Lab #9 - Logistic Regression Part I

Econ 224

September 25th, 2018

Introduction

In this lab we'll study logistic regression. The first part of the lab will involve carrying out some calculations to better understand how logistic regression works and what it means. The second part of the lab will show you the basics of how to carry out logistic regresion in R.

Part I - Theoretical

In this part of the lab, we'll carry out some theoretical derivations to better understand logistic regression. To make things simpler, we'll use some slightly different notation and terminology than ISL. First we'll define the *column vectors* X and β as follows:

$$X = \begin{bmatrix} 1 \\ X_1 \\ X_2 \\ \vdots \\ X_p \end{bmatrix}, \quad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{bmatrix}$$

Notice that the first element of X is *not* X_1: it is simply the number 1. There's an important reason for this that you'll see in a moment. From the reading, we know that logistic regression is a *linear model* for the *log odds*, namely

$$\log\left[\frac{P(X)}{1-P(X)}\right] = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$$

where $P(X)$ is shorthand for $\mathbb{P}(Y = 1|X)$. Note that when I write log I **always** mean the natural logarithm. Also note that when I write $\exp(z)$ I mean e^z. This comes in handy if z is a complicated expression.

Using the vector notation introduced above, we can express this more compactly as

$$\log\left[\frac{P(X)}{1-P(X)}\right] = X'\beta$$

since

$$X'\beta = \begin{bmatrix} 1 & X_1 & X_2 & \cdots & X_p \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{bmatrix} = \beta_0 + \beta_1 X_2 + \beta_2 X_2 + \cdots + \beta_p X_p$$

I will call $X'\beta$ the *linear predictor* since it is the linear function of X that we use to predict Y. By exponentiating both sides of the log-odds expression from above and re-arranging, obtain the following:

$$\begin{aligned}
\frac{P(X)}{1-P(X)} &= \exp(X'\beta) \\
P(X)[1+\exp(X'\beta)] &= \exp(X'\beta) \\
P(X) &= \frac{\exp(X'\beta)}{1+\exp(X'\beta)} \\
P(X) &= \Lambda(X'\beta)
\end{aligned}$$

where the function Λ is defined as follows

$$\Lambda(z) = \frac{e^z}{1+e^z}$$

Exercise #1

(a) Verify that $\Lambda(z) = \dfrac{1}{1+e^{-z}}$.
(b) Using (b), write an alternative expression for $P(X)$.

Solution to Exercise #1

(a) Dividing the numerator and denominator by e^z, which cannot result in division by zero since e^z is always positive, we have

$$\Lambda(z) = \frac{e^z}{1+e^z} = \frac{1}{1/e^z+1} = \frac{1}{1+e^{-z}}$$

(b) $P(X) = \dfrac{1}{1+\exp(-X'\beta)}$

Interpreting β in a Logistic Regression

From the expression above, we see that β_j is the partial derivative of the log-odds with respect to X_j. But it's difficult to think in terms of log-odds. By doing some calculus (see the exercises below), we can work out the partial derivative of $p(X)$ with respect to X_j, but this will *not* turn out to equal β_j. Because $P(X)$ is not a linear function of X, the derivative varies with X, which makes things fairly complicated. There are two main approaches for dealing with this problem. One is to evaluate the derivative at a "typical" value of X such as the sample mean. Another is to use the "divide by 4 rule." This rule says that if we increase X_j by one unit, $P(X)$ will change by *no more than* $\beta_j/4$. In the following exercise, you'll derive this rule.

Exercise #2

(a) Analyze the function $\Lambda(z)$: calculate its derivative, and its limits as $z \to -\infty$ and $+\infty$. What values can this function take? Is it increasing? Decreasing? Explain.
(b) Use the chain rule and your answer to (a) to find the partial derivative of $\Lambda(X'\beta)$ with respect to X_j.
(c) What is the maximum value of the *derivative* of $\Lambda(z)$? At what value of z does it occur?
(d) Use your answers to parts (a), (b) and (c) to justify the "divide by 4 rule."
(e) The "divide by 4 rule" provides an upper bound on the effect of X_j on $P(X)$. When is this upper bound close to the derivative you calculated in part (c)?

