Lab session 9: Probit, logit, know it!

Adam Theising[*]

April 10th, 2018

Contents

Getting started...

- As always, questions/comments?...
- Code and bare-bones solution sketches for assignment 4 are coming...
- Assignment 5 is short - but please don't leave it for the last minute (I feel like I've said this before... :))
- In response to community sentiment from last week, this week's lab consists of two coded examples - we'll go through them step-by-step. Takeaways from this lab should be:
  - Understand how to estimate both homo- and heteroskedastic probit models
  - Gain some initial exposure to MATLAB's basic built-in optimizer[1]
  - Get a simple flavor for how to approach classification problems using logit estimation

1 Brief comment on LPM vs. probit vs. logit

Before we get into the estimation of these models, let's quickly address when their use is most appropriate. As you surely understand at this point, the two main attractions of the probit/logit models are that they (1) bound the dependent variable's probability distribution in the $[0,1]$ space and (2) allow for non-constant marginal effects. Yet, it's fairly common to see LPM estimates of 0-1 dependent variables in well-published academic work. So what gives?

It's probably not a good use of lab time to dive deeply into this, but let me summarize in a sentence: we care more about marginal effects than parameter estimates, and it turns out that the LPM does a decent job of providing marginal effects in many cases. In your work, I'd recommend just estimating both the LPM and logit/probit models, then comparing results - it's

[*]theising@wisc.edu

[1]Make sure you have installed the Global Optimization Toolbox - all UW students have access to it. More info here.

low cost and will satisfy everyone. If you want to read more about this, I'd recommend the short blog posts by Dave Giles here and here, and by JS Pischke here.

When it comes to choosing between probit and logit in a binary choice context, it seems like the rule of thumb is to use whatever is most common in your academic field. Else, if you really want to be proper about it, just let the data speak - compare AICs or other measures of model fit. There are sometimes specific reasons for preferring probit or logit in multinomial cases, but we'll get later this week.

2 Probit model for female labor force participation

We'll start today by estimating a simple model of labor force participation as in Mroz (ECMA, 1987) - this example comes straight out of Greene's 5th edition (Example 21.4), though it turns up in several of the major econometrics textbooks at some point. Brian's lectures notes have a bit of motivation for this model using the notions of household production and reservation wages - feel free to peruse on your own time. For our purposes today, we want simply want to estimate the probability of female labor force participation (LFP) as a function of the covariates $X = (age, age^2, income, education, kids)$. The first 4 covariates are continuous, while *kids* is a dummy indicating the presence of children in the household. Our sample[2] consists of individual-level data for 753 women.

Our goal here is going to be a replication of this table from Greene:

TABLE 21.3 Estimated Coefficients

		Estimate (Std.Er)	*Marg. Effect**	*Estimate (St.Er.)*	*Marg. Effect**
Constant	β_1	−4.157(1.402)	—	−6.030(2.498)	—
Age	β_2	0.185(0.0660)	−0.0079(0.0027)	0.264(0.118)	−0.0088(0.00251)
Age^2	β_3	−0.0024(0.00077)	—	−0.0036(0.0014)	—
Income	β_4	0.0458(0.0421)	0.0180(0.0165)	0.424(0.222)	0.0552(0.0240)
Education	β_5	0.0982(0.0230)	0.0385(0.0090)	0.140(0.0519)	0.0289(0.00869)
Kids	β_6	−0.449(0.131)	−0.171(0.0480)	−0.879(0.303)	−0.167(0.0779)
Kids	γ_1	0.000	—	−0.141(0.324)	—
Income	γ_2	0.000	—	0.313(0.123)	—
Log L		−490.8478		−487.6356	
Correct Preds.		0s: 106, 1s: 357		0s: 115, 1s: 358	

*Marginal effect and estimated standard error include both mean (β) and variance (γ) effects.

The left-hand estimate columns are from a homoskedastic probit model, while the right two columns are from a heteroskedastic probit estimation. Let's start with the homoskedastic case.

