Stats 102B - Homework 1

Instructions

Spring 2020

Homework 1 Requirements

You will submit three files.

The files you submit will be:

1. `102b_hw1_Prob1_First_Last.pdf` The 'scan' of your handwritten work for Problem 1.
2. `102b_hw1_output_First_Last.Rmd` Take the provided R Markdown file and make the necessary edits so that it generates the requested output.
3. `102b_hw1_output_First_Last.pdf` OR `102b_hw_01_output_First_Last.html` Your output file. This can be a PDF or an HTML file. This is the primary file that will be graded. **Make sure all requested output is visible in the output file.**

Failure to submit all files will result in an automatic 40 point penalty.

Academic Integrity

At the top of your R markdown file, be sure to include the following statement after modifying it with your name.

"By including this statement, I, Joe Bruin, declare that all of the work in this assignment is my own original work. At no time did I look at the code of other students nor did I search for code solutions online. I understand that plagiarism on any single part of this assignment will result in a 0 for the entire assignment and that I will be referred to the dean of students."

If you collaborated verbally with other students, please also include the following line to credit them.

"I did discuss ideas related to the homework with Josephine Bruin for parts 2 and 3, with John Wooden for part 2, and with Gene Block for part 5. At no point did I show another student my code, nor did I look at another student's code."

Reading:

a. A First Course in Machine Learning [FCML]: Chapter 1

Part 1: Weighted Least Squares Regression

Task 1A

Solve exercise 1.11 from page 38 of the textbook.

Hint: define matrix **A** be a diagonal matrix of the weights $\alpha_1, ...\alpha_N$ as follows:

$$\begin{bmatrix} \alpha_1 & 0 & \dots & 0 \\ 0 & \alpha_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \alpha_N \end{bmatrix}$$

Hint: With this matrix defined, the loss function can be written as:

$$\mathcal{L} = \frac{1}{N}(\mathbf{t} - \mathbf{X}\mathbf{w})^T\mathbf{A}(\mathbf{t} - \mathbf{X}\mathbf{w})$$

Do your work with pencil and paper. Write the dimensions underneath each matrix to make sure you have the order of multiplication correct. **Use a scanner** (if you have one) **or a document scanner app** like “Google Drive” or “Microsoft Office Lens” or “Genius Scan” etc. to convert your work to a PDF. Submit the PDF along with this file.

After you find your solution, typeset the gradient of the loss w.r.t. $\mathbf{w}$ in the output file.

Also typeset your solution for $\hat{\mathbf{w}}$ in the output file.

If you’re new to LaTex and typesetting, you can visit: https://en.wikibooks.org/wiki/LaTeX/Mathematics for examples to follow.

Task 1B

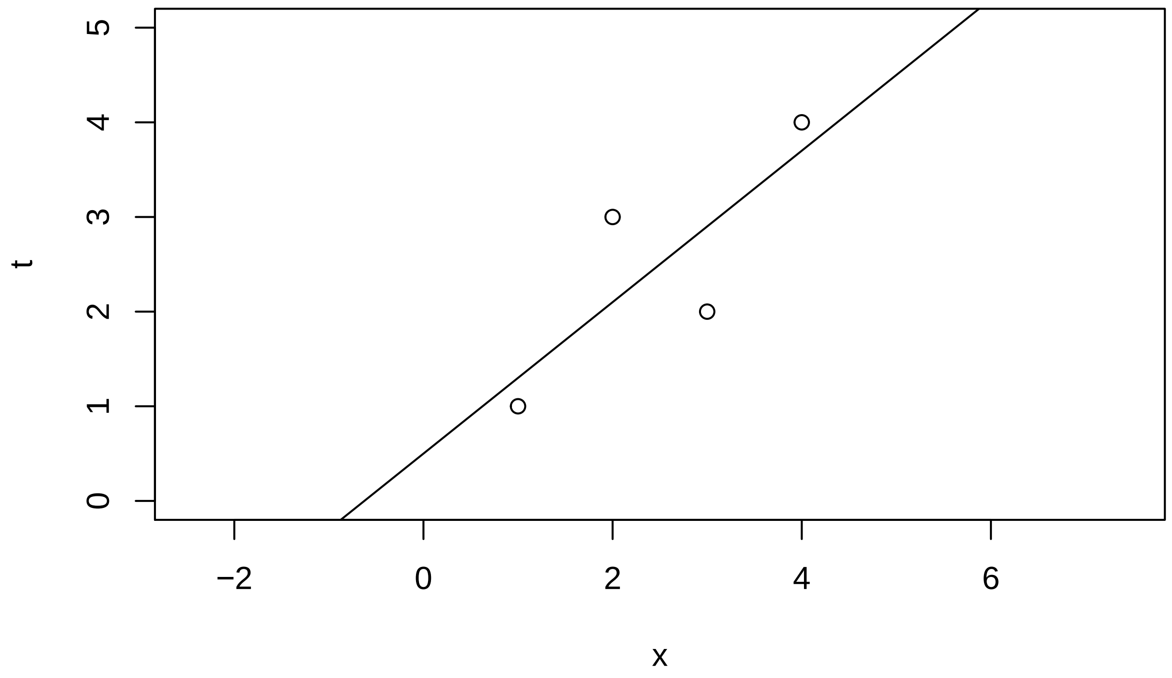
Let's see the effect of altering the values of our weights α.