Solution to Exercise #2

(a) The function Λ takes values between 0 and 1. When $z = 0$, $\Lambda(z) = e^0/(1 + e^0) = 1/2$. As $z \to \infty$, $\Lambda(z) \to 1$ and as $z \to -\infty$, $\Lambda(z) \to 0$. We calculate its derivative using the quotient rule as follows

$$\frac{d\Lambda(z)}{dz} = \frac{e^z(1 + e^z) - e^z e^z}{(1 + e^z)^2} = \frac{e^z}{(1 + e^z)^2}$$

Since e^z is always greater than zero, the derivative is always positive so $\Lambda(z)$ is strictly increasing.

(b) The key is to treat the linear predictor $X'\beta$ as a function of X_j, namely

$$f(X_j) = X'\beta = \beta_0 + \beta_1 X_1 + \cdots + \beta_j X_j + \beta_{j+1} X_{j+1} + \cdots + \beta_p X_p$$

Now, by the chain rule we have

$$\frac{\partial \Lambda(X'\beta)}{\partial X_j} = \frac{\partial \Lambda\left(f(X_j)\right)}{\partial X_j} = \frac{\exp(X'\beta)}{[1 + \exp(X'\beta)]^2} \frac{\partial f(X_j)}{\partial X_j} = \frac{\beta_j \exp(X'\beta)}{[1 + \exp(X'\beta)]^2}$$

(c) To find the value of z that maximizes the first derivative, we take the *second* derivative of Λ as follows

$$\begin{aligned}
\frac{d^2\Lambda(z)}{dz} &= \frac{e^z(1 + e^z)^2 - 2e^z(1 + e^z)e^z}{(1 + e^z)^4} = \frac{e^z(1 + 2e^z + e^{2z}) - 2e^{2z}(1 + e^z)}{(1 + e^z)^4} \\
&= \frac{e^z + 2e^{2z} + e^{3z} - 2e^{2z} - 2e^{3z}}{(1 + e^z)^4} = \frac{e^z - e^{3z}}{(1 + e^z)^4} \\
&= \frac{e^z(1 - e^{2z})}{(1 + e^z)^4} = \frac{e^z(1 + e^z)(1 - e^z)}{(1 + e^z)^4} = \frac{e^z(1 - e^z)}{(1 + e^z)^3}
\end{aligned}$$

Thus, the first order condition is $e^z(1 - e^z) = 0$. Since e^z cannot equal zero for any z, the only way for this equation to be satisfied is if $e^z = 1$ which occurs precisely when $z = 0$. Substituting into our expression from (a), we find that the derivative of $\Lambda(z)$ at $z = 0$ is $e^0/(1 + e^0)^2 = 1/(1 + 1)^2 = 1/4$.

(d) From part (a), we know that the derivative of $\Lambda(z)$ equals $e^z/(1 + e^z)^2$ which is always positive. From part (c) we know that this derivative is *at most* $1/4$. Therefore, the partial derivative of $\Lambda(X'\beta)$ with respect to X_j is *at most* $\beta_j \times 1/4 = \beta_j/4$.

(e) When $X'\beta \approx 0$ it follows that $\exp(X'\beta)/[1 + \exp(X'\beta)]^2 \approx 1/4$ so the "divide by four" rule gives a good approximation to the actual derivative.

The Latent Data Formulation of Logistic Regression

Another way to think about logistic regression is via the following *generative model*:

$$y_i^* = X_i'\beta + \epsilon_i, \quad y_i = \begin{cases} 1 & \text{if } y_i^* > 0 \\ 0 & \text{if } y_i^* \leq 0 \end{cases}, \quad \epsilon_i \sim \text{ iid Logistic}(0, 1)$$

where the Logistic$(0, 1)$ distribution has CDF $\Lambda(z) = e^z/(1+e^z)$ and pdf $\lambda(z) = e^z/(1+e^z)^2$. The expressions Λ and λ should look familiar, since we worked with them above. We call this a generative model because it tells us how to *generate* the observations y_i using the regressors X_i. If we want to *simulate* data from a logistic regression model, the latent data formulation gives us a convenient way to do so.