2.1 *Homoskedastic probit model*

The model we are estimating here is given by:

$$\text{Prob}[LFP = 1|X] = \Phi(X'\beta)$$

where $\Phi(\cdot)$ denotes the normal CDF, as required by the probit specification. Then (skipping some algebra we've covered) the log-likelihood function to maximize is given by:

$$\ell(\beta) = \sum_{i=1}^{N} \left(LFP_i \ln(\Phi(X_i'\beta)) + (1 - LFP_i)\ln(1 - \Phi(X_i'\beta)) \right)$$

Let's take it to MATLAB. We'll run this first with Brian's pre-programmed probit function. This function (*probit_bwg*) makes use of analytical gradients and hessians specific to the probit

[2] From 1975 - when female LFP was much lower than today.

model, and runs a Newton-Raphson algorithm. Feel free to use it on your problem set. To begin, we set our probit log-likelihood function:

```
function [tot_llf] = probit_llf(bo)
  global rhsvar depvar;
  cdf_prob = normcdf(rhsvar*bo);
  llf = depvar.*log(cdf_prob) + (1-depvar).*log(1-cdf_prob);
  tot_llf=sum(llf);
end
```

Next, we run the (*probit_bwg*) code. We see that the algorithm converges to the optimal parameters in four iterations, yielding parameter values and standard errors identical to those in Greene's table[3]. Our maximized log-likelihood is also equivalent to that in the table. We next want to replicate the results for marginal effects.

```
% Marginal effect of age (take into account quadratic term)
age_marg=marg_age(b_prob);
grad_marg_age=Grad(b_prob,'marg_age',1);
se_age_marg=sqrt(grad_marg_age*c_prob*grad_marg_age');

% Marginal effect of education
edu_marg=marg_edu(b_prob);
grad_marg_edu=Grad(b_prob,'marg_edu',1);
se_edu_marg=sqrt(grad_marg_edu*c_prob*grad_marg_edu');

% Marginal effect of income
inc_marg=marg_inc(b_prob);
grad_marg_inc=Grad(b_prob,'marg_inc',1);
se_inc_marg=sqrt(grad_marg_inc*c_prob*grad_marg_inc');

% Discrete effect of having kids (NOT continuous)
kids_disc = disc_kids(b_prob);
grad_disc_kids = Grad(b_prob,'disc_kids',1);
se_kids_disc = sqrt(grad_disc_kids * c_prob * grad_disc_kids');

% Marginal effects table:
disp('********* MARGINAL EFFECTS *********');
fprintf('Age:       %8.8f (%8.7f)\n', age_marg, se_age_marg);
fprintf('Education: %8.8f (%8.7f)\n', edu_marg, se_edu_marg);
fprintf('Income:    %8.8f (%8.7f)\n', inc_marg, se_inc_marg);
fprintf('Kids*:     %8.8f (%8.7f)\n', kids_disc, se_kids_disc);
fprintf('\n*Note: Kids is a discrete effect.')
```

For the education and income marginals, calculation is fairly straightforward - we take the marginal at the mean of the data and calculation is straightforward: $\frac{\partial LFP}{\partial x} = \beta_x \phi(\bar{X}'\beta)$m where $\phi(\cdot)$ denotes the standard normal pdf. For age, note that the additional quadratic term requires some more massaging - the marginal for age can be given by: $\frac{\partial LFP}{\partial age} = (\beta_{age} + 2a\bar{g}e\beta_{age^2})\phi(\bar{X}'\beta)$. Finally, for *kids* - a dummy variable - we must rely on a discrete effect. We calculate this as

$$\begin{aligned}\Delta_{kids} &= \text{Prob}(LFP = 1 | kids = 1, \bar{X}_{-inc}) - \text{Prob}(LFP = 1 | kids = 0, \bar{X}_{-inc}) \\ &= \Phi(\sum_{j=1}^{5} \bar{x}_j \beta_j + \beta_{kids}) - \Phi(\sum_{j=1}^{5} \bar{x}_j \beta_j)\end{aligned}$$

In each, case we see that our calculations yield the identical marginals to those calculated by Greene. To estimate the standard errors, we make use of delta method approach, numerically calculating gradients as we have in previous contexts.

[3]Note that while there is some kind of scaling difference between our *income* parameter and Greene's, the model's estimates are equivalent

2.2 *A taste of MATLAB's built-in optimizers*

We've spent a good portion of this class constructing and applying different optimization algorithms - first in solving NLS models and more recently solving for ML estimators. While coding your own algorithms can be steep learning curve, hopefully it's provided you an understanding of optimization techniques and a peek inside the black boxes that are pre-canned programs.