We'll use the following toy data, where x is the values 1 through 4, and t is the values, 1, 3, 2, 4.

```
x <- c(1, 2, 3, 4)
t <- c(1, 3, 2, 4)
```

We begin by setting our vector of weights all equal. I have already fit a model using `lm()` with the argument `weights`.

```
a1 <- c(1, 1, 1, 1)
model1 <- lm(t ~ x, weights = a1)
plot(x, t, xlim = c(0,5), ylim = c(0,5), asp = 1)
abline(model1)
```



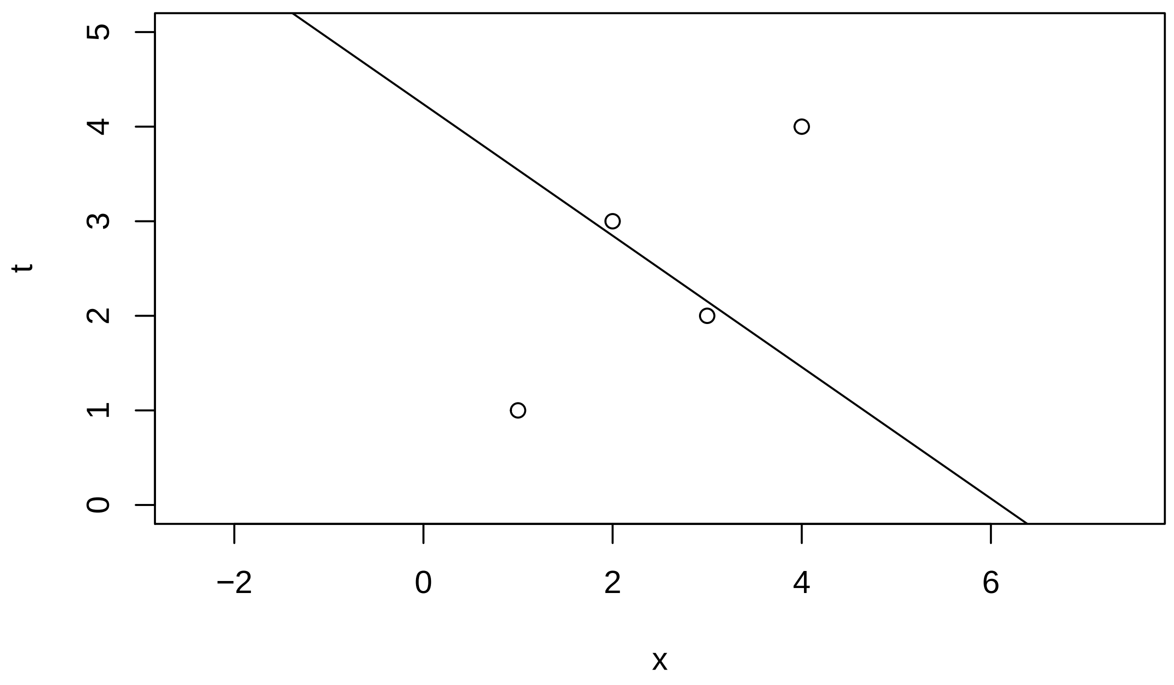
```
print(model1$coefficients)
```

```
## (Intercept)           x
##         0.5         0.8
```