The idea behind the latent data formulation is that a continuous *unobserved* random variable y_i^* determines whether the *observed* binary random variable y_i is zero or one. The term *latent* is just a synonym for unobserved. While this may seem odd, in specific examples we can often give y_i^* a meaningful interpretation. For example, suppose that $y_i = 1$ if person i voted for Hilary Clinton in the 2016 presidential election and X_i contains demographic information, e.g. income, education, race, sex, and age. The latent variable y_i^* can be viewed as a measure of person i's *strength of preference* for Hilary Clinton relative to Donald Trump. If y_i^* is large and positive, person i strongly prefers Clinton; if y_i^* is large and negative, person i strongly prefers Trump; if $y_i^* = 0$, person i is indifferent.

Exercise #3

(a) Show that $\lambda(z)$ is symmetric about zero, i.e. $\lambda(z) = \lambda(-z)$.
(b) Show that the latent data formulation implies $\mathbb{P}(y_i = 1) = \Lambda(X_i'\beta)$. Hint: if Z is a continuous RV with a pdf that is symmetric about zero, then $\mathbb{P}(-Z < c) = \mathbb{P}(Z \leq c)$.

Solution to Exercise #3

(a) Expand the denominator, and then multiply by $\exp(-2z)/\exp(-2z)$, yielding

$$\lambda(x) = \frac{\exp(z)}{[1+\exp(z)]^2} = \frac{\exp(z)}{1+2\exp(z)+\exp(2z)} = \frac{\exp(-z)}{\exp(-2z)+2\exp(-z)+1} = \frac{\exp(-z)}{[1+\exp(-z)]^2} = \lambda(-z)$$

(b) $\mathbb{P}(y_i = 1) = \mathbb{P}(y_i^* > 0) = \mathbb{P}(X_i'\beta + \epsilon_i > 0) = \mathbb{P}(-\epsilon_i < X_i'\beta) = \mathbb{P}(-\epsilon_i \leq X_i'\beta) = \Lambda(X_i'\beta)$

Part II - Logistic Regression in R

Now we'll take a quick look at how to carry out logistic regression in R using a simulated dataset. In Thursday's lab you'll use what you learn in this part to study a real-world example.

Simulating Data from a Logistic Regression

The R function `rlogis` creates iid draws from the logistic distribution. If we only specify one argument, `rlogis` assumes that this is the number of random draws that we wish to make, and sets the values of its *location* and *scale* parameters to 0 and 1, respectively. This is what we want, since these parameters correspond to the Logistic$(0, 1)$ distribution that appears in the latent data formulation from above. Using `rlogis`, we can simulate data from a logistic regression model as follows:

```
set.seed(1234)
n <- 500
b0 <- 0.5
b1 <- 1
x <- rnorm(n, mean = 1.5, sd = 2)
ystar <- b0 + b1 * x + rlogis(n)
y <- 1 * (ystar > 0)
mydat <- data.frame(x, y)
```

Running a Logistic Regression in R

We can now run a logistic regression use the simulated dataset `mydat` to carry out logistic regression. Note that in a certain sense this is silly: we generated the data so we *know* the true values of β_0 and β_1. Why bother carrying out logistic regression to *estimate* them? There are two answers to this question. First, this is only an example: don't be so picky! Second, it can be extremely valuable to work with simulated data to check whether our statistical methods are working correctly. If we *know* for sure that the data came from a logistic regression model, then our logistic regression estimates should be close to the truth. If they're not, then something is wrong with our computer code.

The R function `glm` can be used to carry out logistic regression. The name of this function is an acronym for *generalized linear model*. Generalized linear models (GLMs) are exactly what their name says, a *generalization*

of linear regression. GLMs include logistic regression as a special case. To tell `glm` that we want to carry out a logistic regression, we need to specify `family = binomial(link = 'logit')`. Otherwise the syntax is practically identical to that of `lm`. We specify a *formula*, `y ~ x`, and indicate a dataframe in which R should look to find `y` and `x`:

```
lreg <- glm(y ~ x, mydat, family = binomial(link = 'logit'))
summary(lreg)
```

```
Call:
glm(formula = y ~ x, family = binomial(link = "logit"), data = mydat)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-2.61173   0.04538   0.30466   0.63221   1.88450

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)   0.3630     0.1344   2.700  0.00693 **
x             0.9638     0.1004   9.596  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 555.65  on 499  degrees of freedom
Residual deviance: 381.74  on 498  degrees of freedom
AIC: 385.74

Number of Fisher Scoring iterations: 6
```

Notice that the output of `summary` when applied to a `glm` object is a little different from what we've seen for `lm` objects. But let's focus on what's the same. We still obtain the estimates of each of the coefficients in our model, along with standard errors, test statistics, and p-values. We can use this information to carry out statistical inference exactly as we do with linear regression: R has already done all the hard work for us by calculating the standard errors.

