Penalized Regression

Patrick Kim

2024-03-10

Preliminaries ##
Load the packages and set seed.

```
library(ISLR2)
library(dplyr)
```

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

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

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

```
library(ggplot2)
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-8
```

```
library(ggrepel)
library(tidyr)
```

```
##
## Attaching package: 'tidyr'
```

```
## The following objects are masked from 'package:Matrix':
##
##     expand, pack, unpack
```

```
library(recipes)
```

```
## Warning: package 'recipes' was built under R version 4.2.3
```

```
##
## Attaching package: 'recipes'
```

```
## The following object is masked from 'package:Matrix':
##
##     update
```

```
## The following object is masked from 'package:stats':
##
##     step
```

```
library(vip)
```

```
##
## Attaching package: 'vip'
```

```
## The following object is masked from 'package:utils':
##
##     vi
```

```
library(tibble)
set.seed(295) # set seed for reproducibility.
```

We will be using "Hitters" data in the ISLR2 package, where the "salary" variable will be the response variable of our prediction model. Let's load the data and do simple exploratory analysis.

```
# Let's see what variables are contained in the data.
names(Hitters)
```

```
##  [1] "AtBat"     "Hits"      "HmRun"     "Runs"      "RBI"       "Walks"
##  [7] "Years"     "CAtBat"    "CHits"     "CHmRun"    "CRuns"     "CRBI"
## [13] "CWalks"    "League"    "Division"  "PutOuts"   "Assists"   "Errors"
## [19] "Salary"    "NewLeague"
```

```
# Check the dimension.
dim(Hitters) # There are 322 observations/rows and 20 variables/columns.
```

```
## [1] 322  20
```

```
# Since we want to predict the "Salary" value using appropriate predictors in the following sections, i
sum(is.na(Hitters$Salary)) # We have 59 missing values for "Salary". Let's proceed to drop all the rows
```

```
## [1] 59
```

```
Hitters <- na.omit(Hitters) # Drop the rows with missing values
dim(Hitters) # We can check that 59 defective rows are dropped from the original 322 rows, resulting in
```

```
## [1] 263  20
```

```
sum(is.na(Hitters))
```

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

Unlike simple/multivariate linear regression or kNN, penalized regression (Ridge, Lasso) functions in R take matrix as an input for predictors and a vector as an input for the response variable. The input matrix of predictors is called "design matrix", which has the first column of ones to create the constant term, b0, of the regression (take 336 or ask Prof. Wells/me if you want to know why). Thus, we need to extract variables from the dataframe (here, "Hitter") and put them into matrix. We will be using "model.matrix()" function to create a predictor matrix to be used as an input. %%% Side note: linear regression or kNN are formulated/computed in the terms of vectors and matrices, but the functions in R to implement them do not necessitate vector/matrix inputs. The functions do matrix/vector operations internally so that we don't recognize it.

```
X <- model.matrix(Salary ~., data = Hitters)[, -1] # I am telling the "model.matrix" function to put ev
y <- Hitters$Salary # Recall that data frames are collection of column vectors of different variables.
```

Ridge Regression ##
Now let's do the Ridge regression with the "glmnet" function. This function has four parameters. 1. A matrix of predictors 2. A vector of a response variable 3. alpha value, which determines regression type. Set "alpha = 0" for Ridge regression and "alpha = 1" for Lasso. 4. lambda value/vector for the tuning

parameter (also called "hyperparameter", think of it as a sensitivity of the model). Higher lambda will shrink the coefficients more in terms of L2 (Euclidean) norm and reduce their variance.

```
grid <- 10^seq(10, -2, length = 100) # Create a vector of lambda values. Since we are not doing cross-v
ridge.mod <- glmnet(
  x = X,
  y = y,
  alpha = 0,
  lambda = grid
) # Run the Ridge regression.
```

