Problem Set #1

Kevin McAlister

January 12th, 2022

This is the first problem set for QTM 385 - Intro to Statistical Learning. It includes both analytical derivations and computational exercises. While you may find these tasks challenging, I expect that a student with the appropriate prerequisite experience will be able to complete them

Please use the intro to RMarkdown posted in the Intro module and my .Rmd file as a guide for writing up your answers. You can use any language you want, but I think that a number of the computational problems are easier in R. Please post any questions about the content of this problem set or RMarkdown questions to the corresponding discussion board.

Your final deliverable should be a .zip archive that includes a .Rmd/.ipynb file and either a rendered HTML file or a PDF. Students should complete this assignment **on their own**. This assignment is worth 10% of your final grade.

This assignment is due by January 26th, 2022 at 6:00 PM EST.

Problem 1 (40 pts.)

Linear regression is a fundamental tool for statistics and machine learning. At its core, linear regression is a simple task: given a set of P predictors, $\{\mathbf{x}_i\}_{i=1}^N = \boldsymbol{X}$, with each $\mathbf{x}_i$ a $P+1$-vector of predictors with a 1 as the first element (to account for an intercept) and outcomes, $\{y_i\}_{i=1}^N = \boldsymbol{y}$, find the $P+1$-vector $\hat{\boldsymbol{\beta}}$ that minimizes the residual sum of squares:

$$\hat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta}^*}{\text{argmin}} \left[(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}^*)' (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}^*) \right]$$

This can also be expressed as a summation over errors:

$$\hat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta}^*}{\text{argmin}} \left[\sum_{i=1}^{N} \left(y_i - \boldsymbol{\beta}^{*'} \boldsymbol{x}_i \right)^2 \right]$$

In the case of a single predictor, **simple linear regression** can be easily expressed without matrix notation:

$$\{\hat{\alpha}, \hat{\beta}\} = \underset{\alpha^*, \beta^*}{\text{argmin}} \left[\sum_{i=1}^{N} (y_i - \alpha^* - \beta^* x_i)^2 \right]$$

Part 1 (8 pts.)

Linear regression was widely used when computers were dinky little calculators with only 512 MBs of RAM (I remember upgrading my first PC to a 1 GB stick of RAM and wondered how anyone would ever need more!) because it admits a slew of **analytical solutions** to the above minimization problems.

Let's start with simple linear regression. Find the values of $\hat{\alpha}$ and $\hat{\beta}$ that **minimize** the residual sum of squares. Try to reduce these as much as possible (using the identities for covariance and variance *hint, hint*).

Some hints:

1. I think it's easier to start with α^*.
2. Remember that the empirical covariance estimator is $\frac{\sum_{i=1}^{N}(x_i-\bar{X})(y_i-\bar{Y})}{N}$ and that the empirical variance estimator is $\frac{\sum_{i=1}^{N}(x_i-\bar{X})^2}{N}$. You can do the same thing with $N-1$ (to ensure that the estimator is always unbiased, not just asymptotically).
3. Remember that the regression coefficient is defined as $\frac{\text{Cov}(X,Y)}{\text{Var}(X)}$. You should probably get the same thing here for one of the parts.
4. To make everything easier, try to get sums to averages (divide by N when needed)

Part 2 (7 pts.)

A common theme we'll see this semester is the notion of optimization - finding the values of parameters that maximize/minimize objective functions of interest. Optimization can be tricky when functions are not strictly convex/concave - most computational methods of optimization can only guarantee that they locate one of potentially many **local optima** when we really want to find the **global optimum**. Argue that the sum of squared errors function for simple linear regression is **strictly convex** in α^* and β^* and that there exists a unique and finite global minimum. You can assume that there is nothing funky here (e.g. variance in both X and Y, $N \geq 2$, etc.).

Part 3 (10 pts.)

Coding and statistical learning go hand-in-hand. Your previous classes have largely been pen-and-paper focused - even your regression class likely only covered methods that could, in theory, be done with some pen, paper, and a calculator (though inverting a large matrix by hand would be considered cruel and unusual torture by most). We're going to quickly move out of the realm of methods that are analytically solvable, so understanding how **algorithms** work from the ground up will help you to understand why something works when we can't always prove it via mathematics. In many cases, too, we'll find that computational approaches with no pen-and-paper analogue (cross-validation, bootstrapping, black-box optimization, etc.) will provide superior answers to the analytical methods.[1]

This said, let's assume that you didn't know how to find the optimum derived in part 1. We could always use **numerical optimization** methods to find the minimum. Additionally, since we can leverage our knowledge that the function is strictly convex in α and β, we should always land on the same answer! So, this method will be equivalent.