Exercise #4

Construct approximate 95% confidence intervals for the parameters β_0 and β_1 based on the logistic regression output from above. Do your confidence intervals include the true parameter values that we used to simulate the data?

Solution to Exercise #4

The confidence interval for the regression intercept is 0.36 ± 0.27 which includes the true value: $\beta_0 = 0.5$. Similarly, the confidence interval for the regression slope is 0.96 ± 0.2 which includes the true value: $\beta_1 = 1$.

Predicted Probabilities for Logistic Regression

Many of the functions we used with `lm` also work with `glm`. For example, to extract the coefficients from a generalized linear model, we can use the command `coef`:

```
coef(lreg)
```

```
(Intercept)           x
  0.3629592   0.9637594
```

We can also use the function `predict` to calculated the predicted probability that $y = 1$ given particular values of the predictors X_i. There's just one slight wrinkle here: we need to make sure to specify `type = 'response'` to indicate to R that we want the predicted *probabilities*. For example, we can calculate the predicted probability that $y_i = 1$ given that $X_i = 0$ as follows:

```
predict(lreg, newdata = data.frame(x = 0), type = 'response')
```

```
        1
0.5897566
```

Similarly, we can calculate the predicted probability that $y_i = 1$ given that X_i equals the *sample mean* of X as follows:

```
predict(lreg, newdata = data.frame(x = mean(x)), type = 'response')
```

```
        1
0.8596206
```

If we don't specify anything for `newdata`, then predict will give us the predicted probabilities for the *observed* values of X:

```
phat <- predict(lreg, type = 'response')
head(phat)
```

```
         1          2          3          4          5          6
0.37330981 0.91240407 0.98013788 0.06222354 0.93312689 0.94180674
```

Exercise #5

(a) Write an R function called `Lambda` that calculates the value of $e^z/(1 + e^z)$.
(b) Using your function from part (a) and the results of `lreg`, calculate the predicted probability that $y_i = 1$ when: (i) $X_i = 0$ and (ii) $X_i = \bar{X}$ *without using* `predict`. Check that your results match those calculated using `predict` above.

Solution to Exercise #5

```r
Lambda <- function(x) {
  1 / (1 + exp(-x))
}
bhat_0 <- coef(lreg)[1]
bhat_1 <- coef(lreg)[2]
Lambda(bhat_0)
```

```
(Intercept)
  0.5897566
```

```r
Lambda(bhat_0 + bhat_1 * mean(x))
```

```
(Intercept)
  0.8596206
```

Calculating Marginal Effects

As we discussed above, β_j is *not* the partial derivative of $P(X)$ with respect to X_j. But since we have a formula for this partial derivative, we can calculate it for any value of X. In the following exercise, you will compare the exact calculation to the approximation given by the "divide by 4" rule.

Exercise #6

(a) Use the "divide by 4" rule to calculate the *maximum* possible effect of X on the predicted probability that $y_i = 1$ using the results of `lreg`.
(b) Calculate the effect of X on the predicted probability that $y_i = 1$ at $X_i = \bar{X}$.
(c) Compare your answers to (a) and (b)

Solution to Exercise #6

```r
# Divide by 4 rule
bhat_1 / 4
```

```
        x
0.2409399
```