The resulting "ridge.mod" is very confusing since it doesn't look like outputs from linear regression or kNN. To access the coefficients, we need to call "coef(ridge.mod)".

```
# Check the dimension of the coefficient matrix
dim(coef(ridge.mod)) # It says we have 20 rows and 100 columns, which is consistent with the input of 1
```

```
## [1]  20 100
```

Recall that Ridge regression is designed to "penalize" the L2 norm of the beta coefficients. As the tuning parameter (lambda) increases, the model penalizes the L2 norm of the beta coefficients heavier, resulting in smaller coefficients. Let's check if our coefficients are consistent with the theory so that the choice of greater lambda leads to smaller beta coefficients.

```
# Check the lambda value for the 1st, 50th and 100th model
ridge.mod$lambda[1] # lambda = 1e+10 = 10000000000
```

```
## [1] 1e+10
```

```
ridge.mod$lambda[50] # lambda = 11497.57
```

```
## [1] 11497.57
```

```
ridge.mod$lambda[100] # lambda = 0.01
```

```
## [1] 0.01
```

```
# Access individual coefficient vectors for each lambda value chosen above
coef(ridge.mod)[, 1]
```

```
##   (Intercept)         AtBat          Hits         HmRun          Runs
##  5.359257e+02  5.443467e-08  1.974589e-07  7.956523e-07  3.339178e-07
##           RBI         Walks         Years        CAtBat         CHits
##  3.527222e-07  4.151323e-07  1.697711e-06  4.673743e-09  1.720071e-08
##        CHmRun         CRuns          CRBI        CWalks       LeagueN
##  1.297171e-07  3.450846e-08  3.561348e-08  3.767877e-08 -5.800263e-07
##     DivisionW       PutOuts       Assists        Errors    NewLeagueN
## -7.807263e-06  2.180288e-08  3.561198e-09 -1.660460e-08 -1.152288e-07
```

```
coef(ridge.mod)[, 50]
```

```
##   (Intercept)         AtBat          Hits         HmRun          Runs
## 407.356050200   0.036957182   0.138180344   0.524629976   0.230701523
##           RBI         Walks         Years        CAtBat         CHits
##   0.239841459   0.289618741   1.107702929   0.003131815   0.011653637
##        CHmRun         CRuns          CRBI        CWalks       LeagueN
##   0.087545670   0.023379882   0.024138320   0.025015421   0.085028114
##     DivisionW       PutOuts       Assists        Errors    NewLeagueN
##  -6.215440973   0.016482577   0.002612988  -0.020502690   0.301433531
```

```r
coef(ridge.mod)[, 100]
```

```
##   (Intercept)         AtBat          Hits         HmRun          Runs
##  164.11321606   -1.97386151    7.37772270    3.93660219   -2.19873625
##           RBI         Walks         Years        CAtBat         CHits
##    -0.91623008    6.20037718   -3.71403424   -0.17510063    0.21132772
##        CHmRun         CRuns          CRBI        CWalks       LeagueN
##     0.05629004    1.36605490    0.70965516   -0.79582173   63.40493257
##     DivisionW       PutOuts       Assists        Errors    NewLeagueN
## -117.08243713    0.28202541    0.37318482   -3.42400281  -25.99081928
```

```r
# We can see that as we move from 1 to 50 to 100, the coefficients get larger. Now let's check if the L
sqrt(sum(coef(ridge.mod)[-1, 1]^2)) # 8.080244e-06
```

```
## [1] 8.080244e-06
```

```r
sqrt(sum(coef(ridge.mod)[-1, 50]^2)) # 6.360612
```

```
## [1] 6.360612
```

```r
sqrt(sum(coef(ridge.mod)[-1, 100]^2)) # 136.2012
```

```
## [1] 136.2012
```