Lest you think that MATLAB is only good for programming your own optimization algorithms, I wanted to make sure you're all aware of some built-in optimizers that exist in MATLAB's global optimization toolbox. They are super useful not only in econometric contexts, but also for broader optimization problems. Let's re-attack the homoskedastic probit model above using 2 built-in optimizers, and then MATLAB's pre-canned probit routine.

First, let's try to solve the ML problem using fminunc. This optimizer attempts to find the global solution for unconstrained minimization problems. If you don't provide analytic gradients, this function uses BFGS (quasi-Newton methods) - see here for details.

```
80 %% Alternative estimation methods using MATLAB optimizers...
81 % Since these fncts are MINIMIZERS - adapt the LLF as below!!!
82 % Built in "unconstrained" optimizer... NOTE: this gets stuck @ local max
83 [x, fval] = fminunc(@probit_llf_fmin, b_ini)
```

Using *fminunc* is straightforward - at a minimum, enter the function to be minimized[4] and initial values, and off we go. On the LHS, we've asked the function to return the optimal parameters (x) and the LLF value ($fval$). Looking at our results, we see that unfortunately the combination of starting values and the algorithm's methods leads us to a local rather than global minimum. In my experience, this is a fairly common result when using *fminunc* - usually the work around when using this kind of function is just to try several starting points.

Instead, we turn to an alternative function, fminsearch. This function is described as most suitable for optimizing "nondifferentiable problems or problems with discontinuities, particularly if no discontinuity occurs near the solution". It runs a Nelder-Mead simplex algorithm which is probably beyond the scope of this course; that said, you should know it is a derivative-free algorithm, which is what makes it especially appropriate for nondifferentiable problems.

```
85 % Built in "search" optimizer... DINGDING!: converges to global opt.
86 % options = optimset('PlotFcns',@optimplotfval); %turn on for iteration graph
87 [x, fval] = fminsearch(@probit_llf_fmin,b_ini)
```

Again, *fminsearch* is very user-friendly. Enter the function to minimize, initial values, and tell the function which outputs you would like to have returned. Note: as in Row 86 of our code, there's the possibility to pre-set an option for graphing out the LL value over iterations - this can be useful to track progress in particularly nasty problems. We run the optimization function- and VOILA! we converge at the global maximum.

Finally, let's quickly touch on MATLAB's family of glmfit functions. These are the MATLAB equivalent of a Stata regression command. You simply enter the appropriate matrices, prescribe a likelihood function ('binomial','link','probit'), and the function will return a bevy of parameters and regression statistics.

```
89 % MATLAB's built in generalized linear model function(s)
90 [b,dev,stats] = glmfit(rhsvar,depvar,'binomial','link','probit','constant','off');
91 stats.beta % Call estimated parameters out.
```

We run the function, and receive the *stats* data structure as a return. If we type "stats.beta" to the command line, we are returned the parameter estimates, "stats.se" returns the standard errors, and so on. Looking at our estimation results, we find that the *glmfit* returns identical results to those from Brian's function.

[4] Note this algorithm finds a minimum, so we need to multiply the return from our LLF function by -1.

2.3 *Heteroskedastic probit model*

Let's turn to a probit model that accounts for heteroskedasticity in the error term. Unlike in the the standard linear model, ML estimates in binary choice models are inconsistent in the presence of heteroskedasticity[5]. Parameter biases are increasing in the degree of correlation between explanatory variables and error term. To account for this, the standard approach is to incorporate multiplicative heteroskedasticity à la Harvey (1976). In the general case, imagine the latent regression $y^* = X'\beta + \varepsilon$ where $\mathbf{E}(\varepsilon) = 0$ and $\text{Var}(\varepsilon) = [e^{Z'\gamma}]^2$. Here, X is our standard vector of covariates, while Z is a vector of covariates that affect the degree of heteroskedasticity. Then:

$$\begin{aligned}
\text{Prob}(y_i = 1) &= \text{Prob}(\varepsilon_i < X_i'\beta) \\
&= \text{Prob}\left(\frac{\varepsilon_i}{exp(Z_i'\gamma)} < \frac{X_i'\beta}{exp(Z_i'\gamma)}\right) \\
&= \Phi\left(\frac{X_i'\beta}{exp(Z_i'\gamma)}\right)
\end{aligned}$$

where the last equality holds because $\frac{\varepsilon_i}{exp(Z_i'\gamma)} \sim \mathcal{N}(0,1)$ due to our multiplicative heteroskedasticity assumption. Thus, for our labor participation model, we are left with the following log-likelihood function:

$$\ell(\beta,\gamma) = \sum_{i=1}^{N}\left(LFP_i \ln(\Phi(\frac{X_i'\beta}{exp(Z_i'\gamma)})) + (1 - LFP_i)\ln(1 - \Phi(\frac{X_i'\beta}{exp(Z_i'\gamma)}))\right)$$

