.88591) image. If the variance is less, we can expect that we need more principal components to compress the image in a way that is recognizable.

c. (3 points) Use the `eigen` command to get the eigendecomposition of the blue variance-covariance matrix for Lev. Multiply the centered blue data matrix by the eigenvector corresponding to the largest eigenvalue to get the first principal component. Compute its variance.

```r
eigs_lev <- eigen(vc_lev)
pcr_1_lev <- b_centered_lev %*% eigs_lev$vectors[,1]
var_pcr_1_lev <- sum(pcr_1_lev^2) / length(pcr_1_lev)

var_pcr_1_lev
```

```
## [1] 6.31684
```

Answer:

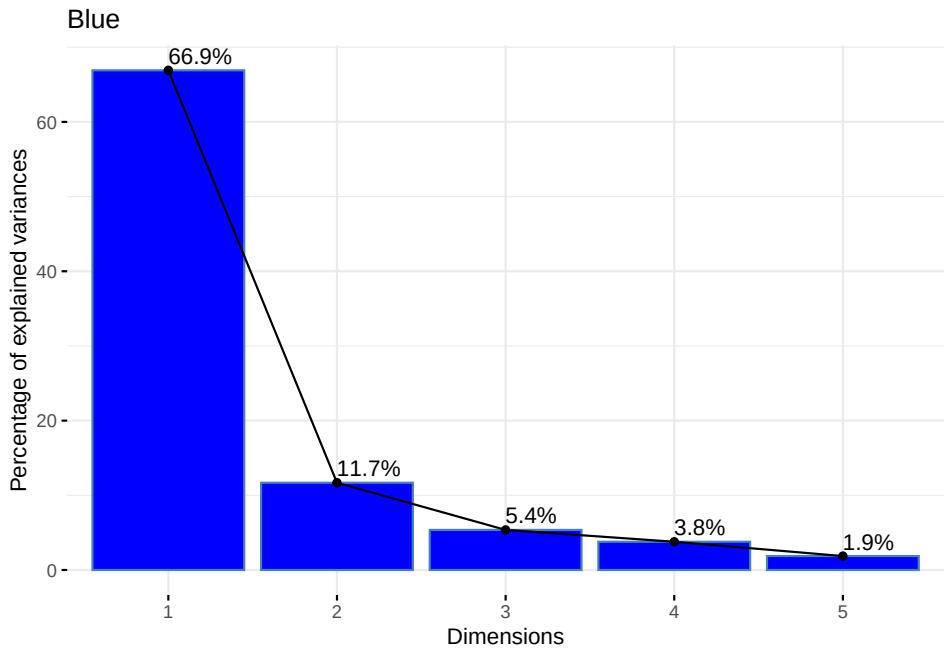
The variance of the first principal component for the blue channel of the Lev image is approximately 6.317.

d. (3 points) Use the answers to the previous two questions to compute the proportion of variance explained for Lev's first principal component. Check your answer against a scree plot produced by the `fviz_eig` command.[1] Compare to what we saw for Laszlo in the lab.

```
prop_var_pca_lec <- var_pcr_1_lev/var_total_lev
prop_var_pca_lec
```

```
## [1] 0.6690778
```

```
b_pca_lev <- prcomp(b_lev, center = TRUE, scale. = FALSE)
plot_pca_lev <- fviz_eig(b_pca_lev, main = "Blue",
                         barfill = "blue", ncp = 5, addlabels = TRUE)
```



Answer:

The proportion of variance explained by the first principal component is approximately 66.91%. The same result is given using the two methods.

e. (3 points) Now compute Lev's second principal component. Compute the covariance of the first principal component with the second.[2] Is this what you expected? Comment briefly.

```
pcr_2_lev <- b_centered_lev %*% eigs_lev$vectors[,2]
var_pcr_2_lev <- sum(pcr_2_lev^2) / length(pcr_2_lev)

cov_prc_lev <- cov(pcr_1_lev, pcr_2_lev)
round(cov_prc_lev, 10)
```

```
##      [,1]
## [1,]    0
```

Answer:

[1] If they are similar up to the second decimal place, that's good enough.
[2] You can round to 10 decimal places.