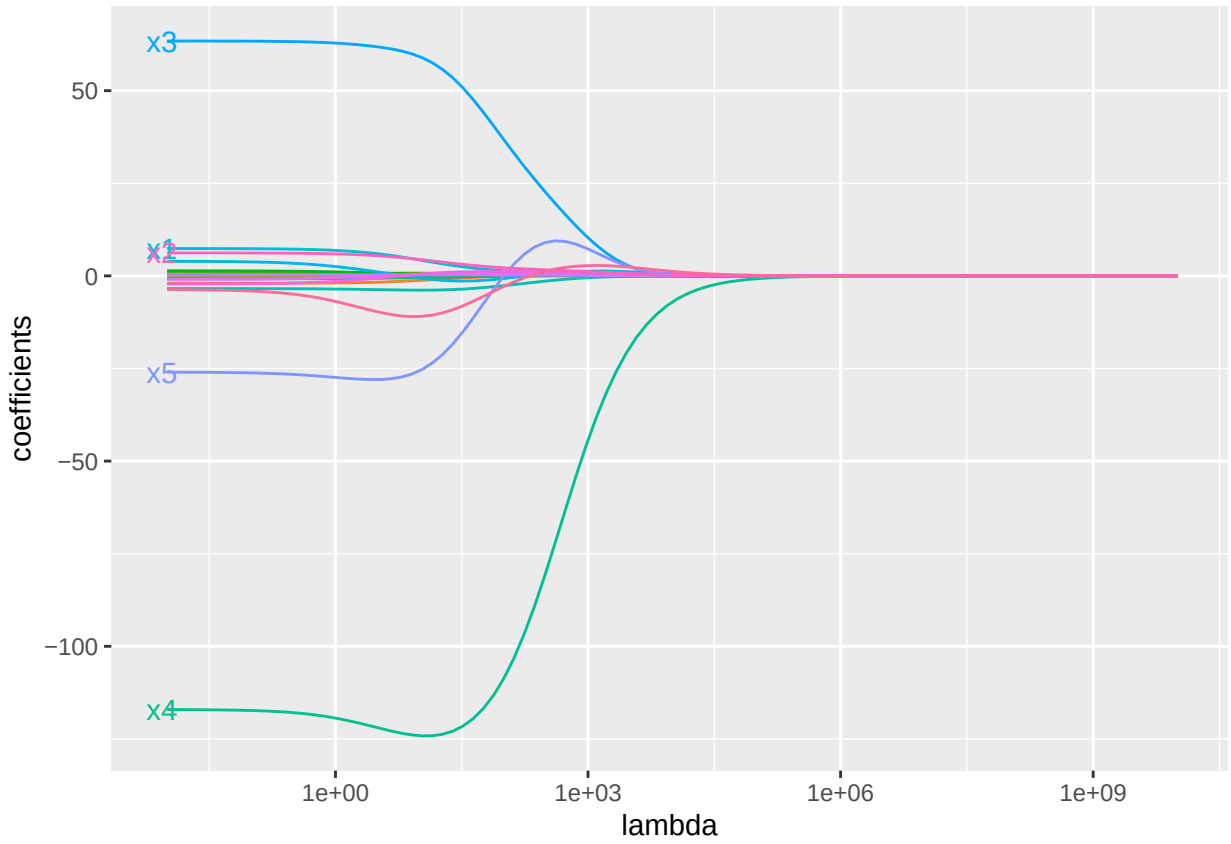
```r
# As we choose larger tuning parameter, the L2 norm of the regression coefficient gets penalized less,
lam <- grid %>%
  as.data.frame() %>%
  mutate(penalty = ridge.mod$a0 %>% names()) %>%
  rename(lambda = ".")

ridge.results <- ridge.mod$beta %>%
  as.matrix() %>%
  as.data.frame() %>%
  rownames_to_column() %>%
  gather(penalty, coefficients, -rowname) %>%
  left_join(lam)
```

```
## Joining with `by = join_by(penalty)`
```

