Linear models

Instructor: Maxim Fedotov*

September 2022

Linear regression

Linear regression is a fundamental foundation of statistics. Despite its observable simplicity, it is still used widely and is an origin of more sophisticated models.

Consider a setting where you have a dataset with n observations of a form $\{(y_i, \mathrm{x}_i)\}_{i=1}^n$ where y_i is an output variable. Linear regression assumes linear dependence between the output y and the vector of regressors x together with persistence of random noise. That is,

$$y_i = \mathrm{x}_i^T \beta + \varepsilon_i$$

where $\beta \in \mathbb{R}^p$ is a vector of coefficients, $\varepsilon = (\varepsilon_1, \ldots, \varepsilon_n) \sim \mathbb{P}_\varepsilon$ is a noise vector.

The formulation can be rewritten in matrix notation:

$$y = X\beta + \varepsilon,$$

where $y \in \mathbb{R}^n$, $X \in \ltimes \times \imath$ is a design matrix, $\varepsilon \in \mathbb{R}^n$.

Simulating data

Let's first generate some artificial data for which we know the ground truth. We are going to write a function which allows to create a dataset of arbitrarily many dimensions (i.e. a vector of coefficients can be of any length). We assume that the regressors come from multivariate normal distribution.

```
sim.linmod <- function(n, b, noise.sd, rho.x = 0) {

    # How many regressors?
    p <- length(b)

  # First, define a covariance matrix (Sigma) for the regressors.
    # Note: assume that an intercept is also included in b.
  Sigma <- diag(p - 1)
  Sigma[upper.tri(Sigma)] <- Sigma[lower.tri(Sigma)] <- rho.x

  # Draw the design matrix from a multivariate normal.
    x <- rmvnorm(n = n, sigma = Sigma)
```

*This handout is a re-adaptation of part of Jordi Llorens's materials who was teaching this brushup in the previous years. The original licence is located here.

```
  # Then generate y as a function of x and some noise.
  # A linear model of the form: y = Xb + e;
  # use matrix multiplication operator
  y <- cbind(1, x) %*% b + rnorm(n, mean = 0, sd = noise.sd)

  # put x and y in a dataframe
  data <- data.frame(y, x)

  # rename all the x variables so that they have the following format:
  # x1, x2, ...
  names(data)[2:p] <- paste0("x", 1:(p - 1))

  return(data)
}

# lets try it out
set.seed(1234)
data <- sim.linmod(200, c(5, 3, -2), 2, 0.3)
head(data)
```

```
##           y         x1          x2
## 1 -1.088649 -1.1509832  0.09103446
## 2 11.527929  0.7159032 -2.15395689
## 3  4.529436  0.5009523  0.56531759
## 4  2.503199 -0.6510388 -0.62752183
## 5  5.686707 -0.6929845 -0.96538910
## 6  5.555835 -0.6231786 -1.05924053
```

```
# output should be:
#           y         x1          x2
# 1 -1.088649 -1.1509832  0.09103446
# 2 11.527929  0.7159032 -2.15395689
# 3  4.529436  0.5009523  0.56531759
# 4  2.503199 -0.6510388 -0.62752183
# 5  5.686707 -0.6929845 -0.96538910
# 6  5.555835 -0.6231786 -1.05924053
# ...
```

Illustrate the data

We will first generate the data of lower dimension that can be easily visualized. We will use this data for the rest of the exercise.

```
set.seed(1234)
trainData <- sim.linmod(200, c(5, 3), 2)
head(trainData)
```

```
##           y         x1
## 1  2.3492564 -1.2070657
## 2  7.2258253  0.2774292
```