The result of the covariance is effectively zero. This is precisely what you would expect in a PCA, since the principal components are to each other, meaning they should be uncorrelated. The covariance between two uncorrelated variables (or principal components, in this case) should ideally be zero, and so it is!

f. (5 points) Now let's run PCA on Lev's `r`, `g`, and `b` matrices using the `prcomp` function with the options `center=FALSE` and `scale.=FALSE`.[3] Combine these objects into a list. Now, looping over a handful of numbers of principal components, reconstitute images of Lev as we did in Lab 9. How many principal components does it take to start to recognize Lev as a cat?

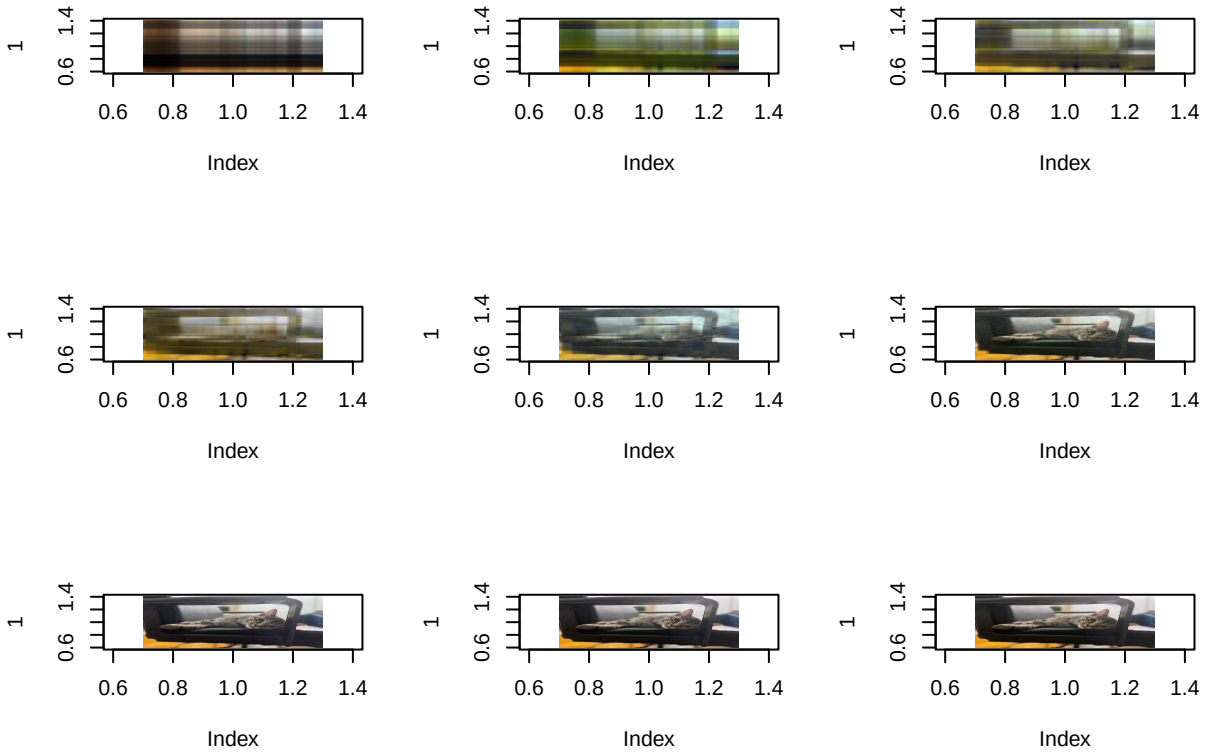
```
r_pca <- prcomp(r_lev, center = FALSE, scale. = FALSE)
g_pca <- prcomp(g_lev, center = FALSE, scale. = FALSE)
b_pca <- prcomp(b_lev, center = FALSE, scale. = FALSE)

rgb_pca <- list(r_pca, g_pca, b_pca)

expanded_1 <- sapply(rgb_pca, function(j) {
  # recompose the compressed image
  new_RGB <- j$x[,1] %*% t(j$rotation[,1])
  # rescale to between 0 and 1
  new_RGB <- (new_RGB - min(new_RGB)) / (max(new_RGB) - min(new_RGB))
}, simplify = "array")

plot(1, type = "n")
rasterImage(expanded_1, 0.7, 0.6, 1.3, 1.4)

dim(lev)
```

```
## [1] 150 200   3
```

```
vec <- c(1, 2, 3, 4, 5, 10, 20, 50, 100, 150)

# In PCA the maximum number of principal components that can be extracted is
# equal to the minimum of the number of rows and columns of the data matrix.
# For Lev, each color channel (red, green, blue) is effectively a 150 x 200 matrix,
# so we can calculate a maximum of 150 principal components.  For Lazlo, it was
# 267 x 200, so we were able to calculate 200 principal components.

max_components <- min(sapply(rgb_pca, function(j) min(ncol(j$x),
                                                      ncol(j$rotation))))
for(i in vec) {
  # Check if i is within bounds
  if (i <= max_components) {
    photo_pca <- sapply(rgb_pca, function(j) {
      # recompose the compressed image
      new_RGB <- j$x[, 1:i] %*% t(j$rotation[, 1:i])
      # rescale to between 0 and 1
      new_RGB <- (new_RGB - min(new_RGB)) / (max(new_RGB) - min(new_RGB))
      return(new_RGB)
    }, simplify = "array")
    assign(paste("photo_", round(i, 0), sep = ""), photo_pca)
  }
}

par(mfrow=c(3,3))
```