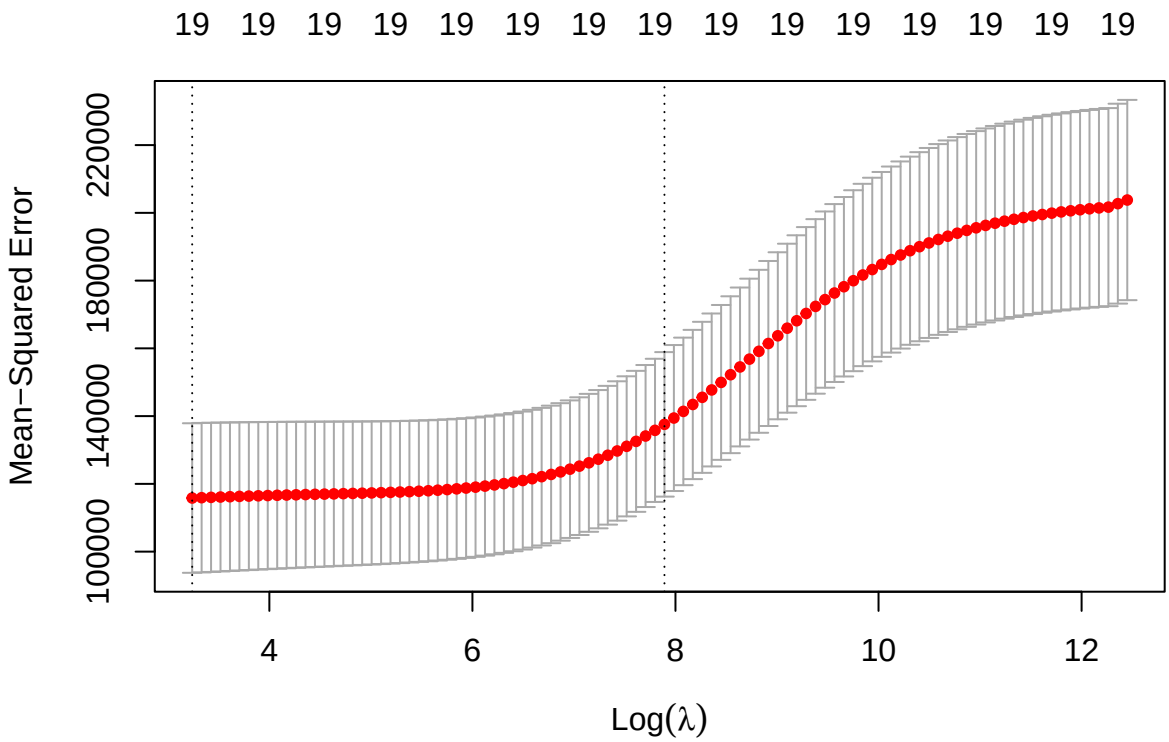
```r
result_labels <- ridge.results %>%
  group_by(rowname) %>%
  filter(lambda == min(lambda)) %>%
  ungroup() %>%
  top_n(5, wt = abs(coefficients)) %>%
  mutate(var = paste0("x", 1:5))

ggplot() +
  geom_line(data = ridge.results, aes(lambda, coefficients, group = rowname, color = rowname), show.leg
  scale_x_log10() +
  geom_text(data = result_labels, aes(lambda, coefficients, label = var, color = rowname), nudge_x = -.
```



Now let's cross-validate to choose the optimal tuning parameter

```
cv.ridge <- cv.glmnet(X, y, nfolds = 10, alpha = 0) # "cv.glmnet" does sample splitting and n-fold cros
plot(cv.ridge) # plot the cross-validation result to see how different choices of lambda results in dif
```



```
optimal.ridge.lambda <- cv.ridge$lambda.min # this saves the lambda value with the lowest MSE. Remember
optimal.ridge.lambda # explicitly, the lambda value that gives us the lowest MSE is lambda = 25.52821
```

```
## [1] 25.52821
```

Lasso Regression ##
Now let's do the Lasso regression with the "glmnet" function and X, y, and the vector of lambda values "grid" we created above. This function has four parameters. 1. A matrix of predictors 2. A vector of a response variable 3. alpha value, which determines regression type. Set "alpha = 0" for Ridge regression and "alpha = 1" for Lasso. 4. lambda value/vector for the tuning parameter (also called "hyperparameter", think of it as a sensitivity of the model). Higher lambda will shrink the coefficients more in terms of L1 (Manhattan) norm and induce "sparsity"-there will be more variables with zero coefficient so that they are not "selected" in the model.