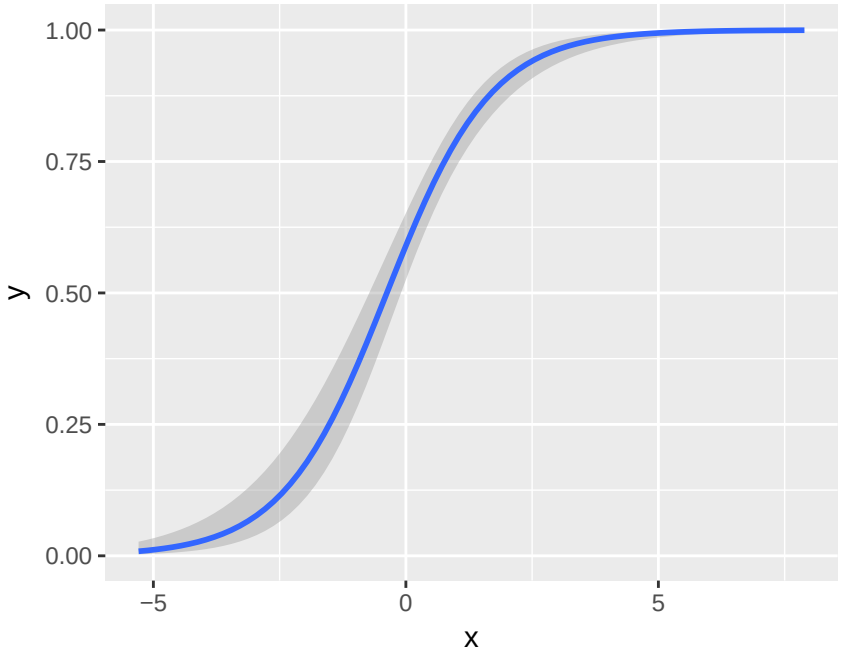
```r
# Marginal effect at average x
linear_predictor <- bhat_0 + bhat_1 * mean(x)
bhat_1 * exp(linear_predictor) / (1 + exp(linear_predictor))^2
```

```
        x
0.1162997
```

Plotting a Logistic Regression

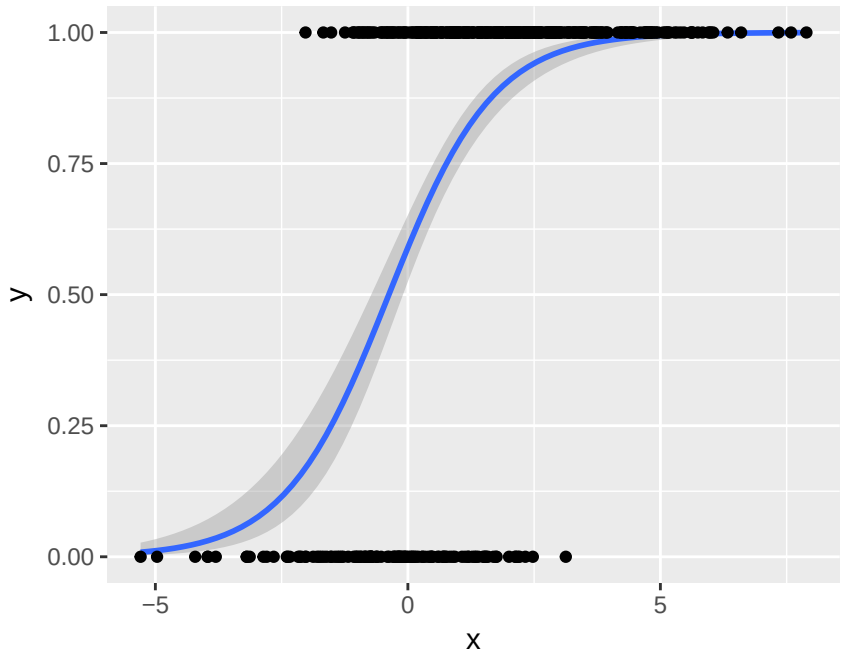
We can plot a logistic regression function using a method very similar to the one we used to plot a linear regression:

```
library(ggplot2)
ggplot(mydat, aes(x, y)) +
  stat_smooth(method='glm',
              method.args = list(family = "binomial"),
              formula = y ~ x)
```