In the output file, use the matrix operations from your solution to Part 1a to find and print the parameter estimates. They should equal the parameter estimates found via `lm()`

Now we alter the weights. This vector puts large weight on the two inner points (x=2, x=3), and small weight on the outer points (x=1, x=4).

```r
a2 <- c(0.1, 5, 5, 0.1)
model2 <- lm(t ~ x, weights = a2)
plot(x, t, xlim = c(0, 5), ylim = c(0, 5), asp = 1)
abline(model2)
```



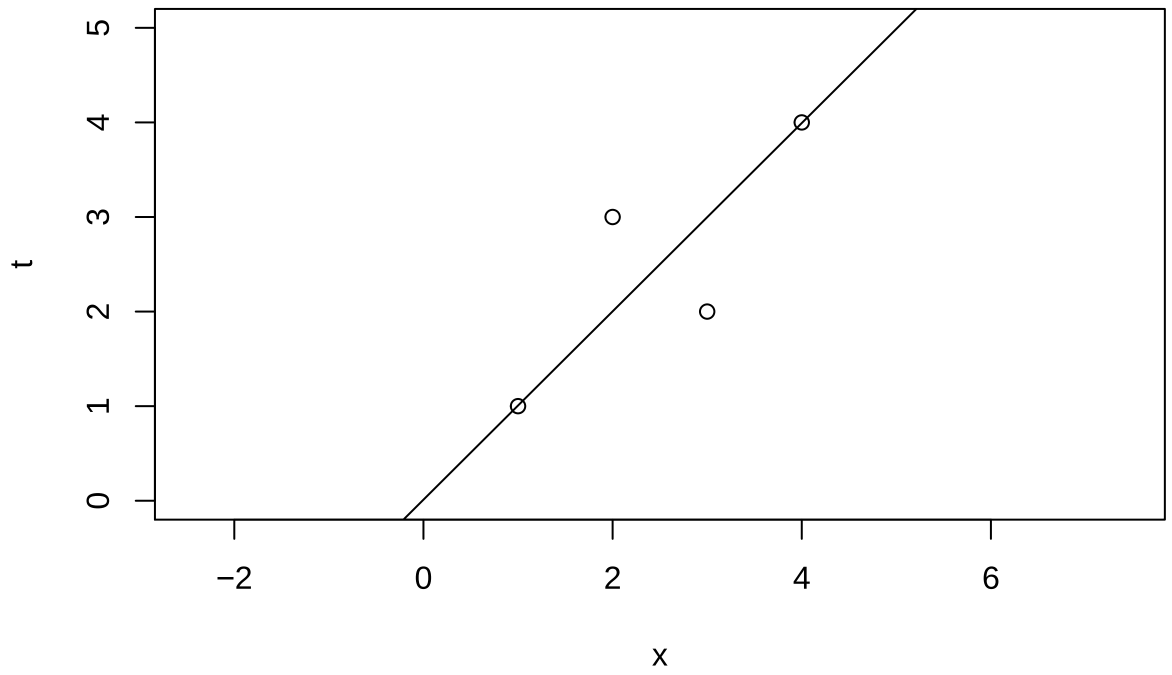
```r
print(model2$coefficients)
```

```
## (Intercept)           x
##   4.2372881  -0.6949153
```

Again, use the matrix operations to find and print the parameter estimates using the provided weights. Compare them against the estimates found via `lm()`. I have already plotted the fitted line, comment on the effect of the weights.