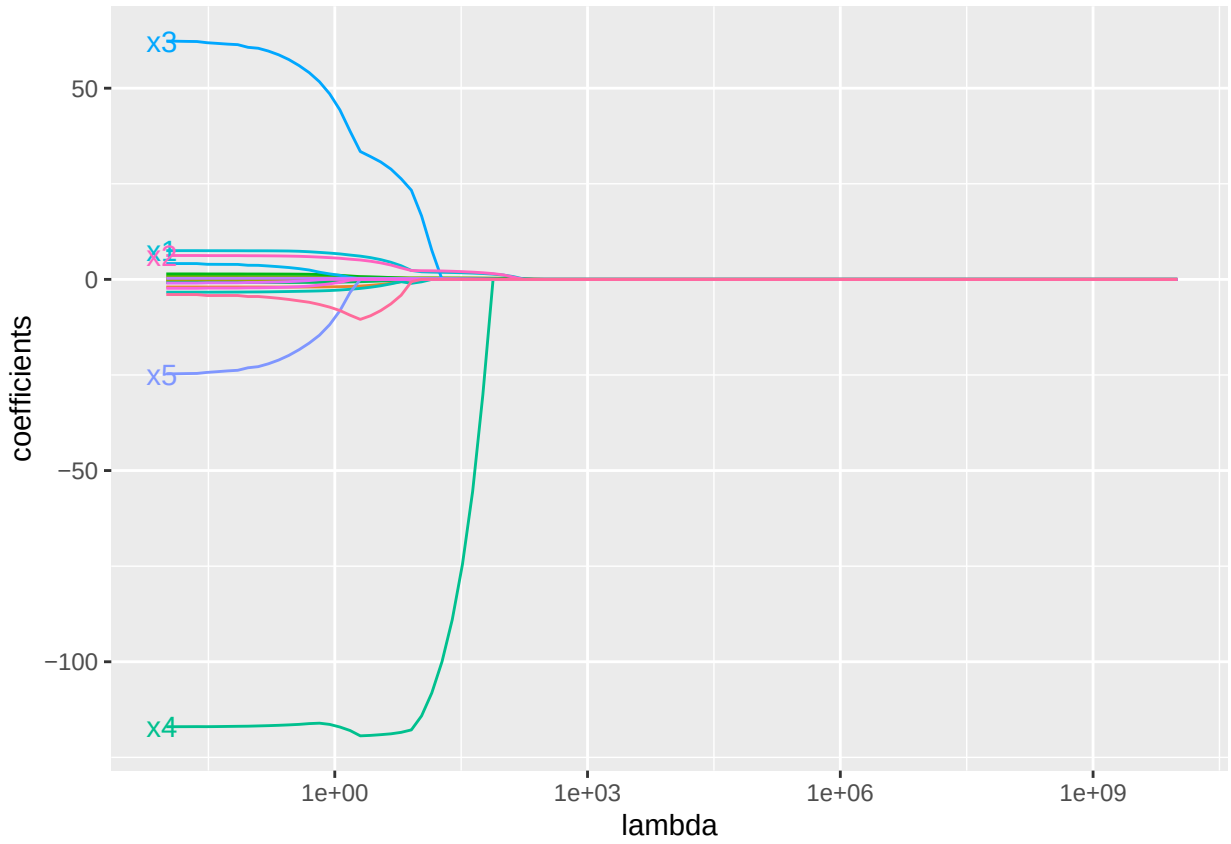
```
lasso.mod <- glmnet(
  X,
  y,
  alpha = 1,
  lambda = grid
) # Run the Lasso regression.

# The plot below shows how greater lambda results in smaller coefficients.
lasso.results <- lasso.mod$beta %>%
  as.matrix() %>%
  as.data.frame() %>%
  rownames_to_column() %>%
  gather(penalty, coefficients, -rowname) %>%
  left_join(lam)
```

```
## Joining with `by = join_by(penalty)`
```