[3] Scaling and centering is desirable when your variables are on different scales, but in this case it messes up the colors.

```
plot(1, type = "n")
rasterImage(photo_1, 0.7, 0.6, 1.3, 1.4)
plot(1, type = "n")
rasterImage(photo_2, 0.7, 0.6, 1.3, 1.4)
plot(1, type = "n")
rasterImage(photo_3, 0.7, 0.6, 1.3, 1.4)
plot(1, type = "n")
rasterImage(photo_5, 0.7, 0.6, 1.3, 1.4)
plot(1, type = "n")
rasterImage(photo_10, 0.7, 0.6, 1.3, 1.4)
plot(1, type = "n")
rasterImage(photo_20, 0.7, 0.6, 1.3, 1.4)
plot(1, type = "n")
rasterImage(photo_50, 0.7, 0.6, 1.3, 1.4)
plot(1, type = "n")
rasterImage(photo_100, 0.7, 0.6, 1.3, 1.4)
plot(1, type = "n")
rasterImage(photo_150, 0.7, 0.6, 1.3, 1.4)
```



Answer:

At the 10th principal component, you can guess that the image represents a cat. At the 20th, you could probably recognize it's Lev! For Lazlo, we could guess it was a cat from the 3rd or 5th principal component already, and you could recognize Lazlo already with the 10th!

g. (5 points) Using what you learned from parts (b), (d), and (f) above, explain why Lev requires more principal components to be minimally represented than Laszlo, despite being equally beautiful.

