BIOS 731 Advanced Statistical Computing
Fall 2022
Homework 1

Due 9/9/2022 Friday at 11:59pm

Instruction:

- Please submit both write-ups and programs in two separate files.
- All submissions need to be in electronic format.
- The write-up is preferred be in pdf format (written in Word or LaTeX, or scanned hand-written document). You can take a picture of hand-written document and submit JPG if you don't have a scanner, but make sure the picture is clear and readable. Name the file **BIOS731_NAME_hw1.EXT**. Replace NAME by your name, and EXT by proper extension name (pdf or jpg).
- The programs need to be written in a high-level language (no compilation required), and R is recommended. The codes for all problems need to be saved in a **single** file named **BIOS731_NAME_hw1.EXT**. Replace NAME by your name, and EXT by proper extension name, e.g., R, sas, m, py, etc. Provide adequate comments in the codes to clearly mark the section for different questions. The codes should generate all results and figures in the homework. Please make sure the codes are "self-contained", e.g., does not depend on platform, can be run at any other machine in any subdirectory, and does not require user input.
- Total is 50 points, with 10 points bonus for the last question. Partial credit will be given.

Problem 1: Permutation test and bootstrap in linear regression. (30 points)

Consider a multiple linear regression model $y_i = b_0 + b_1 x_{1i} + b_2 x_{2i} + \epsilon_i$. Assume $b_0 = 10$, $b_1 = 1$, $b_2 = 0.5$.

1. (15 pts) Assume $\epsilon_i \sim N(0, 1)$, use permutation test to test null hypothesis $H_0 : b_2 = 0$. Report result p-value, and then compare with the one from using R (from the `lm` function). You can use following codes to simulate data (y_i, x_{1i} and x_{2i}).

```
n=100
b0=10; b1=1; b2=0.5
x1 = rnorm(n); x2=rnorm(n); eps=rnorm(n)
y = b0+b1*x1+b2*x2+eps
```

2. (15 pts) Use non-parametric bootstrap to estimate the 95% confidence interval (CI) for $\hat{b}_2$. Generate data using different residual distributions: (1) $\epsilon_i \sim N(0,1)$; (2) $\epsilon_i \sim t(5)$. Compare the estimated CIs to theoretical values assuming normally distributed residuals (using R `confint` function). Comment on the results.

Problem 2: Poisson regression (20 + 10 bonus points)

Write your own function for Poisson generalized linear model (GLM) with canonical link (log) using different algorithms: Iteratively reweighted least squares (IRLS), gradient descent (GD), and stochastic gradient descent (SGD). You can write three separate functions named `poisreg_IRLS`, `poisreg_GD`, and `poisreg_SGD`. Or if you prefer, you can write a single function with a `method` parameter to switch the methods. There are 10 points each for IRLS and GD, and 10 bonus points for SGD.

The function(s) should take a response y (a vector of integers) and a covariate matrix X. Other (optional) parameters of the functions could be

- the maximum number of iterations allowed
- tolerance parameter to check convergence
- Batch size and number of epochs (for SGD)

The function should return the estimated coefficient, the estimated variance/covariance matrix of the estimates, and number of iterations. Compare your results with these functions from the `glm` function in R, and make some comments.

You can use the following codes to simulate data and run `glm`:

```
n = 100 ## number of observations
p = 3 ## number of covariates
## generate X, the covariates
X = cbind(1, matrix(rnorm(n*p), ncol=p))
```

```
beta = c(1, .5, 1, 2)
mu = exp(X %*% beta)
## generate Y, the outcome
Y = rpois(n, mu)

### use R's glm function to fit
fit = glm(Y~X-1, family=poisson)
coef(fit) ## estimated coefficients
vcov(fit) ## estimated variance/covariance matrix of the estimates
```

Tips for SGD implementation:

We didn't have in-depth discussion for the implementation of SGD in the class, but there are many such discussions online, for example, `https://www.ocf.berkeley.edu/~janastas/stochastic-gradient-descent-in-r.html` and `https://towardsdatascience.com/implementing-sgd-from-scratch-d425db18a72c`.

Here are a few tips in SGD implementation:

- A good start point could be a rough estimate derived from response y and covariates X.
- SGD doesn't guarantee the increase of likelihood at every step, so we don't need to check that.
- People usually don't check the convergence from SGD, instead, they run a fixed number of iterations. There are concepts of "batch size" and "epoch". Briefly speaking, batch size means number of samples used in a single mini-batch for one iteration, and epoch means the entire dataset has been run/looped exactly one time. For example for a dataset of 2000 samples, if the batch size is 500, it will take 4 iterations to complete 1 epoch. Thus, the total number of iterations equals to total sample size $\times$ epoch number/batch size.
- The choice of batch size and number of epoch is usually arbitrary. Traditional SGD uses batch size 1 (sampling one data point each time), while the *mini-batch* SGD can use batch size 10-50. Choices for number of epoch depends on the data, but anything between 10-100 will be enough for this question.

- The step size is referred to as "learning rate" in SGD. To determine the learning rate can be tricky. Usually a large learning rate will cause the algorithm to diverge, and a small one will cause a slow convergence. There is no general prescriptions for choosing an appropriate learning rate. One method is to use diminishing step size, which means the step size decreases with increase of iteration number. Another method is to determine the learning rate on a small fraction of training data before running SGD on full data set and update the learning rate after each epoch. You may check some discussion from following online link: `https://www.cs.ubc.ca/~fwood/CS340/lectures/L24.pdf`.