Linear models

Jordi Llorens

September 2020

Linear models

Generating Data from a linear model

Let's first generate some artificial data for which we know the ground truth. You can reuse the code from the `rng.R` exercise, but now we will make the function more general, so that we can create dataset of arbitrarily many dimensions (i.e. vector w can be of any length). Moreover, there will be another argument to the function, `dist`, that will determine whether Uniform or Normal distribution is used for generating values for each feature.

```r
rlinmod <- function(n, b, sd, dist = runif) {

    # how many features?
    m <- length(b)

  # lets draw observations for x from a distribution given by the argument
  x <- matrix(dist(n*(m-1)),ncol=m-1,nrow=n)

  # and then generate y as a function of x and some optional error
  # a linear model of the form: y = Xw
  # use matrix multiplication operator
  y <- cbind(1, x) %*% b + rnorm(n, mean = 0, sd = 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, ... xm
  names(data)[2:m] <- paste0("x",1:(m-1))

  return (data)
}

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

```
##          y        x1        x2
## 1 4.990055 0.1137034 0.6607546
## 2 7.203717 0.6222994 0.5283594
## 3 6.563864 0.6092747 0.3174938
```

```
## 4 6.735894 0.6233794 0.7678555
## 5 7.153491 0.8609154 0.5263085
## 6 6.977253 0.6403106 0.7323019
```

```
# output should be:
#          y        x1        x2
# 1 4.990055 0.1137034 0.6607546
# 2 7.203717 0.6222994 0.5283594
# 3 6.563864 0.6092747 0.3174938
# 4 6.735894 0.6233794 0.7678555
# 5 7.153491 0.8609154 0.5263085
# 6 6.977253 0.6403106 0.7323019
# ...
```

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 <- rlinmod(200, c(5, 3), 2)
head(trainData)
```

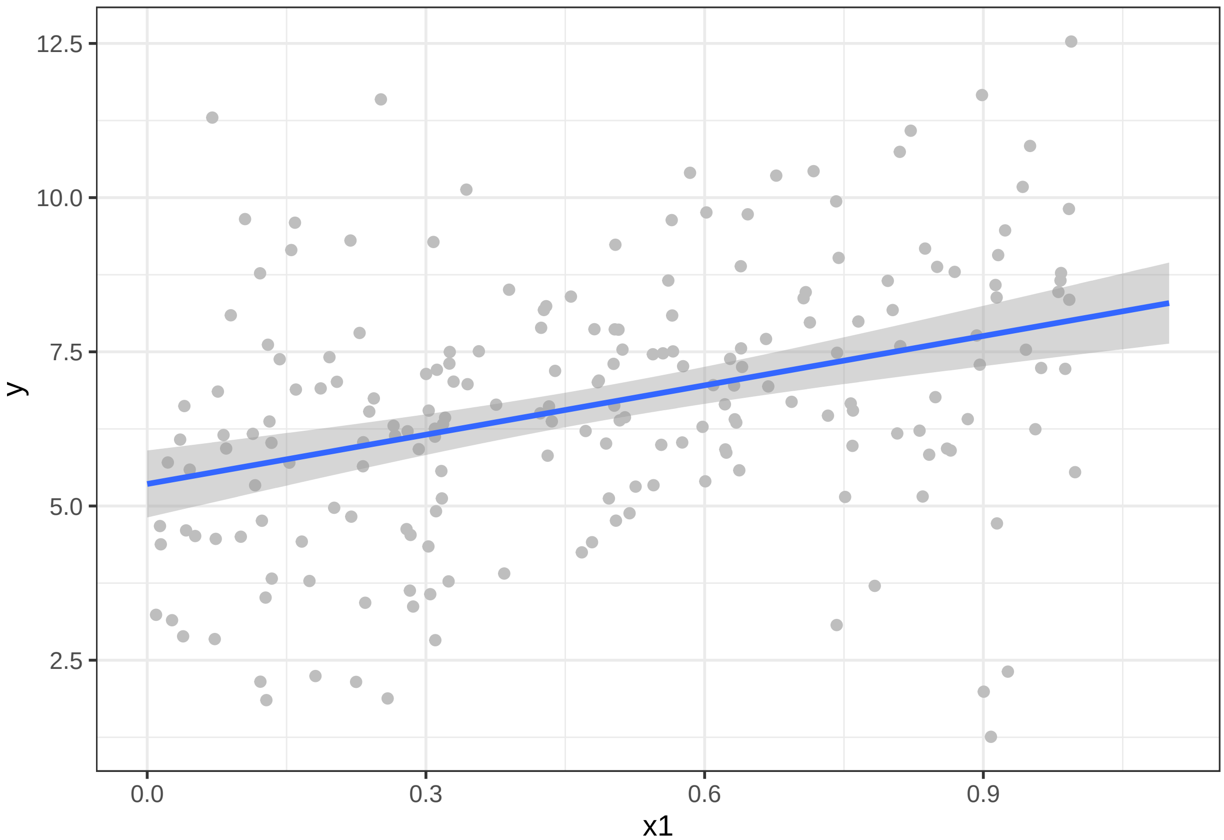
```
##          y        x1
## 1 6.170157 0.1137034
## 2 5.917461 0.6222994
## 3 6.959811 0.6092747
## 4 5.865183 0.6233794
## 5 5.930749 0.8609154
## 6 7.254910 0.6403106
```

```
# output should be:
#          y        x1
# 1 6.170157 0.1137034
# 2 5.917461 0.6222994
# 3 6.959811 0.6092747
# 4 5.865183 0.6233794
# 5 5.930749 0.8609154
# 6 7.254910 0.6403106
# ...
```