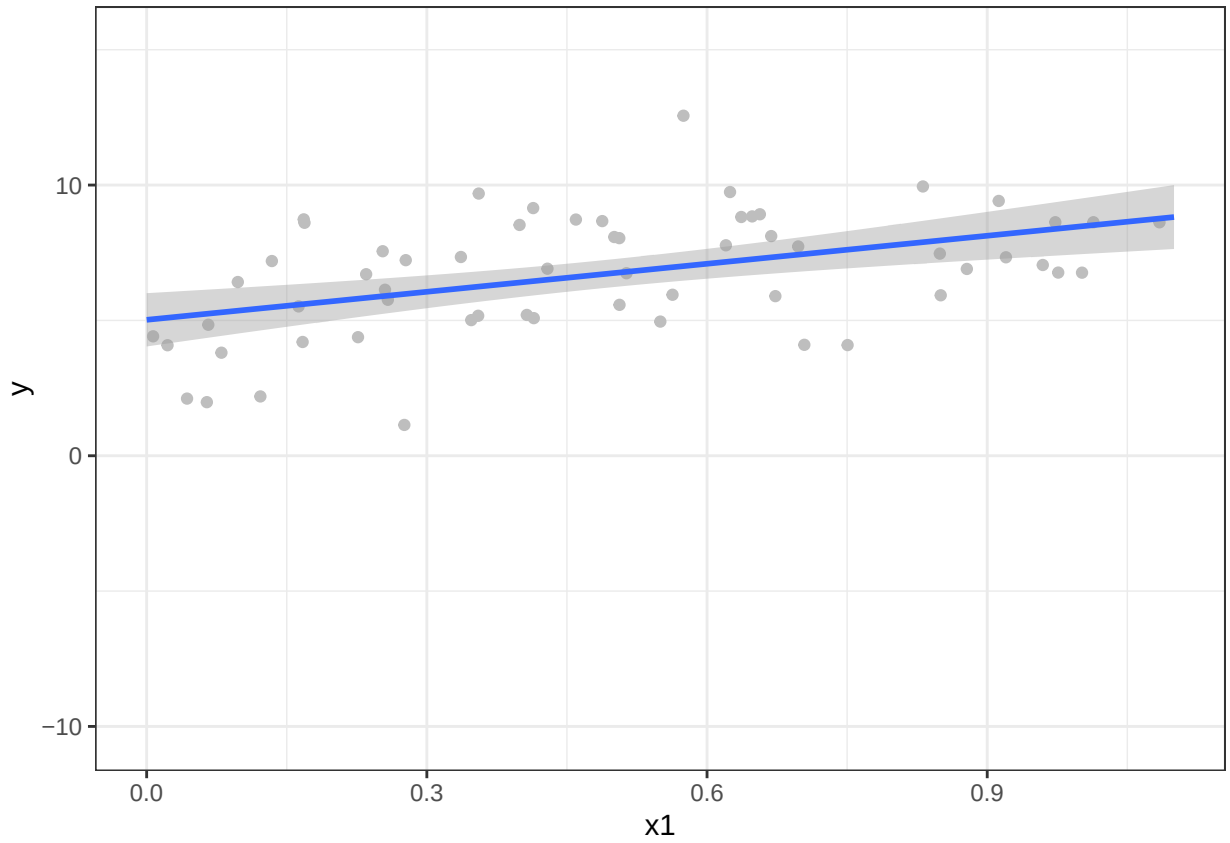
```
## 3  8.6243514  1.0844412
## 4 -0.6356261 -2.3456977
## 5  6.9107361  0.4291247
## 6  8.0390924  0.5060559
```

```
# output should be:
#            y         x1
# 1  2.3492564 -1.2070657
# 2  7.2258253  0.2774292
# 3  8.6243514  1.0844412
# 4 -0.6356261 -2.3456977
# 5  6.9107361  0.4291247
# 6  8.0390924  0.5060559
# ...
```

Use the `ggplot2` package to create a nicely formatted scatter plot of the data.

```
library(ggplot2)
# generate the figure
figure <- ggplot(trainData, aes(x1, y))+
  geom_point(shape = 19, colour = 'gray')+
  geom_smooth(method = lm, formula = y ~ x, se = TRUE, fullrange = T)+
  scale_x_continuous(limits = c(0, 1.1))+
  theme_bw()

# show the figure in the report
print(figure)
```



Fitting linear models

The classic estimator of the linear regression model is the ordinary least squares estimator, defined as the minimizer of the residual sum of squares

$$\hat{\beta} = \text{argmin}_{\beta}(y - X\beta)^T(y - X\beta).$$

The estimator has a closed form solution whenever X^TX is invertible.

$$\hat{\beta} = (X^TX)^{-1}X^Ty.$$

Although, it is worth mentioning that properties of this estimator depends on different conditions (e.g. see Gauss–Markov theorem).