Write a function called `sse_minimizer` that uses a built-in optimization routine to find the values of α and β that minimize the sum of squared errors for simple linear regression. `sse_minimizer` should take in two arguments: `y` - a N-vector of outcome values and `x` - a N-vector of predictor values. It should return a list (or equivalent holder) with five elements elements: 1) `alpha_est` - the value of the intercept given by the optimization routine, 2) `beta_est` - the value of the regression coefficient given by the optimization routine, 3) `alpha_true` - the true value of the intercept computed using your answer from part 1, 4) `beta_true` - the value of the regression coefficient computed using your answer from part 1, and 5) `mse` - the mean squared error (the sum of squared errors divided by the number of observations) between the predicted value of the outcome and the true value of the outcome.

[1] Have you ever really thought about how a computer inverts a matrix? It requires a lot of mathematics that aren't needed when thinking about doing it by hand. There's tricks and decompositions that make it work almost instantaneously! This is just one example of a situation where the computational approach is far superior to the analytical one

To test your function, generate some simulated data:

1. Generate 1000 uniform random values between -3 and 3 as `x`
2. Choose some values for the intercept and slope, `a` and `b`. Using `x`, generate `y` as `a + b*x`.
3. Add some random noise to `y` - `y <- y + rnorm(100,0,1)` for example.
4. Plug `x` and `y` into your function and see if it returns the correct parameter values.

A full-credit answer will demonstrate that the function works!

Some tips to help you get going (in R, if that's your choice; Python is similar):

1. Write a function called `sum_squared_errors` that takes three arguments: 1) `coefficients` - a 2-vector of coefficients (α and β, respectively, 2) `y` - a N-vector of outcome values, and 3) `x` - a N-vector of predictor values. The function should return the sum of squared errors **given** the input values of α and β.
2. Call `sum_squared_errors` from `sse_minimizer` within the optimization routine. I recommend using base R's `optim` to find the minimum. `optim` can be a bit confusing on the first go, so be sure to **read the documentation** and look for examples online if you're confused. I'm also happy to help in office hours, but I think this is an important "do-it-yourself" technique.
3. Lists are nifty because we can easily name elements. If we have `lst <- list()`, then we can assign `lst$alpha <- foo` and `lst$beta <- bar`.

Part 4 (7 pts.)

Now, let's move on to multiple linear regression. Using the same logic as above, we can show that the sum of squared errors is strictly convex in $\hat{\boldsymbol{\beta}}$, so there exists a unique minimum (assuming $\boldsymbol{X}$ is of full rank). Working with matrix derivatives, show that the $\hat{\boldsymbol{\beta}}$ that minimizes the sum of squared errors is:

$$\underset{\boldsymbol{\beta}^*}{\operatorname{argmin}} \left[(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}^*)' (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}^*) \right] = \left(\boldsymbol{X}'\boldsymbol{X} \right)^{-1} \boldsymbol{X}'\boldsymbol{y}$$

Part 5 (8 pts.)

Frequently, we seek to perform **inference** on the values of $\boldsymbol{\beta}$ - we want to determine if the noise associated with the OLS estimator is small enough to say that each β_j is statistically different from zero. First, we want to show that the OLS estimator is **unbiased** for the true value of $\boldsymbol{\beta}$ so that we can claim that our inference is meaningful. Then, we need to derive the **standard error** of the estimator to perform inference.

Show that $\hat{\boldsymbol{\beta}} = \left(\boldsymbol{X}'\boldsymbol{X} \right)^{-1} \boldsymbol{X}'\boldsymbol{y}$ is unbiased for $\boldsymbol{\beta}$ - $E[\hat{\boldsymbol{\beta}}] = \boldsymbol{\beta}$. Then, derive the **variance-covariance matrix** for $\hat{\boldsymbol{\beta}}$ - $\text{Cov}[\hat{\boldsymbol{\beta}}]$. The square root of the diagonal of the variance-covariance matrix then provides the **standard errors**.