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

Linear regression is the most widely used tools in statistics. Consider the following linear regression model

$$y_i = x_i\beta + \epsilon_i, i = 1, ..., n$$

where $\beta \in R^m$. We assume usual things, like ϵ_i is a zero mean and σ_ϵ^2 variance error term is uncorrelated with x_i. The system can be equivalently expressed using matrix notation

$$y = X\beta + \epsilon.$$

The classic estimator of the linear regression model is the 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

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

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.

```r
# 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)
```

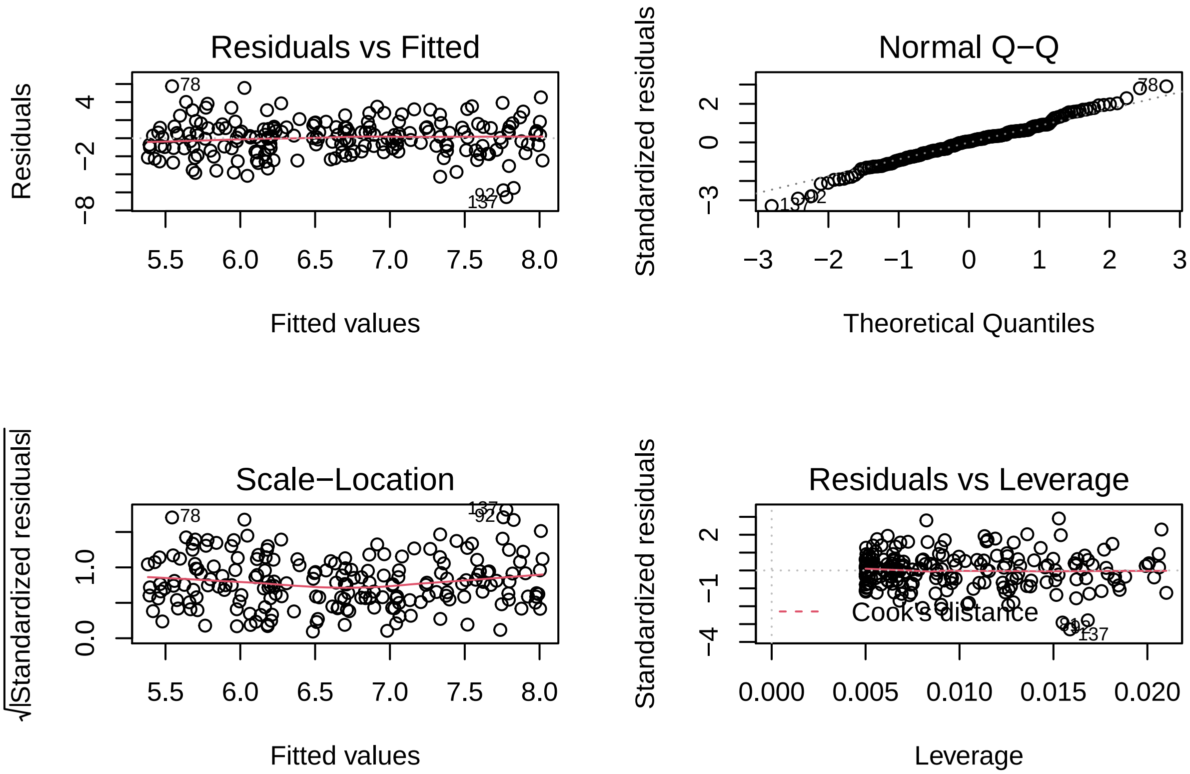
```
##
## Call:
## lm(formula = y ~ x1, data = trainData)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -6.520 -1.183  0.106  1.151  5.754
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   5.3571     0.2755  19.448  < 2e-16 ***
## x1            2.6663     0.4896   5.446 1.51e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.995 on 198 degrees of freedom
## Multiple R-squared:  0.1303, Adjusted R-squared:  0.1259
## F-statistic: 29.66 on 1 and 198 DF,  p-value: 1.515e-07
```

```r
# 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) 5.357099  0.2754508 19.44848 8.188681e-48
## x1          2.666343  0.4895750  5.44624 1.514705e-07
```

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

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




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

Obtain predictions for the training data based on the fitted model

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

```
##        1        2        3        4        5        6
## 5.660271 7.016362 6.981634 7.019242 7.652594 7.064386
```

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