You will use this estimator to compute the regression weights in your `linearmodel` function. It should produce the same coefficients as the `lm` function built-in R. But before that, use the `lm` function and fit a linear model to the data.

```
# regress y on x
lmfit <- lm(y ~ x1, data = trainData)

# use the "summary" command on results to get a more detailed overview of the
# fit, you will get additional info like standard errors
summary(lmfit)
```

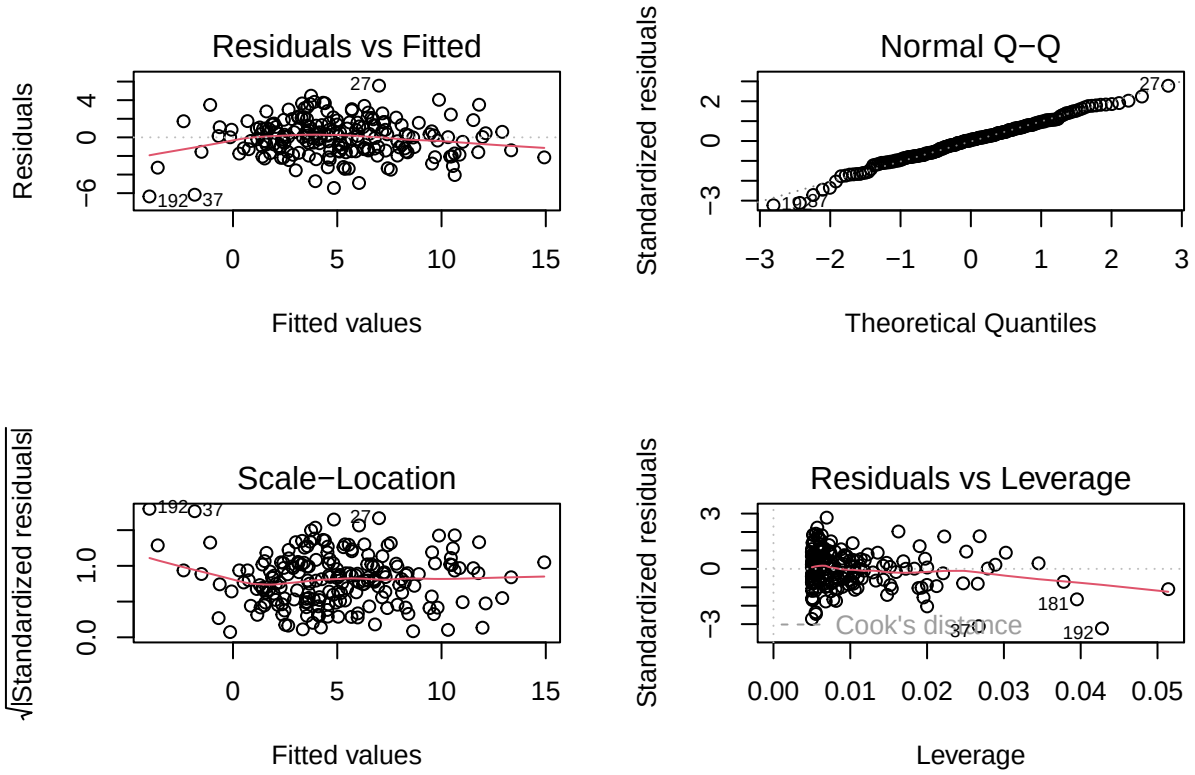
```
##
## Call:
## lm(formula = y ~ x1, data = trainData)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.3532 -1.3936  0.1024  1.3172  5.5594
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   5.1586     0.1422   36.28   <2e-16 ***
## x1            3.2092     0.1394   23.02   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.008 on 198 degrees of freedom
## Multiple R-squared:  0.7279, Adjusted R-squared:  0.7265
## F-statistic: 529.7 on 1 and 198 DF,  p-value: < 2.2e-16
```

```
# extract the coefficients with SE's, t-statistics and p-values
# note that this is a matrix so we can extract anything we like from here
print(summary(lmfit)$coef)
```

```
##             Estimate Std. Error  t value     Pr(>|t|)
## (Intercept) 5.158584  0.1421965 36.27786 2.080665e-89
## x1          3.209236  0.1394427 23.01472 7.229516e-58
```

Let's obtain diagnostic plots by using plot command on the output of the lm function (again, example of overloading the functions)

```
par(mfrow = c(2, 2))
plot(lmfit)
```




```
par(mfrow = c(1, 1))
```

Obtain predictions for the training data based on the fitted model

```
trainPredictions <- fitted(lmfit)
head(trainPredictions)
```

```
##         1         2         3         4         5         6
##  1.284825  6.048920  8.638811 -2.369313  6.535746  6.782636
```

and compute the mean square error between predictions and true values, y

```
trainMSE <- mean((trainPredictions - trainData$y)^2)
trainMSE
```

```
## [1] 3.990684
```