Here, our X vector remains the same as in the homoskedastic case. The Z vector includes *income* and *kids*. Let's think about how we undertake estimation. First, we need to add the γ parameters and covariates to their respective *rhsvar* and initial value matrices:

```
% Bring in covariates for heteroskedastic correction
hetvar = rhsvar(:,5:6);
hetnames = parnames(5:6,:);
names = char(parnames,hetnames);

% Starting values for heteroskedastic model
b_ini = b_prob; % start with homosk ests
g_ini = [0.01, 0.01]'; % start with small values
p_ini = vertcat(b_ini, g_ini);
```

Then we code up the LLF function:

```
function [llf] = probit_hetero_llf(bo)
    global rhsvar hetvar depvar ;
    [~,numbetas]=size(rhsvar);
    alp=bo(numbetas+1:end);  %*** Alpha Coefficients ***
    betas=bo(1:numbetas);       %*** Beta Coefficients ***
%   sigmasq=(exp(hetvar*alp)).^2;
    sigma=exp(hetvar*alp);      %*** Std. Error of Error term ***
    z = (rhsvar*betas)./sigma;   %*** Standardized Value ***
    cdf_prob = normcdf(z);
    llf = depvar.*log(cdf_prob) + (1-depvar).*log(1-cdf_prob);
end
```

We can estimate our ML parameters as usual, using the LLF we just derived. We run the standard BHHH algorithm and after 20-something iterations, find that we've replicated the results in the Greene table. As you can see, the parameter estimates are markedly different when we account for heteroskedasticity. But to better understand the bias from resulting from misspecification, it's more important to compare marginal effects. To do so, let's first derive the general

[5] See Ch. 15.7.1 in Wooldridge (2009) for an excellent summary of the consequences of neglected heteroskedasticity in probit/logit models.

marginal effect formula for a given covariate w_k in the heteroskedastic case:

$$\frac{\partial \text{Prob}(y_i = 1|X,Z)}{\partial w_k} = \phi\left(\frac{X'\beta}{exp(Z'\gamma)}\right)\frac{\beta_k - (X'\beta)\gamma_k}{exp(Z'\gamma)}$$

Three things to note here. First, if the covariate is in both the X and Z vectors, then the marginal is given by the above formula. Second, if the covariate is only in the X vector, then $(X'\beta)\gamma_k = 0$ and that term drops out of the formula. Third, if the covariate is only the heteroskedasticity correction vector, Z, then $\beta_k = 0$, and that term drops out of the formula.

Let's take it to our data and estimated parameters - note that we'll take the marginal at the mean of the data. Each marginal effect calculation is slightly different here, so by going through them individually, we should get a pretty good feel for what's going on. For education, we have $\frac{\partial LFP}{\partial edu} = \phi(\bar{X}'\beta e^{-\bar{Z}'\gamma})\frac{\beta_{edu}}{exp(\bar{Z}'\gamma)}$. For age, we have: $\frac{\partial LFP}{\partial age} = \phi(\bar{X}'\beta e^{-\bar{Z}'\gamma})\frac{\beta_{age}+2a\bar{g}e\beta_{age^2}}{exp(\bar{Z}'\gamma)}$. These first two covariates were not present in the Z vector, and their marginals reflect that. For income, however, we have: $\frac{\partial LFP}{\partial inc} = \phi(\bar{X}'\beta e^{-\bar{Z}'\gamma})\frac{\beta_{inc}-(X'\beta)\gamma_{inc}}{exp(\bar{Z}'\gamma)}$. And finally, the discrete effect for the *kids* dummy, present in both the X and Z vectors, is:

$$\Delta_{kids} = \Phi\left(\frac{\sum_{j=1}^{5}\bar{x}_j\beta_j + \beta_{kids}}{exp(\gamma_{inc}\bar{inc} + \gamma_{kids})}\right) - \Phi\left(\frac{\sum_{j=1}^{5}\bar{x}_j\beta_j}{exp(\gamma_{inc}\bar{inc})}\right)$$