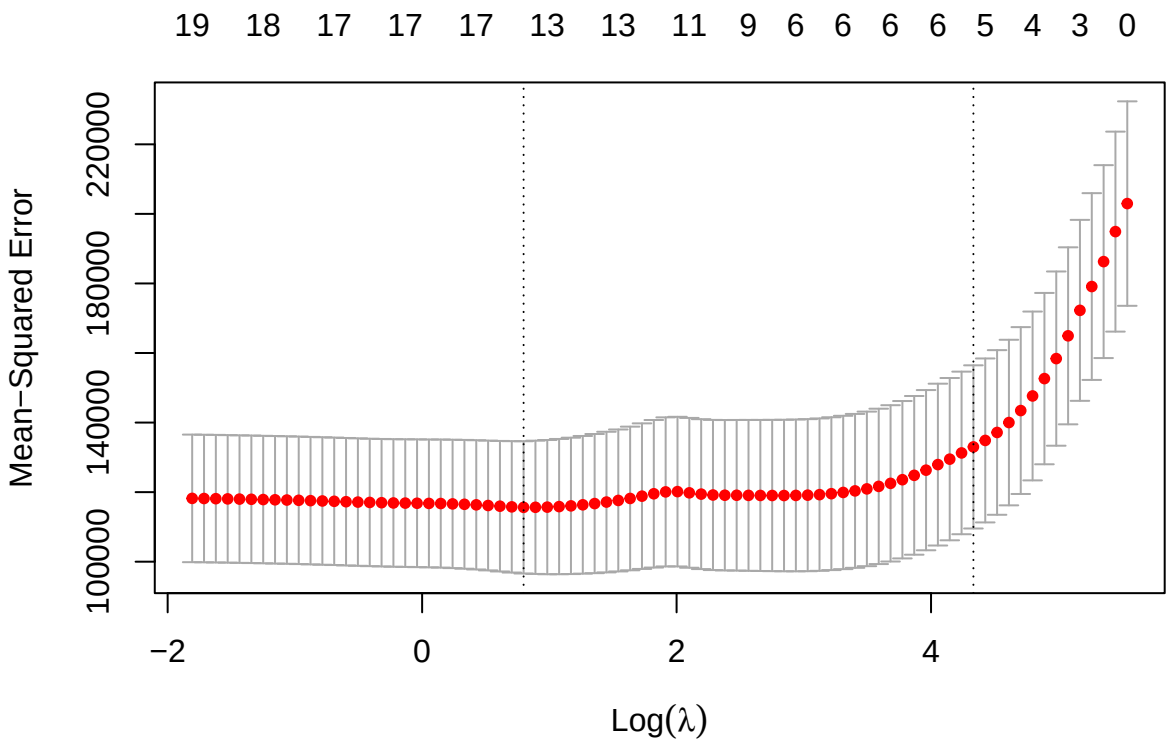
```
result_labels <- lasso.results %>%
  group_by(rowname) %>%
  filter(lambda == min(lambda)) %>%
  ungroup() %>%
  top_n(5, wt = abs(coefficients)) %>%
  mutate(var = paste0("x", 1:5))

ggplot() +
  geom_line(data = lasso.results, aes(lambda, coefficients, group = rowname, color = rowname), show.lege
  scale_x_log10() +
  geom_text(data = result_labels, aes(lambda, coefficients, label = var, color = rowname), nudge_x = -.
```