Answer:

Even though they are clearly equally beautiful, Lev require some more principal components to be represented because Lev's image has less variance compared to Lazlo's image. Indeed, for Lev, at least for the color blue

matrix, the first principal component explain the 66.91% of the variance for the color blue. For Lazlo, the first prinipal component explained the 76.7% of the variation. For Lev, the successive principal components explain relatively more compared to the successive principal components for Lazlo. Let's see if the variance is minor for all the colors of the image.

```
r_lazlo <- laszlo[,,1]
g_lazlo <- laszlo[,,2]
# b_lazlo
# r_lev
# g_lev
# b_lev

# b_centered_lev
r_centered_lev <- scale(r_lev, center = TRUE, scale = FALSE)
g_centered_lev <- scale(g_lev, center = TRUE, scale = FALSE)

r_centered_lazlo <- scale(r_lazlo, center = TRUE, scale = FALSE)
g_centered_lazlo <- scale(g_lazlo, center = TRUE, scale = FALSE)
b_centered_lazlo <- scale(b_lazlo, center = TRUE, scale = FALSE)

var_total_lev_r <- sum(apply(r_centered_lev, 2,
                             function(x) sum(x^2))) / nrow(r_centered_lev)
var_total_lazlo_r <- sum(apply(r_centered_lazlo, 2,
                               function(x) sum(x^2))) / nrow(r_centered_lazlo)
var_total_lev_r
```

```
## [1] 16.1877
```

```
var_total_lazlo_r
```

```
## [1] 17.12905
```

```
var_total_lev_b <- sum(apply(b_centered_lev, 2,
                             function(x) sum(x^2))) / nrow(b_centered_lev)
var_total_lazlo_b <- sum(apply(b_centered_lazlo, 2,
                               function(x) sum(x^2))) / nrow(b_centered_lazlo)
var_total_lev_b
```

```
## [1] 9.441113
```

```
var_total_lazlo_b
```

```
## [1] 12.88591
```

```
var_total_lev_g <- sum(apply(g_centered_lev, 2,
                             function(x) sum(x^2))) / nrow(g_centered_lev)
var_total_lazlo_g <- sum(apply(g_centered_lazlo, 2,
                               function(x) sum(x^2))) / nrow(g_centered_lazlo)
var_total_lev_g
```

```
## [1] 12.2169
```

```
var_total_lazlo_g
```

```
## [1] 13.37922
```

The variance is less for all the colors. Indeed, in general, the image of Lev is less contrasted compared to the image of Lazlo: with its black coat, Lazlo stand out against the background. Lev instead camouflages itself better with the background, so that the variance of the picture is minor. This is the only reason why it

requires more principal components to be recognized (nothing to do with the beauty of the cats!).

2. Penalized Regression

(25 points.) We will derive the estimator for ridge regression, which is one of several *penalized regression* methods.[4] The process we will follow is very similar to the standard regression estimator we derived in Lab 8, but with a small change to the loss function. Rather than minimizing the sum of squared errors, we will minimize the sum of squared errors subject to a constraint:

$$||\beta||_2^2 \leq s$$

where s is just some constant chosen by the analyst. Recall that $||\beta||_2^2$ is the squared L_2 norm of the β vector.[5] This question will build on the concepts and data in Lab 8, so please revisit that lab if anything here is unclear.

a. (2 points) Write down the Lagrangian for this constrained minimization problem. Please use matrix notation (including for the L_2 norm) as we did in Lab 8.

The linear regression is a minimization problem. We want to minimize the sum of the squared errors to find the best fitted line through the data defined by the linear model having β as coefficients. The minimization problem is written as $\min_{\beta_0,\beta_1,\ldots,\beta_p} \sum_{i=1}^{n} \epsilon_i^2$. In matrix notation, this is equivalent to $(Y - X\beta)^\top (Y - X\beta)$, which in L2 (or Euclidian) form is $||Y - X\beta||_2^2$.

For some reason, we want to penalize the linear regression model (for example to address overfitting, multicollinearity, or increase the interpretability of the model). Perhaps we prefer to perform penalize regression instead of just dropping some variables causing (e.g.) multicollinearity because we might think that those variables are still fundamental to explain some fact (we would fall into the omitted variable bias otherwise).