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

```
## [1] 3.93848
```

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

```r
baseMSE <- mean((mean(trainData$y) - trainData$y)**2)
baseMSE
```

```
## [1] 4.528486
```

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.

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

```
##            y          x1
```

```
## 1  6.466320 0.33477802
## 2  8.146859 0.90913948
## 3  9.184255 0.41152969
## 4  9.292019 0.04384097
## 5  7.587329 0.76350011
## 6 10.688372 0.75043889
```

and verify how our model predicts on it

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

```
##        1        2        3        4        5        6
## 6.249732 7.781176 6.454378 5.473994 7.392852 7.358026
```

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

```
## [1] 3.941675
```

```r
# 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] 4.468892
```

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 X

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

```
##
## Call:
## lm(formula = y ~ x1 + sqrt(x1) - 1, data = trainData)
##
## Coefficients:
##       x1  sqrt(x1)
##   -11.03     18.11
```

```r
summary(lmfit2)
```

```
##
## Call:
## lm(formula = y ~ x1 + sqrt(x1) - 1, data = trainData)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.9836 -1.2029  0.0775  1.3730  7.2772
##
## Coefficients:
##          Estimate Std. Error t value Pr(>|t|)
## x1        -11.029      1.262  -8.739    1e-15 ***
## sqrt(x1)   18.109      1.021  17.730   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.115 on 198 degrees of freedom
```

```
## Multiple R-squared:  0.909,  Adjusted R-squared:  0.9081
## F-statistic: 989.3 on 2 and 198 DF,  p-value: < 2.2e-16
```

Defining your own function for fitting linear models

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.