Analogous to the Ridge regression, the optimal tuning parameter lambda for Lasso can be chosen using the "cv.glmnet" function.

```
cv.lasso <- cv.glmnet(X, y, nfolds = 10, alpha = 1) # "cv.glmnet" does sample splitting and n-fold cros
plot(cv.lasso) # plot the cross-validation result to see how different choices of lambda results in dif
```




```
optimal.lasso.lambda <- cv.lasso$lambda.min # this saves the lambda value with the lowest MSE. Remember
optimal.lasso.lambda # explicitly, the lambda value that gives us the lowest MSE is lambda = 2.220313.
```

```
## [1] 2.220313
```

Recall that Lasso regression (1) shrinks the coefficients and (2) selects variables. With the optimal lambda chosen via cross-validation above (lambda = 2.220313), let's see which variables are selected.

```
optimal.lasso <- glmnet(
  X,
  y,
  alpha=1,
  lambda=2.220313
)
coef(optimal.lasso) # we can see that "Runs", "RBI", "CAtBat", and "CHits" are omitted in this model wi
```

```
## 20 x 1 sparse Matrix of class "dgCMatrix"
##                       s0
## (Intercept)  134.40580935
## AtBat         -1.69157543
## Hits           5.97284108
## HmRun          0.04943773
## Runs           .
## RBI            .
## Walks          4.99708657
## Years        -10.07327200
## CAtBat         .
## CHits          .
## CHmRun         0.59083783
## CRuns          0.71299766
## CRBI           0.37491789
## CWalks        -0.59215657
```

```
## LeagueN       33.10488183
## DivisionW   -119.19786791
## PutOuts        0.27640562
## Assists        0.19985017
## Errors        -2.24472742
## NewLeagueN     .
```

Basic example with matrix algebra in R: ordinary least squares regression ##############################

```
OLS <- function(X, y) {
  # Check if X and y are compatible in terms of dimensions for further matrix-vector operations
  if (nrow(X) != length(y)) {
    stop("The number of rows in X must be equal to the length of y.")
  }

  # Add a column of ones to X to account for the intercept term (this is called the design matrix)
  X <- cbind(1, X)

  # Calculate the OLS estimator beta_hat
  beta_hat <- solve(t(X) %*% X) %*% t(X) %*% y # here, the "solve" function computes the inverse of a g

  return(beta_hat)
}

# The OLS function above gives us the same result as the built-in lm function when y is regressed on X
lm(y ~ X)
```

```
##
## Call:
## lm(formula = y ~ X)
##
## Coefficients:
## (Intercept)       XAtBat        XHits       XHmRun        XRuns         XRBI
##    163.1036      -1.9799       7.5008       4.3309      -2.3762      -1.0450
##      XWalks       XYears      XCAtBat       XCHits      XCHmRun       XCRuns
##      6.2313      -3.4891      -0.1713       0.1340      -0.1729       1.4543
##       XCRBI      XCWalks     XLeagueN   XDivisionW     XPutOuts     XAssists
##      0.8077      -0.8116      62.5994    -116.8492       0.2819       0.3711
##     XErrors  XNewLeagueN
##     -3.3608     -24.7623
```

```
OLS(X, y)
```

```
##                   [,1]
##             163.1035878
## AtBat        -1.9798729
## Hits          7.5007675
## HmRun         4.3308829
## Runs         -2.3762100
## RBI          -1.0449620
## Walks         6.2312863
## Years        -3.4890543
## CAtBat       -0.1713405
## CHits         0.1339910
## CHmRun       -0.1728611
## CRuns         1.4543049
```

```
## CRBI          0.8077088
## CWalks       -0.8115709
## LeagueN      62.5994230
## DivisionW  -116.8492456
## PutOuts       0.2818925
## Assists       0.3710692
## Errors       -3.3607605
## NewLeagueN  -24.7623251
```