Some helpful identities:

1. For multiple linear regression, we assume that $\boldsymbol{X}$ is constant while $E[\boldsymbol{y}] = X\boldsymbol{\beta}$ and $\text{Cov}[\boldsymbol{y}] = \sigma^2 \mathcal{I}_N$ where σ^2 is the constant error variance (e.g. a scalar) and $\mathcal{I}_N$ is the $N \times N$ identity matrix.
2. Suppose we want to know $\text{Cov}[\boldsymbol{A}\boldsymbol{y}]$ where $\boldsymbol{A}$ is a matrix of constants and $\boldsymbol{y}$ is a random vector. Then $\text{Cov}[\boldsymbol{A}\boldsymbol{y}] = \boldsymbol{A} \times \text{Cov}[\boldsymbol{y}] \times \boldsymbol{A}'$

Problem 2 (40 pts.)

Over the semester, we're going to leverage probability **distributions** and common summaries of probability distributions - expectations, variance, covariance, etc. The goal of this problem is to review what you've already learned in previous classes and (perhaps) introduce the idea of **simulation** to understand properties of distributions.

Suppose that we have a random variable x such that:

$$f(x) \sim \exp[-\lambda x] \text{ if } x \geq 0 \text{ else } f(x) = 0$$

where $\sim$ means **distributed as** and λ is an arbitrary parameter value greater than 0. Furthermore, we know that x can only take values on the positive real line so the **density** of any x less than zero is exactly 0.

Part 1 (7 pts.)

As is, the probability distribution above is not a proper probability density function - it doesn't integrate to 1! Given the above info, show that the value of Z that **normalizes** the density function to a proper probability density function:

$$f(x) = Z \times \exp[-\lambda x] \text{ if } x \geq 0 \text{ else } f(x) = 0$$

is λ.

Part 2 (8 pts.)

Given the proper PDF above, derive the **expected value** and **variance** of x.

Part 3 (7 pts.)

Show that the corresponding cumulative density function for the above PDF is $1 - \exp[-\lambda x]$ and that the median of this distribution is $\frac{\ln(2)}{\lambda}$.

Part 4 (8 pts.)

The most common scenario where we'll see probability distributions is when trying to estimate the parameters that dictate a data generating process. For example, we may observe N observations that are assumed to independent and identically distributed draws from the above probability distribution. Since we've observed these draws and made the i.i.d. assumption, we can derive a **likelihood** function for the data:

$$f(X; \lambda) = \prod_{i=1}^{N} \lambda \exp[-\lambda x_i]$$

where x_i is one of the N observations ($x_i \in \{x_1, x_2, ..., x_{N-1}, x_N\}$). Find the value of λ that maximizes the above likelihood function (e.g. the maximum likelihood estimator of λ) given N observations.

Some hints:

1. Start by simplifying the product as much as possible. Remember that a constant, a, times itself M times is a^M. Also, don't forget the rules of exponents! Finally, anything not subscripted is a constant - use that to your advantage!

2. Take a natural log of the simplified likelihood function - this is called the **log-likelihood**. Remember that the log of a product is equal to the sum of the logs. This is a good step because it gets rid of the annoying exponentials. We can also do this because a log is a **one-to-one** transformation, so it preserves the maximum.
3. Use calculus to find the value of λ that maximizes the expression. If this solution is unique, use a second derivative test to show that you have found a maximum or logically argue that the original or log likelihood is strictly concave in λ.

Part 5 (10 pts.)

Maximum likelihood estimators are an important part of statistics and machine learning as they are commonly used to minimize certain types of **loss functions** - find the parameter that maximizes the likelihood/minimizes the loss with respect to the data generating process. MLEs are desirable in statistics because they have a number of desirable properties like asymptotic unbiasedness, consistency, and efficiency and follow the same generic recipe regardless of data type. Asymptotic unbiasedness means that the estimator converges to the correct answer almost surely as $N \to \infty$. Asymptotic consistency implies that as $N \to \infty$, the standard deviation of the sampling distribution (e.g. the standard error) goes to zero. Asymptotic efficiency implies that the estimator achieves the lowest possible variance on the path to zero - otherwise known as the Cramer-Rao lower bound.

For this last part, I want you to graphically show the asymptotic consistency and efficiency properties using a method called **parametric bootstrapping**. We're going to discuss bootstrapping as a method of uncertainty calculation and this part is a short introduction to the method. Bootstrapping techniques leverage the fact that probability can be interpreted as the long run proportion of occurrence. We can replicate long-run frequencies by using computational methods to take random draws from a distribution.