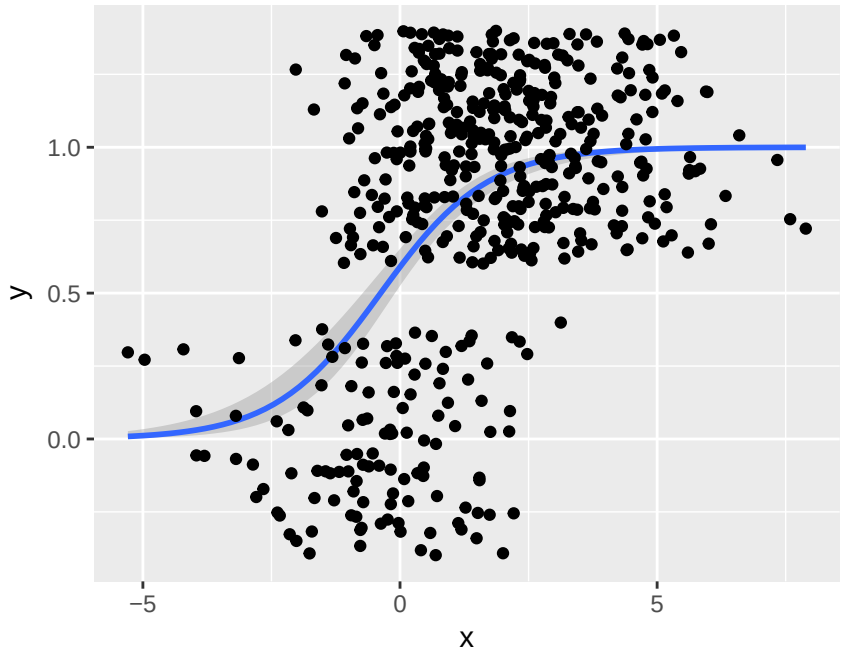
To add the datapoints, we just add `geom_point()`

```
library(ggplot2)
ggplot(mydat, aes(x, y)) +
  stat_smooth(method='glm', method.args = list(family = "binomial"),
              formula = y ~ x) +
  geom_point()
```



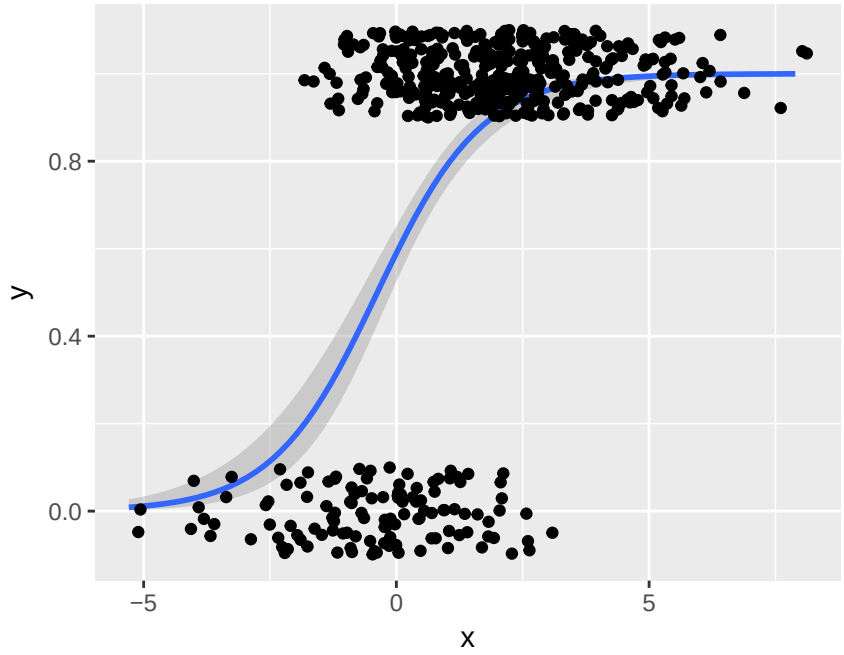
This doesn't look very nice! That's because there are only *two* possible y-values meaning that the observations will overlap substantially. A helpful way to distinguish them visually is to add a bit of random noise to the points so they no longer overlap. This is called *jittering* and `ggplot2` will do it for us if we replace `geom_point()` with `geom_jitter()`

```
library(ggplot2)
ggplot(mydat, aes(x, y)) +
  stat_smooth(method='glm', method.args = list(family = "binomial"),
              formula = y ~ x) +
  geom_jitter()
```



That's a bit *too much* random noise in the y-dimension. We can control the amount of jittering by specifying `width` and `height` arguments to `geom_jitter` as follows

```
library(ggplot2)
ggplot(mydat, aes(x, y)) +
  stat_smooth(method='glm', method.args = list(family = "binomial"),
              formula = y ~ x) +
  geom_jitter(width = 0.5, height = 0.1)
```



From this plot it is easy to tell that there are many more observations with $y = 1$ than $y = 0$, something that was not at all clear from the plot using `geom_point()`.