We alter the weights again. This time large weight are on the two outer points (x=1, x=4), and small weight on the inner points (x=2, x= 3). Again, use the matrix operations to find and print the parameter estimates using the provided weights. Compare them against the estimates found via `lm()`. Look at the fitted line and comment on the effect of the weights.

```
a3 <- c(5, 0.1, 0.1, 5)
model3 <- lm(t ~ x, weights = a3)
plot(x, t, xlim = c(0, 5), ylim = c(0, 5), asp = 1)
abline(model3)
```



```
print(model3$coefficients)
```

```
## (Intercept)           x
##  0.01108647  0.99556541
```

- Explain Weighted Least Squares regression. What effect would putting a very large weight, say 1000, on a point have on the regression line? What effect would putting a weight of 0 on a point have on the regression line?

Part 2: OLS Matrix Notation

Task 2

Review the Lecture notes.

We'll take a look at the Chirot dataset which covers the 1907 Romanian Peasant Revolt. General info on the event: https://en.wikipedia.org/wiki/1907_Romanian_Peasants%27_revolt

The data covers 32 counties in Romania, and the response variable is the intensity of the rebellion.

The orginal paper by Daniel Chirot, which provides details of the analysis and variables: https://www.jstor.org/stable/2094430

A basic data dictionary can be found with `help(Chirot)`

```r
library(carData)
data(Chirot)
chirot_mat <- as.matrix(Chirot)
```

Do the analysis with matrix operations only.

- Start by extracting the commerce column and creating our **X** matrix.
- Use `lm()` to fit the rebellion intensity to matrix X (which has columns for a constant and commercialization). Make sure you only calculate the coefficient for the intercept once.
- Using only matrix operations, calculate and show the coefficient estimates. Verify that they match the estimates from `lm()`.
- Create another matrix (call it X_ct, the ct is for commerce and tradition) with three columns: a constant, variable commerce, variable tradition. Print the head of this matrix.
- Using matrix operations, calculate and show the coefficient estimates of the model with the variables commerce and tradition.
- Create another matrix (call it X_all) with all of the x variables (plus a constant). Print the head of this matrix. Using matrix operations, calculate and show the coefficient estimates of the model with all the variables.
- Using matrix operations, calculate the fitted values for all three models. (No need to print out the fitted values.) Create plots of the fitted values vs the actual values for each model (there will be three plots). Be sure to provide a title for each plot.
- Now that you have calculated the columns of fitted values, find the Residual Sum of Squares and the R-squared values of the three models. Which model has the smallest RSS? (RSS is the sum of (actual - fitted)^2. R-sq is the correlation between actual and fitted, squared.)

Part 3: Cross-Validation

We can use cross-validation to evaluate the predictive performance of several competing models.

You will manually code leave-one-out cross-validation from scratch first and then you will use the built-in function for LOOCV in R.

You will use the dataset `ironslag` from the package `DAAG` (a companion library for the textbook Data Analysis and Graphics in R).

The description of the data is as follows: The iron content of crushed blast-furnace slag can be determined by a chemical test at a laboratory or estimated by a cheaper, quicker magnetic test. These data were collected to investigate the extent to which the results of a chemical test of iron content can be predicted from a magnetic test of iron content, and the nature of the relationship between these quantities. [Hand, D.J., Daly, F., et al. (1993) **A Handbook of Small Data Sets**]

The `ironslag` data has 53 observations, each with two values - the measurement using the chemical test and the measurement from the magnetic test.

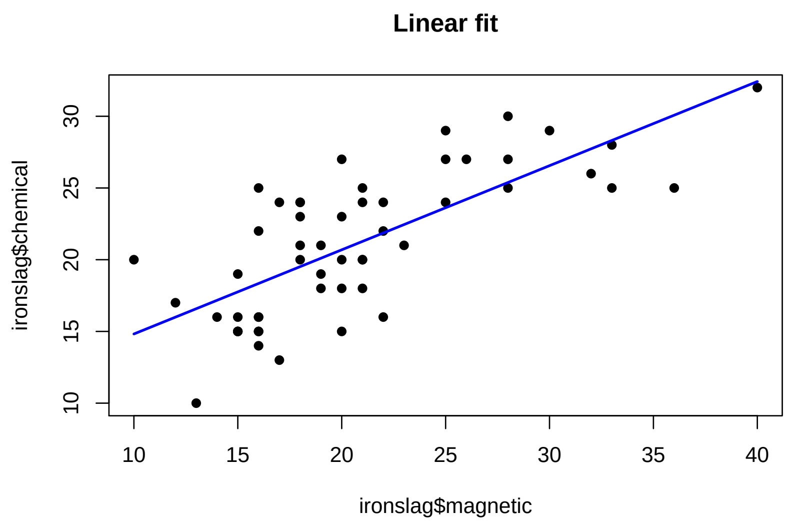
You can start by fitting a linear regression model $y_n = w_0 + w_1 x_n + \epsilon_n$. A quick look at the scatterplot seems to indicate that the data may not be linear.

```r
# install.packages("DAAG") # if necessary
library(DAAG)
```

```
## Loading required package: lattice
```

```r
x <- seq(10,40, .1) # a sequence used to plot lines

L1 <- lm(chemical ~ magnetic, data = ironslag)
plot(ironslag$magnetic, ironslag$chemical, main = "Linear fit", pch = 16)
yhat1 <- L1$coef[1] + L1$coef[2] * x
lines(x, yhat1, lwd = 2, col = "blue")
```