We code these messy functions up separately in MATLAB, and once more replicate Greene's results[6].

```
125 % Education marginal effect
126 hmarg_edu = het_marg_edu(b_phet);
127 grad_hmarg_edu=Grad(b_phet,'het_marg_edu',1);
128 se_edu_marg= sqrt(grad_hmarg_edu*c_phet*grad_hmarg_edu');
129
130 % Age marginal effect
131 hmarg_age = het_marg_age(b_phet);
132 grad_hmarg_age=Grad(b_phet,'het_marg_age',1);
133 se_age_marg= sqrt(grad_hmarg_age*c_phet*grad_hmarg_age');
134
135 % Income marginal effect
136 hmarg_inc = het_marg_inc(b_phet);
137 grad_hmarg_inc=Grad(b_phet,'het_marg_inc',1);
138 se_inc_marg = sqrt(grad_hmarg_inc*c_phet*grad_hmarg_inc');
139
140 % Kids discrete* effect
141 %*Note: Greene (mistakenly?) takes the marginal effect in his table...
142 hdisc_kids = het_disc_kids(b_phet);
143 grad_hdisc_kids=Grad(b_phet,'het_disc_kids',1);
144 se_kids_disc = sqrt(grad_hdisc_kids*c_phet*grad_hdisc_kids');
```

Again, we've used the delta method to back out standard errors for these marginal effects. Pay careful attention to the change in income's marginal effect between the homo- and heteroskedastic cases: it's certainly notable on a first glance! Homoskedastic estimation here substantially attenuates the true marginal effect of income. Moving forward, maybe start to think about how you'd test for significance here...

So there you have it - hopefully a fairly comprehensive example of how to about estimating and interpreting a probit model, with and without heteroskedasticity. Let's move on a to a quick example of a logit model.

[6] You will note that the discrete effect we find for *kids* is different - best I can tell, this is because Greene used the marginal effect instead of the discrete effect. I think this is an oversight on his part - discrete effect is appropriate here.

3 Using a logit model for classification

Logit models are often used in econometrics and beyond to tackle a set of problems known as *classification*. The general idea is to *classify* a data observation (y_i, the LHS variable) based on its characteristics (X_i, the RHS variable(s)) - some illustrative binary examples include (1) classifying email as spam or not, (2) classifying a tumor as cancerous or not, or (3) classifying a production output as high quality or not. This is an interesting set of problems - and can obviously be extended to more than two classes. As such, a LOT of attention has been directed towards these problems in statistics, econometrics, and data science[7]. The logit model is the ground floor of classification approaches; but to give you some flavor of applicability, know that (much) more sophisticated classification models are helping operate self-driving cars.

3.1 *An example: classification of water quality*

In this (albeit contrived) example, using a logit model, we're going to try and classify whether a water source has "clean" water or not based on its characteristics. Suppose the following scenario: there 11000 streams in the state of WI. To ensure compliance with the Clean Water Act, the state is required to do some annual testing of these streams. Problematically, it is costly to undertake this testing. The state can only afford to do a full panel of water quality tests at 1000 of the streams - these tests yield a 0-1 measure of clean water (with 1 meaning the water is sufficiently clean, and 0 meaning dirty). Fortunately, however, previous evidence has suggested a strong correlation between the state's binary cleanliness measure, and two covariates: pH and TSS (total suspended solids). Importantly, the state has collected data on these two covariates for *all* 11000 streams.

Our task then is to estimate a logit model on the 1000 observations[8] for which we have full LHS (y_i - quality measure) and RHS (X_i - pH and TSS) data. Then, we'll use the parameters from this model and the X covariates to try and classify the other 10000 observations[9]. After making predictive classifications, we can see how well our model performed out-of-sample.