The quadratic nature of the L2 norm shrinks the coefficients towards zero, thereby reducing model complexity. The norm of a vector is the measure of its magnitude, and in ridge regression, the L2 norm is used to penalize the magnitude of the coefficient vector β, to prevent any individual coefficient from becoming too large. The constraint, in this case, is that the sum of the squares of the β coefficients is restricted to be less than or equal to a certain threshold (s).

The constraint is to be incorporated into the Lagrangian:

$$L(\beta) = ||\mathbf{Y} - \mathbf{X}\beta||_2^2 + \lambda(||\beta||_2^2 - s)$$

or

$$L(\beta) = (\mathbf{Y} - \mathbf{X}\beta)^T(\mathbf{Y} - \mathbf{X}\beta) + \lambda(\beta^T\beta - s)$$

where:

- $||Y - X\beta||^2$ is the loss function in OLS (how well the model defined by β fits the data, with $||Y||$ being the response vector, $||X||$ the design matrix),
- $\lambda(\|\beta\|_2^2 - s)$ is the penalty term. s sets a threshold for the size of the coefficients and λ adjusts the strength of the penalty relative to this threshold.

b. (7 points) Take the first derivative of the Lagrangian with respect to β and set it equal to $\mathbf{0}$ to get the first order condition. Use the Matrix Cookbook to help you. Solve for $\hat{\beta}$.

Our solution must lie on $||\beta||_2^2 \leq s$. This is our constraint. Let's build the constraint as a function $g(x) = 0 : ||\beta||_2^2 - s \leq 0$. This mean: $g(x) = 0 : \beta^\top\beta - s \leq 0$.

[4] See p. 237-244 of ISL.

[5] Also known as the Euclidian norm.

So, putting all together we have:

$$\begin{aligned} L(\beta) &= (Y - X\beta)^\top (Y - X\beta) + \lambda\left(\beta^\top \beta - s\right) \\ &= Y^\top Y - Y^\top X\beta - \beta^\top X^\top Y + \beta^\top X^\top X\beta + \lambda\left(\beta^\top \beta - s\right) \\ &= Y^\top Y - 2Y^\top X\beta + \beta^\top X^\top X\beta + \lambda\left(\beta^\top \beta - s\right) \end{aligned}$$

Now, we can take the derivative in respect to β ...

$$\frac{\partial L}{\partial \beta}(L(\beta)) = -2X^\top Y + X^\top X 2\beta + \lambda 2\beta$$

... set the derivative to zero and resolve for β:

$$\begin{aligned} -2X^\top Y + 2X^\top X\beta + 2\lambda\beta &= 0 \\ 2X^\top X\beta + 2\lambda\beta &= 2X^\top Y \\ X^\top X\beta + \lambda\beta &= X^\top Y \\ \left(X^\top X + \lambda I\right)\beta &= X^\top Y \\ \hat{\beta} &= X^\top Y \left(X^\top X + \lambda I\right)^{-1} \end{aligned}$$

We isolated $\hat{\beta} = X^\top Y \left(X^\top X + \lambda I\right)^{-1}$.

c. (3 points) Compare your answer to the standard linear regression estimator. Under what condition are they the same?

Since λ represents the strength of the penalty, if $\lambda = 0$, the penalized regression is equal to the standard linear regression, as the value of the constraint (or penalty) would be zero.

d. (5 points) Using the same `BostonHousing` dataset and the same set of variables from Lab 8, compute the ridge regression $\hat{\beta}$ with the equation you found in part (b) above. First standardize your $\mathbf{X}$ matrix using the `scale()` function in `R`. Use the `glmnet()` function in the `glmnet` package to check your answer.[6]

```
# data preparation and manipulation
data(BostonHousing)
data <- BostonHousing

data_clean <- data |>
  dplyr::select(crim, chas, nox, dis, ptratio, rad, medv)

# our data set, set of dependent variables
# this is the design matrix
BH_matr <- as.matrix(data_clean |> dplyr::select(-medv))
BH_matr <- apply(BH_matr, 2, as.numeric)
X <- scale(BH_matr)
X <- cbind(intercept = 1, X)
```

