Exercise 9 - Statistical modeling with maximum likelihood

Work through a t-test from statistical model equation to pseudocode to code

The statistical model we worked through in lecture on Wednesday was:

$y = \beta_0 + \beta_1 x + \epsilon$,

where y is a continuous dependent variable, x is a continuous independent variable, the β's describe the linear deterministic component of the regression and ϵ is distributed normally with a mean of zero and variance of σ^2.

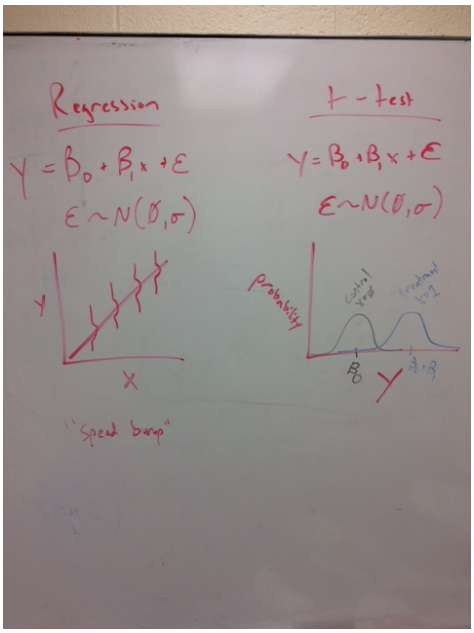
I think you can remind them of the equation above and what the symbols mean, then remind them this is the same statistical model for a t-test where we assume the variance is the same between treatments and controls, but x is 1 for treatment experimental units and 0 for control units. Talking through this while creating the picture comparing a regression and t-test included below might be helpful.

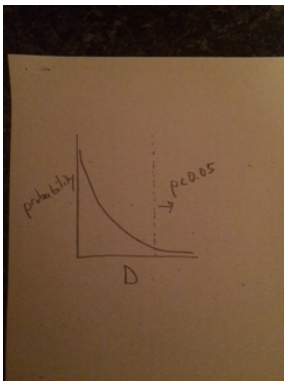
For pseudo-code, I think this is essentially the "recipe" we talked about on Wednesday, although maybe laying out the pseudo-code within the second and third step of the broader "recipe" might be helpful. Here is my quick version of that pseudo-code:

1. load data in a dataframe
2. create a custom function that specifies our model structure, takes parameters and observations as arguments, and returns the negative log likelihood of our model and parameter values given our observations. This function will include code to:
    - specify the name of our function and the arguments to be expected
    - "unpack" our vector/list of parameters
    - calculate expected values of y given the model and parameter values
    - calculate the negative log likelihood given the expected values, observed values, and parameter value for sigma
3. use our custom function, observations, and minimization function to estimate the maximum likelihood values of our parameters. These parameters tell us something about how the variables are related or effect of our treatment in the t-test case. We can also use the minimum negative log likelihood values to compare one model to another, as long as we use the same data for both models.
    - create a vector/list of initial parameter values
    - implement the minimization function with appropriate arguments

Then I think you could open up the actual code from lecture on Wednesday that shows the example for a regression and ask the students what has to change for a t-test. Actually, the only thing that needs to change is the data going in (x has to be 1's and 0's). If they don't have an answer you can walk through what the various steps of the code are doing, and help them find what has to change. If someone comes up with the answer right away, you could say "that's right, but let's walk through the code and talk about why it does or doesn't have to change".

To really do a t-test and assign a p-value, you need to compare this model ($y = \beta_0 + \beta_1 x + \epsilon$) to a "null" model ($y = \beta_0 + \epsilon$) using a likelihood ratio test. Yes, this means you'll need to write two custom likelihood functions.





Likelihood ratio tests

The homework exercise introduces a likelihood ratio test as a more formal way to compare models. I think it would be good for you to talk the students through this briefly. One key point is that this can only be used to compare models where one is a subset of the other. The basic idea here is that we expect 2 times the difference in negative log likelihoods between two models is expected to be distributed according to a chi-squared distribution with the degrees of freedom equal to the difference in number of parameters in the two models. This allows us to ask whether the observed 2 times the difference in negative log likelihoods is "extreme" relative to expected values, which allows us to assign a p-value to the model comparison and say whether the additional model parameter(s) and complexity provides a statistically significant improvement in model performance. Drawing a picture like the second one above, may be helpful.

As a reminder here is the code I provide to implement the likelihood ratio test:

For `R`: `pchisq(q=D,df=1,lower.tail=FALSE)`

For `Python`: `1-chi2.cdf(x=D,df=1)`; to use `chi2.cdf` you need to first run `from scipy.stats import chi2`

In both of these codes, D is 2 times the difference in negative log likelihoods.