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

```r
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[, paste0("x", 1:m)]))
    } else {
        X <- as.matrix(data[, paste0("x", 1:m)])
    }
    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(weights = bhat, predictions = yhat, MSE = MSE))
}

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

```
##          [,1]
## [1,] 5.357099
## [2,] 2.666343
```

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

```
## (Intercept)          x1
##    5.357099    2.666343
```

Illustrate the data and your model predictions

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

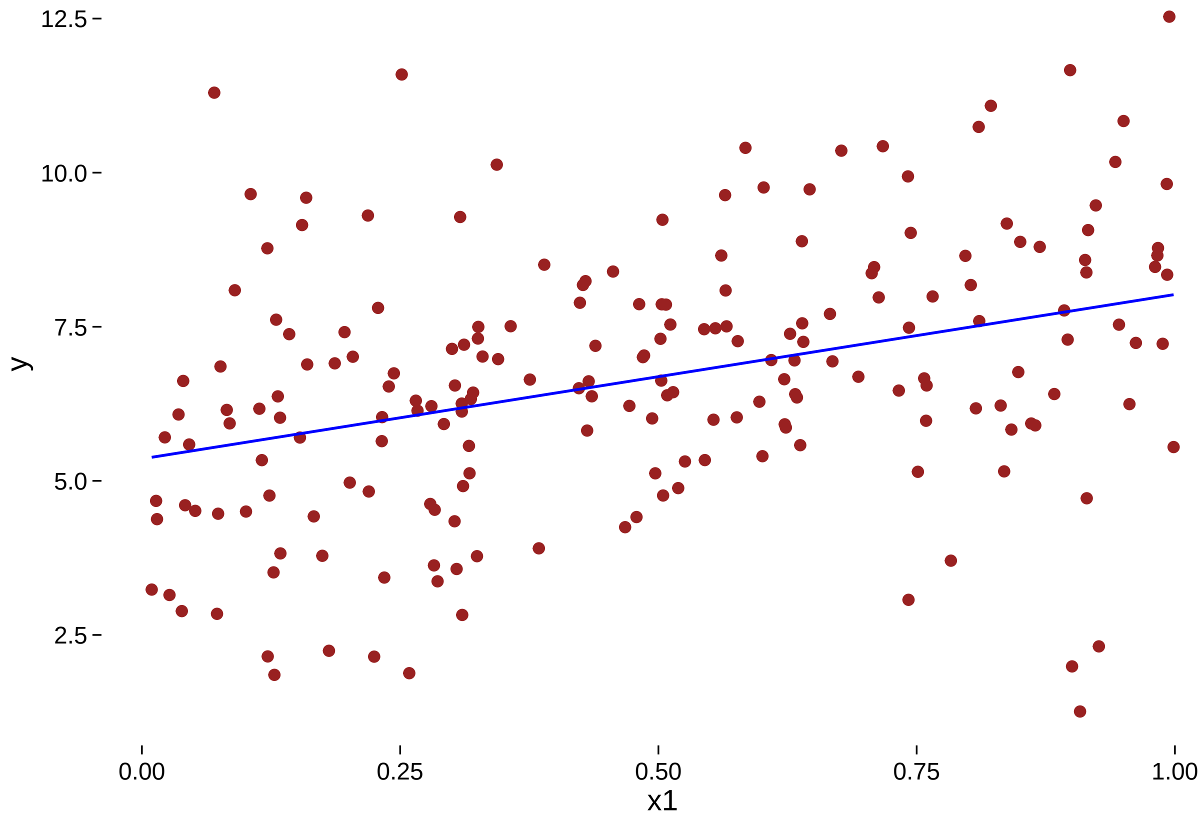
```r
# first create an additional data frame with x1 variable from trainData as one
# column and predictions from the model, yhat, as a second model
predData <- data.frame(x1 = trainData$x1, yhat = lmmefit$predictions)

# 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)
```


12.5
10.0
7.5
5.0
2.5
y
0.00
0.25
0.50
0.75
1.00
x1