In addition to the linear model, you will fit a few additional models that predict the magnetic measurement (Y) from the chemical measurement (X).

Task 3A

Create a plot showing the fitted line for each of the following models:

Linear: $y_n = w_0 + w_1 x_n + \epsilon_n$.

Quadratic: $y_n = w_0 + w_1 x_n + w_2 x_n^2 + \epsilon_n$

Exponential: $\log(y_n) = w_0 + w_1 x_n + \epsilon_n$, equivalent to $y_n = \exp(w_0 + w_1 x_n + \epsilon_n)$

log-log: $\log(y_n) = w_0 + w_1 \log(x_n) + \epsilon_n$

Task 3B: Leave-one-out Cross validation

You will now code leave-one-out cross validation.

In LOOCV, we remove one data point from our data set. We fit the model to the remaining 52 data points to get the coefficient estimates. With these estimated coefficients, we make a prediction for the left-out point. We then compare that prediction to the actual value to calculate the squared error. Once we find the squared error for all 53 points, we can take the mean to get a cross-validation error estimate of that model.

To test out the four models, you will build a loop that will remove one point of data at a time. Thus, you will make a for loop that iterates over each observation. For each iteration of the loop, you will fit all four models on the remaining 52 data points to get coefficient estimates, and then using those estimates you will make a prediction for the point

Follow the guide and hints provided in the Output file

Compare the sizes of the cross validation error to help you decide which model does the best job of predicting the test cases.

Task 3C: Cross-validation with R

Now that you have written your cross-validation script from scratch, you will use R's native functions to perform cross-validation. Library(boot) has the function `cv.glm()` which can be used to estimate cross-validation error.

To make use of `cv.glm()` on the linear models, you must first use `glm()` to fit a generalized linear model to our data. If you do not change the attribute "family" in the function `glm()`, it will fit a linear model and is pretty much equivalent to using `lm()`. You have to use `glm()` however, because the cross-validation function `cv.glm()` does not work on the output of `lm()` but only on the output of `glm()`.

Your LOOCV estimates from `cv.glm()` should match your estimates when you coded your algorithm from scratch in Task 3B.

Based on your Cross Validation scores, which model seems to be the best?

Task 3D: Revisit Chirot

Following what you did in Task 3C, use `glm()` to fit the following three models to the Chirot data.

- model1: intensity ~ commerce
- model2: intensity ~ commerce + tradition
- model3: intensity ~ all other variables

Use `cv.glm()` to calculate the LOOCV scores to compare models. Which model would you select?

Part 4: Using R's `optim()` function

The `optim()` function in R is quite versatile and can be used to minimize a wide range of functions using a variety of methods. The default method, the Nelder-Mead simplex algorithm, is a numeric method that works quite well for multidimensional functions. Other methods, like BFGS which uses a modified version of gradient descent, are included and are also quite powerful.