How do you know how good is this? Compute now a mean square error for the base model - a simple mean of the observed y values

```
baseMSE <- mean((mean(trainData$y) - trainData$y)^2)
baseMSE
```

```
## [1] 14.66632
```

so the model does a bit better obviously, however the true indicator of how well the model does is the generalization performance, predictions on the data it has not been fitted to. So, lets now create some new, test data.

```
set.seed(4321)
testData <- sim.linmod(100, c(5, 3), 2)
head(testData)
```

```
##           y         x1
## 1  5.329033 -0.4267574
## 2  5.415174 -0.2236118
## 3  5.480114  0.7176068
## 4  7.018580  0.8414457
## 5  7.421491 -0.1283573
## 6 10.314894  1.6093472
```

and verify how our model predicts on it

```
testPredictions <- predict(lmfit, testData)
head(testPredictions)
```

```
##         1         2         3         4         5         6
##  3.789019  4.440961  7.461553  7.858981  4.746655 10.323358
```

```
# compute the mean square error between predictions and true values, y
testMSE <- mean((testPredictions - testData$y)^2)
testMSE
```

```
## [1] 3.520773
```

```
# and benchmark against MSE of just the sample mean of the training set
test_baseMSE <- mean((mean(trainData$y) - testData$y)^2 )
test_baseMSE
```

```
## [1] 11.98646
```

Extra: next, fit the linear model on the trainData this time don't include the intercept in the model and include an additional variable that is a square root of absolute value of X.

```
lmfit2 <-lm(y ~ x1 + sqrt(abs(x1)) - 1, data = trainData)
lmfit2
```

```
##
## Call:
## lm(formula = y ~ x1 + sqrt(abs(x1)) - 1, data = trainData)
##
## Coefficients:
##            x1  sqrt(abs(x1))
##         3.056          5.182
```

```
summary(lmfit2)
```

```
##
## Call:
## lm(formula = y ~ x1 + sqrt(abs(x1)) - 1, data = trainData)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10.3899  -0.9946   1.0098   2.8956   6.8776
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## x1              3.0558     0.2078   14.70   <2e-16 ***
## sqrt(abs(x1))   5.1820     0.2360   21.96   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.996 on 198 degrees of freedom
## Multiple R-squared:  0.7745, Adjusted R-squared:  0.7722
## F-statistic:   340 on 2 and 198 DF,  p-value: < 2.2e-16
```

Custom function to compute coefficients of a linear model (OLS).

Now we will define our own function for fitting linear models, `linearmodel` function, that will use the closed form solution of least squares estimator to compute the weights.

$$\hat{\beta} = (X^T X)^{-1} X^T y.$$

```
linearmodel <- function(data, intercept = TRUE) {

    # we will assume that first column is the response variable
    # an all others are having a form: x1, x2, ...

    # number of features
    m <- ncol(data) - 1

    # first we need to add a vector of 1's to our x (intercept),
    # if instructed by the intercept argument
    if (intercept) {
        X <- cbind(1, as.matrix(data[, 2:(m + 1)]))
    } else {
        X <- as.matrix(data[, 2:(m + 1)])
    }
    y <- data$y

    # now implement the analytical solution
    # using the matrix operations
    # hint: check "solve" command
    bhat <- solve(t(X) %*% X) %*% t(X) %*% y

    # compute the predictions for the training data, i.e. fitted values
```

```
    yhat <- X %*% bhat

    # compute the mean square error for the training data, between y and yhat
    MSE <- mean((yhat - y)**2)

    return(list(coef = bhat, predictions = yhat, MSE = MSE))
}

# check out the function
lmmefit <- linearmodel(trainData)
lmmefit$coef
```

```
##          [,1]
## [1,] 5.158584
## [2,] 3.209236
```

```
# compare it to the output of the lm function
coefficients(lmfit)
```

```
## (Intercept)          x1
##    5.158584    3.209236
```

Illustrate the data and your model predictions

We will now create a plot where we illustrate our predictions.

```
# first create an additional data frame with x1 variable from trainData as one
# column and predictions from the model, yhat, as a second model
predData <- data.frame(x1 = trainData$x1, yhat = lmmefit$predictions)

# generate the figure
figure <-
    ggplot(trainData, aes(y = y, x = x1)) +
    geom_point(size = 1.5, color = "#992121") +

    # you will need to use geom_line, but now with
    # predData, to illustrate the linearmodel fit
    geom_line(data = predData, aes(x = x1, y = yhat),
              color = "blue") +

    theme(
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        panel.background = element_blank(),
        axis.line.x = element_blank(),
        axis.line.y = element_blank(),
        axis.ticks = element_line(lineend = 4, linetype = 1,
                                  colour = "black", size = 0.3),
        axis.text = element_text(colour = "black"),
        axis.text.x = element_text(vjust = 0.5),
```

```
        validate = TRUE
    )

# show the figure in the report
print(figure)
```