[6]Note that `glmnet`'s `lambda` parameter corresponds to your $\frac{\lambda}{N}$, where N is the number of rows of your data. Also, your estimates may differ slightly from `glmnet`'s, at around the second decimal place. That's alright; this is likely due to `glmnet`'s optimization algorithm. It's computationally expensive to invert large matrices so `glmnet` is probably taking some more efficient but (slightly) less precise approach.

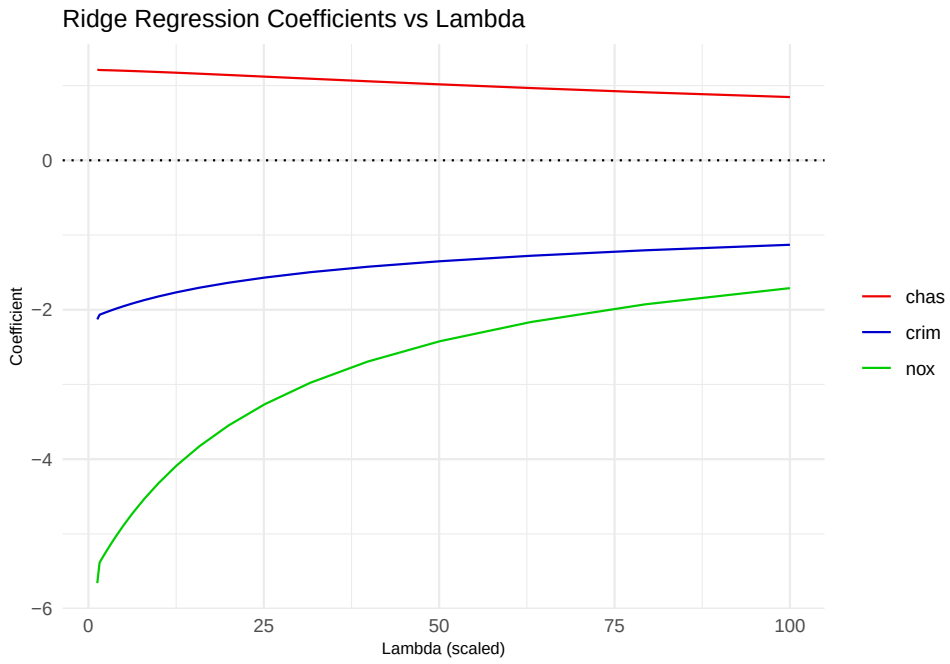
```r
# our independent variable 'medv': median value of owner-occupied homes
# this is the response vector
Y <- as.vector(data_clean$medv)

# define lambda
lambda_value <- 1
lambda_scaled <- lambda_value / nrow(X)

# create the identity of the design matrix
I <- diag(ncol(X))

# manual computation of beta-hat
beta_hat <- solve(t(X) %*% X + lambda_value * I) %*% t(X) %*% Y

# using glmnet
glmnet_fit <- glmnet(X, Y, alpha = 0, lambda = lambda_scaled)
beta_glmnet <- coef(glmnet_fit, s = lambda_scaled)

# comparing
comparison <- data.frame(Manual = beta_hat, glmnet = beta_glmnet[-2])
comparison
```

```
##              Manual    glmnet
## intercept 22.488363 22.532806
## crim      -2.117056 -2.127686
## chas       1.211308  1.211766
## nox       -5.608049 -5.651956
## dis       -2.763344 -2.796078
## ptratio   -4.372866 -4.389615
## rad        1.914031  1.942497
```