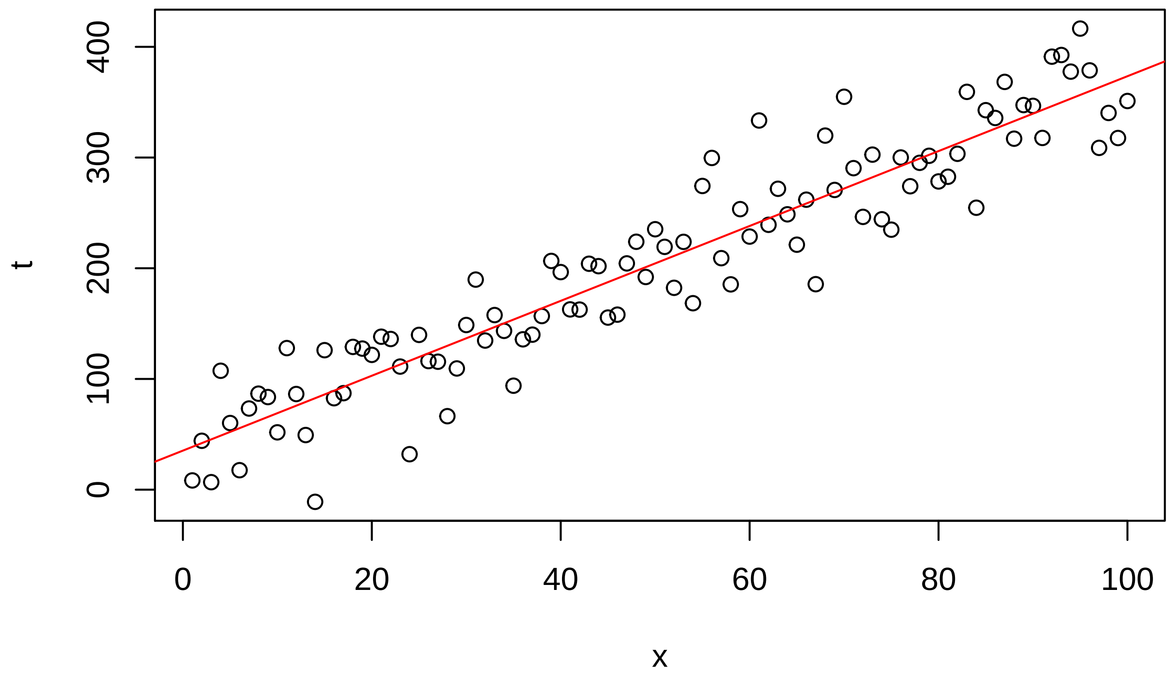
You will do an exercises to use `optim()` in to minimize the loss function in linear regression.

In the output file, I provide code to create a synthetic data set. The generated code has a true slope of 3.4 and a true intercept of 30. Random noise has been added and you will fit an OLS model to the noisy data.

```r
x <- 1:100
set.seed(1)
e <- rnorm(100, 0, 40)
t <- 30 + 3.4 * x + e    # the true slope is 3.4, the true intercept is 30
coefs <- coef(lm(t ~ x)) # the coefficients from using lm
print(coefs)  #if you use set.seed(1), this should be 35.266629, 3.381958
```

```
## (Intercept)           x
##   35.266629    3.381958
```

```r
plot(x, t)
abline(coef = coefs, col = "red")
```



Task 4: Use `optim()` to minimize the sum of squared residuals in linear regression

For the first task, we define the loss function as the sum of squared residuals:

$$\mathcal{L} = F(w_0, w_1) = \sum_{n=1}^{N} (t_n - \hat{t}_n)^2$$

Where $\hat{t}$ is the predicted value from the linear model: $t_i = w_0 + w_1 x_i$

We can also use matrix notation:

$$\mathcal{L} = F(\mathbf{w}) = (\mathbf{t} - \mathbf{X}\mathbf{w})^T(\mathbf{t} - \mathbf{X}\mathbf{w})$$

Write your loss function as a function of the vector `par` which will have a length of 2. The vector par will contain the current estimates of the intercept and slope. Treat the data (`x` and `t`) as fixed quantities that are available in the global environment. The function will return a single value: the sum of squared residuals.

If you've never used or seen `optim()` before, I've uploaded lecture notes from 102A to CCLE that covers basic usage of `optim()`.

Look at the results and the values of the parameters that minimize the loss function, they should match the coefficients estimated by `lm()`.