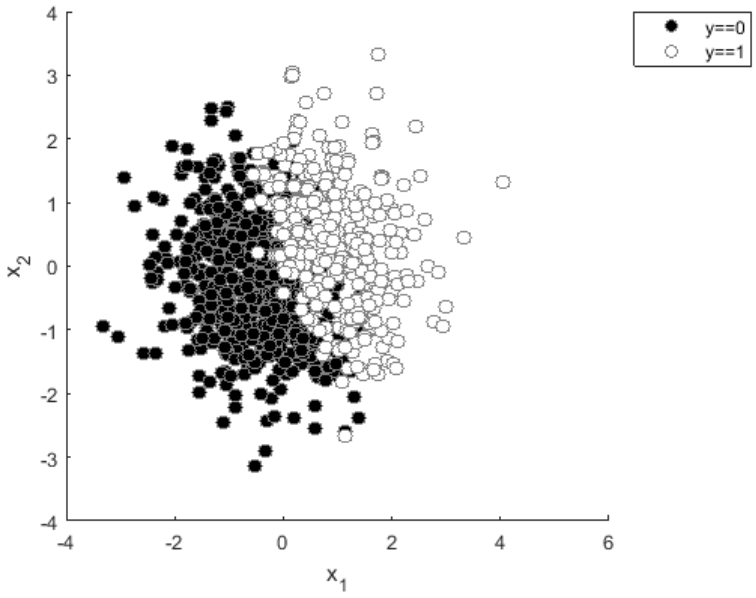
Let's start by constructing some fictional data. We draw 11000 normally distributed random values for two variables, pH (x_1) and TSS(x_2). For 1000 of these observations, we define water quality (*lhs*) as a function of these variables.

```
%% Training data generation :
% Imagine we have information on 11000 water source's pH level and TSS count.
% Suppose this data is normalized (demeaned) across samples.
% Both characteristics are roughly normally distributed
rng(23571113)

% All 11000 water source
x_ph = randn(11000,1);
x_tss = randn(11000,1);

% This is our training set (we have the true y value)
x1 = x_ph(1:1000,:); % Normalized pH
x2 = x_tss(1:1000,:); % Normalized TSS count

% Suppose the gov't made a clean/not-clean determination for 1000 sources
lhs = (2*x1 + x2 + randn(size(x1))) > 1; % Clean water (yes = 1 / no = 0)

% Let's visualize our data in the 2D characteristic space
figure; hold on;
h1 = scatter(x1(lhs==0),x2(lhs==0),50,'k','filled'); % black dots for 0
h2 = scatter(x1(lhs==1),x2(lhs==1),50,'w','filled'); % white dots for 1
set([h1 h2],'MarkerEdgeColor',[.5 .5 .5]); % outline dots in gray
legend([h1 h2],{'y==0' 'y==1'},'Location','NorthEastOutside');
```

[7] I'd strongly recommend this textbook as a primer if you're interested - Chapter 4 is a nice starting point.
[8] This subset of data is known as the *training* set in machine learning applications
[9] This subset of observations is known as the *test* set

```
xlabel('pH');
ylabel('TSS');
```

If we run the code above, it outputs the following visualization of our data in 2-dimensional space:



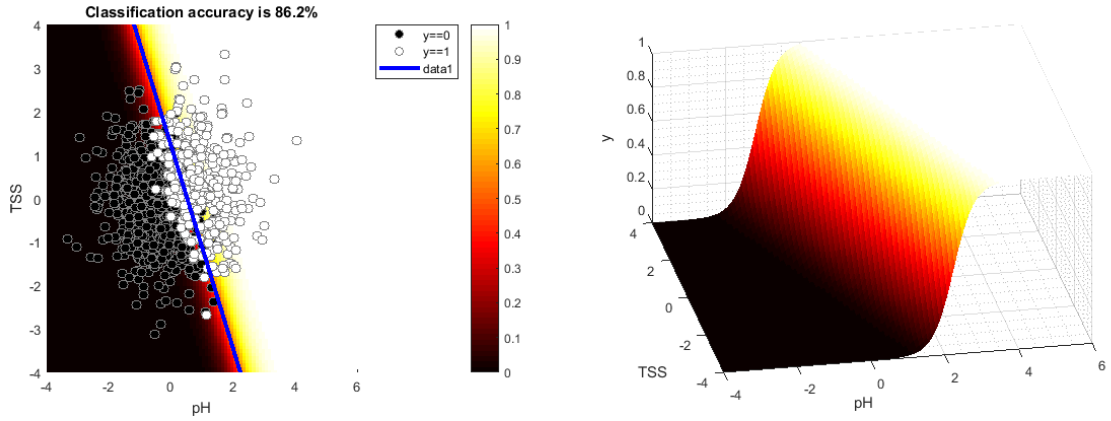
Next, we want to estimate our logit model using this data. Recall that the likelihood function the logit model is symmetric to the probit model, except for using a logistic distribution CDF instead of a normal distribution CDF - see my notes from lab 7 for this derivation. Skipping some algebra, and cutting to the chase, we code this the logit log-likelihood function in MATLAB as:

```
function [tot_llf] = logistic_llf(bo)
    global rhs lhs
    lamb = exp(-rhs*bo);
    llf = lhs.*log(1./(1+lamb)) + (1-lhs).*log(1 - (1./(1+lamb)));
    tot_llf = -(sum(llf));
end
```