e. (3 points) For the sequence of lambdas below, make a plot with `lambda_seq` on the x-axis (increasing from 0 to 100) and the estimated β coefficients for per capita crime rate in the town (in blue), proximity to the Charles River (in red), and nitric oxides concentration (in green) on the y-axis. Use `geom_line` to plot the coefficients and add a black horizontal line at $y = 0$. Label your axes and include a legend.

```r
lambda_seq <- c(10^seq(2, -1, by = -.1), 0)
lambda_seq <- (lambda_seq[order(lambda_seq)])

glmnet_seq <- glmnet(X, Y, alpha = 0, lambda = lambda_seq)
beta_seq_mtr <- coef(glmnet_seq, s = lambda_seq) |>
  as.matrix()

beta_df <- as.data.frame(beta_seq_mtr, row.names = rownames(beta_seq_mtr)) |>
  tibble::rownames_to_column('variable') |>
  dplyr::select(variable, everything()) |>
  filter(variable %in% c("crim", "chas", "nox"))

beta_long <- beta_df %>%
  pivot_longer(cols = -variable, names_to = "lambda", values_to = "coefficient",
               names_prefix = "s", names_transform = list(lambda = as.numeric))  |>
  mutate(lambda = (10^(lambda / 10))) |>
  filter(lambda <= 100)
```

```r
plot_seq <-  ggplot(beta_long, aes(x = lambda, y = coefficient, color = variable)) +
  geom_line() +
  geom_hline(yintercept = 0, linetype = "dotted", color = "black")  +
  scale_color_manual(values = c("crim" = "blue3", "chas" = "red2", "nox" = "green3")) +
  labs(x = "Lambda (scaled)", y = "Coefficient", title = "Ridge Regression Coefficients vs Lambda") +
  theme_minimal() +
  theme(legend.title = element_blank(),
                plot.title = ggplot2::element_text(size = 12),
                axis.title.x = ggplot2::element_text(size = 8),
                axis.title.y = ggplot2::element_text(size = 8))
```



f. (5 points) Discussion:

- Based on your plot above, explain why ridge is one of a number of so-called "shrinkage" estimators.

Ridge regression is referred to as a "shrinkage" estimator because it applies a penalty to the coefficients of the regression model which "shrinks" them towards zero as the penalty grows large. This penalty is the (squared) magnitude of the coefficients (the L2 norm). Therefore, we are effectively forcing the β coefficients to be limited to a constraint (which in this case in $\leq s$). Hence, the coefficients are subjected to the constraint of being less than a certain threshold s. This is likely to introduce some bias, since if the coefficient is constrained, the predicted sum of squared error will not be completely minimized. This constraint (or alternatively, a penalty) is controlled by the parameter λ, which define the "relevancy" of the constraint. From the plot, we can clearly see that the three variables in question converge towards zero as λ grows large. If we penalize, we chose λ directly, while if we constrain, we chose s.

- Tie this back to the constrained optimization problem you solved to obtain the ridge regression $\hat{\beta}$. Can you see how a larger value of λ (or equivalently a small value of s) corresponds to greater shrinkage?

Intuitively, when λ increase, the penalty term in the objective function becomes more significant compared to the residual sum of squares, and the optimization will increasingly prefer solutions with smaller coefficient magnitudes to minimize the overall objective, prioritizing the minimization of the magnitude of the coefficients over the minimization of the sum of squared errors. Conversely, when λ is small, the penalty term has less impact, and the solution will be closer to the ordinary least squares estimates, which means less shrinkage.

- You will learn more about this class of estimators and their virtues next semester, but do you have any

intuitions as to when and why they might be desirable?

As anticipated, they might be useful when we work with high dimensional data. For example, if we want to address multicollinearity without dropping variables because we think they are still important to explain a phenomena (otherwise we would fall into omitted variable bias), we can use ridge regression. Or when we are overfitting the model in some other way, ridge regression can be useful. It is probably also useful for prediction, as ridge models would intuitively make less extreme predictions, which can be beneficial when the goal is prediction on new data.