The distribution you've been working with here is called the **exponential distribution** and is a key distribution in the study of random counting processes and Bayesian statistics. This makes the process of taking **random draws** from the distribution easy - just use `rexp()` in R and the equivalent functions in other languages! The main gist of bootstrapping is that we can replicate the process of repeated sampling by taking a large number of random draws from the distribution of interest to estimate the quantity of interest.

Write a function called `bootstrap_se` that takes in three arguments: 1) `n` - the sample size, 2) `b` - the number of bootstrap replicates, and 3) `lambda` - a value for λ. Your function should do the following:

```
For n, b, and lambda
  For i in 1:b
    Draw n values from Exp(lambda)
    Compute MLE for lambda
    Store MLE
  Compute standard deviation of MLEs
  Return standard deviation
```

In words, this function should take N samples from an exponential distribution parameterized by λ and compute the MLE implied by your random draws B times. The standard deviation of these B values converges almost surely to the standard error of the sampling distribution for the MLE which is used to perform inference about the value of the parameter.

Using your function, evaluate the standard deviation of the MLE setting `n` equal to 10,20,30,. . . ,280,290,300, `b` equal to 250, and `lambda` equal to 1/3. Then, plot the standard deviations against the values of `n` as a line graph. Label this as the "Bootstrap" line. Does this line approach 0? What is approximate rate at which it converges (think in terms of the sample size)?

To demonstrate that the MLE is maximally efficient (the most efficient estimator possible), compute the Cramer-Rao lower bound for the exponential MLE given a value of N. The Cramer-Rao lower bound, the

minimum variance that can be achieved with an asymptotically unbiased estimator of λ, is shown below:

$$\text{CRLB}[f(X;\lambda)] = \sqrt{-\left[\frac{\partial^2 \log f(X;\lambda)}{\partial\lambda\partial\lambda}\right]^{-1}} = \frac{\lambda}{\sqrt{N}}$$

Plot this value against the corresponding values of `n` on the same plot as your bootstrapped line and label this as the "CRLB" line. Does the bootstrapped MLE standard error reach the Cramer-Rao lower bound? Generate the same figure for at least 2 other values of λ. Does the same relationship hold?

Problem 3

For the last part of this problem set, I want you to think about **classification** models. At their core, classification is a task that seeks to uncover a set of rules that determine whether an observation has or does not have a specific trait - Did the member of Congress cast a "Yes" vote for the bill? Will the customer buy a $2100 phone? Does the patient have cancer? These are all questions that can addressed using classification models.

We're going to talk about a variety of approaches for building classification models. However, all methods share a similar core strategy: given a set of predictors, what is a **rule set** that best predicts the observations true class? For this problem, I want you to try to build a common-sense rules based model that predicts whether or not a patient has heart disease.

Part 1 (15 pts.)

The data set `heartTrain.csv` contains information about 200 patients that were tested for heart disease. `AHD` tells whether a patient was diagnosed with heart disease (`AHD = 1`) or not (`AHD = 0`). There are also five predictors included - 1) `Age` (in years), 2) `Sex` (0 is female, 1 is male), 3) `RestBP` (the patient's resting blood pressure in mm/Hg), 4) `Chol` (the patient's serum cholesterol in mg/dl), and 5) `Slope` (a three category measure of heart responsiveness to exercise (a.k.a. the ST segment) - 1 means that the heart increased activity during exercise, 2 means that the heart activity remained constant, while 3 means that heart activity decreased during exercise).

Using this data set, write a function called `heartDisease_predict` that takes in values for each of the five predictors and returns a prediction for whether or not the patient has a heart disease. Then, use your predictive model to assess the **accuracy** of the predictions by comparing them to the real labels. What proportion are correct? What proportion are incorrect?

Your predictive model should only use if/else style rules (for example, if `Sex = 0` and `Rest BP < 140` then `AHD = 0`). You can determine these rules by examining summary statistics, plots, running regression models, etc.

You'll receive 10 points for the programming of your function and explanation of your approach. You'll receive 2 more points for this problem if your model gets at least 60% of your predictions are correct, 3 more points pts. for at least 75%, and 5 more points for at least 85%. If you come up with an approach that gets more than 95% correct, I'll add a bonus point to this assignment. You can use any combination of predictors, transformations, and approaches to determine the rules set.

Part 2 (5 pts.)

There is a second data set, `heartTest.csv`, that includes more observations from the original data set. Using your version of `heartDisease_predict`, create predictions for this second data set. What is the predictive accuracy for the test data set? Is it better or worse than the accuracy for the training set? Provide some intuition for this result.