

# STAT 243 Problem Set 8

Treves Li

2024-11-27

## Collaboration Statement

I did not collaborate with anyone.

## Question 1

Consider probit regression, which is an alternative to logistic regression for binary outcomes. The probit model is  $Y_i \sim \text{Ber}(p_i)$  for  $p_i = P(Y_i = 1) = \Phi(X_i^\top \beta)$  where  $\Phi$  is the standard normal CDF, and  $\text{Ber}$  is the Bernoulli distribution. We can rewrite this model with latent variables, one latent variable,  $z_i$ , for each observation:

$$\begin{aligned} y_i &= I(z_i > 0) \\ z_i &\sim \mathcal{N}(X_i^\top \beta, 1) \end{aligned}$$

## Question 1a

Design an EM algorithm to estimate  $\beta$ , taking the complete data to be  $Y, Z$ . You'll need to make use of the mean and variance of truncated normal distributions (see hint below). Be careful that you carefully distinguish  $\beta$  from the current value at iteration  $t$ ,  $\beta_t$ , in writing out the expected log-likelihood and computing the expectation and that your maximization be with respect to  $\beta$  (not  $\beta_t$ ). Also be careful that your calculations respect the fact that for each  $z_i$  you know that it is either bigger or smaller than 0 based on its  $y_i$ . You should be able to analytically maximize the expected log likelihood. A couple hints:

- i. From the Johnson and Kotz ‘bibles’ on distributions, the mean and variance of the truncated normal distribution,  $f(w) \propto \mathcal{N}(w; \mu, \sigma^2)I(w > \tau)$ , are:

$$\begin{aligned}\mathbb{E}(W \mid W > \tau) &= \mu + \sigma\rho(\tau^*) \\ \text{Var}(W \mid W > \tau) &= \sigma^2 (1 + \tau^*\rho(\tau^*) - \rho(\tau^*)^2) \\ \rho(\tau^*) &= \frac{\phi(\tau^*)}{1 - \Phi(\tau^*)} \\ \tau^* &= \frac{\tau - \mu}{\sigma}\end{aligned}$$

where  $\phi(\cdot)$  is the standard normal density and  $\Phi(\cdot)$  is the standard normal CDF. Or see the Wikipedia page on the truncated normal distribution for more general formulae.

- ii. You should recognize that your expected log-likelihood can be expressed as a regression of some new quantities (which you might denote as  $m_i, i = 1, \dots, n$ , where the  $m_i$  are functions of  $\beta_t$  and  $y_i$ ) on  $X$ .

---

Text

**Question 1b**

Propose how to get reasonable starting values for  $\beta$ .

---

Text

### Question 1c

Write a Python function to estimate the parameters. Make use of the initialization from part (b). You may use existing regression functions for the update steps. You'll need to include criteria for deciding when to stop the optimization.

---

Text

### Question 1d

Try out your function using data simulated from the model. Take  $n = 100$  and the parameters such that  $\hat{\beta}_1/\text{se}(\hat{\beta}_1) \approx 2$  and  $\beta_2 = \beta_3 = 0$ . In other words, I want you to choose  $\beta_1$  such that the signal to noise ratio in the relationship between  $x_1$  and  $y$  is moderately large. You can do this via trial and error simply by simulating data for a given  $\beta_1$  and fitting a logistic regression to get the estimate and standard error. Then adjust  $\beta_1$  as needed.

---

Text

## Question 2

A different approach to this problem just directly maximizes the log-likelihood of the observed data under the original probit model (i.e., without the  $\mathbf{z}$ s).

### Question 2a

Write an objective function that calculates the negative log-likelihood of the observed data using JAX or PyTorch syntax (for use in part (d)).

---

Text

## Question 2b

Estimate the parameters for your test cases using `scipy.optimize.minimize()` with the BFGS option. Compare how many iterations EM and BFGS take. Note that this provide a nice test of your EM derivation and code, since you should get the same results from the two optimization approaches. Calculate the estimated standard errors based on the inverse of the Hessian. Note that the `hess_inv` returned by `minimize` is probably NOT a good estimate of the Hessian as it seems to just be the approximation built up during the course of the BFGS iterations and not a good numerical derivative estimate at the optimum. Try using `numdifftools` and compare to what is seen in `hess_inv`. If you get warnings about loss of precision, you may need to tell JAX or PyTorch to use 64-bit rather than 32-bit floating point numbers.

---

Text

### Question 2c

As part of this, try a variety of starting values and see if you can find ones that cause the optimization **not** to converge using BFGS. Also try them with Nelder-Mead.

---

Text



## Question 2d

Now use JAX or PyTorch automatic differentiation (AD) functionality to create a gradient function. Set up your objective and gradient functions to use just-in-time compilation (you can use `jit()` or `@jit` for JAX and `torch.compile` or `@torch.compile` for PyTorch). Use these functions to find the parameters using BFGS. Check that you get the same results as in (b) and compare the number of iterations and timing to using BFGS without providing the gradient function (and thereby relying on `scipy` using numerical differentiation). Finally, use JAX or PyTorch functionality to create a Hessian function and use it to calculate the Hessian at the optimum. Compare the inverse of the Hessian and the estimated standard errors to what you got in part (b).

---

Text