We estimate the model using MATLAB's *fminsearch* function.

```
%% Logistic regression setup and estimation :
% Construct RHS vars vector:
rhs = [x1 x2];
rhs(:,end+1) = 1; % Add a constant

% Set initial paramater values
paramso = (ones(1,size(rhs,2)))'; % Set init params to 1.

% Run LLF minimization algorithm (note that we multiplied LLF by -1)
[params, fval] = fminsearch(@logistic_llf,paramso);

% Compute the model's output for each data point
modelfit = 1./(1+exp(-rhs*params)) >= 0.5; % Set to clean = 1 if fit >=0.5

% Calculate % of data pts correctly classified
pctcorrect = sum(modelfit==lhs) / length(lhs) * 100;
```

Note that we've added a constant to our parameter vector, so our probability of clean water is a function of the constant, pH and TSS. We calculate *modelfit* as the predicted values based on our



estimated parameters and RHS variables - it turns out our simple model, linear in parameters, correctly classifies 86.2% of the observations in the training data.

Obviously, we'd like this to be better, but do note that there's a trade-off between getting this number up to 100% and and applying the model to our full data. If we're not careful, we may *overfit* the model to this sub-sample. I digress, but keep this in mind if you do work like this in the future.

Let's now take our parameter estimates to the broader test data. Remember, we're imagining that the state didn't have funds to do full water quality testing at the remaining 10000 sites - we are going to classify their cleanliness based on their pH and TSS levels, as well as the parameters we just estimated using the logit model. (Also, note that for the purpose of this lab, we know the data generating process and therefore the true value of y - see line 107 below. Thus, we can compare the modeled and true value in this test data.)

```
102 %% Now let's take the model to "testing" data :
103 % 1000 other points where we "only have" the pH and TSS values.
104 % We want to predit the water cleanliness
105 x3 = x_ph(1001:11000,:); % Normalized pH
106 x4 = x_tss(1001:11000,:); % Normalized TSS (total suspended particulates) count
107 lhs = (2*x3 + x4 + randn(size(x3))) > 1; % Clean water (yes = 1 / no = 0)
108
109 % Testing data construction
110 rhs = [x3 x4];
111 rhs(:,end+1) = 1; % Add a constant
112
113 % Use params from "training" data for model fit to "testing" data
114 modelfit2 = 1./(1+exp(-rhs*params)) >= 0.5; % Set to clean = 1 if fit>=0.5
115 pctcorrect2 = sum(modelfit2==lhs) / length(lhs) * 100;
```

Upon running the model, we see that our logit approach correctly classifies the observation 87.47% of the time. More broadly speaking, as discussed by Brian last week, we might be interested in a contingency table.

```
-------------------------(PREDICTED VALS)----------
---------------------- CLEAN ------ NOT CLEAN -----
---- CLEAN ----         2669.0           738.0
-- NOT CLEAN --          515.0          6078.0
-----------------------------------------------------
 Precision:    83.83
 Recall:       78.34
```

Here, we see that our precision measure - how often stream was truly clean when we predicted it clean - is at 83.83%. Our recall measure - how often we predicted the stream clean when it was truly clean - is only 78.34%. Thus, we see that our model is slightly more likely to yield false negatives that false positives. In this context, that's probably a good thing - the health/environmental repercussions of predicting a stream clean when it is actually dirty could be sizable. Note that as Brian mentioned last week, if we wanted to boost our precision measure even more, at the prediction stage, we could require $modelfit2 > 0.6$ for classification as "clean". This stricter measure would generate fewer false positives.

We'll stop here because we're talking about the logit model and not the broader theme of classification. There's a lot more to think about on this subject - even when only using a simple logit model for classification - and if you're interested, I'd recommend diving into the broader data science literature for basic information on cross-validation, bootstrapping, and alternative classification approaches like LDA, SVM, matching, neural nets, etc.