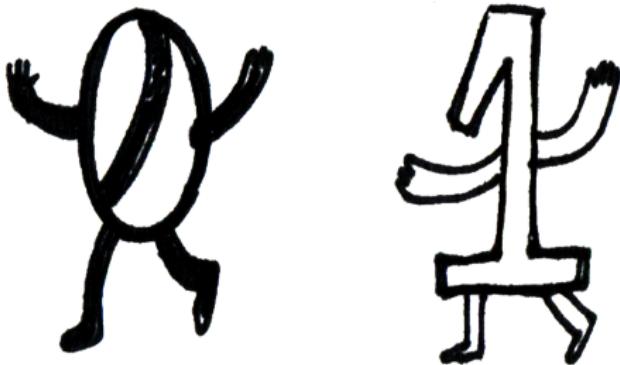


How to Build a Logit Model from Scratch



Jon Kropko
University of Virginia
June 11, 2019

Before we get started

You can access the slides, data, and code for today's class at

github.com/jkropko/ds_sampleclass

I imagine this class to be in the **middle of a semester** on probability modeling. I assume students have prior exposure to the following topics:

- ▶ Probability and distributions
- ▶ Linear regression and OLS
- ▶ R programming, including writing functions
- ▶ Pre-calculus, derivatives, and basic matrix operations

Please feel free to ask questions at any time!

What is Logit?

Running Logit Models in R

Climbing Out of the Black Box

The Math Behind Logit Models

Generalized linear models (GLMs)

Logit as a GLM

Logit's likelihood function

Logit's log-likelihood function

Programming Our Own Logit from Scratch in R

Coding the log-likelihood as an R function

Using `optim()`

Conclusion: You are a Craftsman, not just a User

Appendix 1: Interpreting Logit Model Results

Odds ratios (from a logit model)

Why odds ratios are going out of style

Predicted probability

Appendix 2: Algebraically Reducing the Logit Log-Likelihood

Logistic Regression (logit)

Logistic regression (also called **logit**) is a tool for modeling the variance of a **binary** (two values, usually coded 0 and 1) outcome variable.

There are two important ways we use logit models:

1. Calculating the probability that the outcome is 1 (vs. 0) given values of a set of X variables
2. Assessing whether each X has an effect on this probability (and how big is the effect)

In practice, the code and results of a logit model look a lot like **linear regression**.

R Code

Let's use data from the **American National Election Study**. There are 2,795 observations collected from surveys just before and after the 2016 U.S. presidential election.

The outcome variable is **vote**, where 0 indicates a vote for Donald Trump and 1 indicates a vote for Hillary Clinton.

I regress the vote on a voter's:

- ▶ **age** – years
- ▶ **marital status** – single, married, no longer married
- ▶ **education** – less than a HS diploma, HS diploma, some college, college degree, graduate degree
- ▶ **union membership status** – member (1) or non-member (0)
- ▶ **race** – white, black, Hispanic, other
- ▶ **gender** – male, female

R Code

The code to run a logit model is

```
logit <- glm(vote ~ age + marital + education + union +
              race + gender,
              data = anes,
              family=binomial(link="logit"))
```

There are **two differences** between this code and the code to run a linear regression:

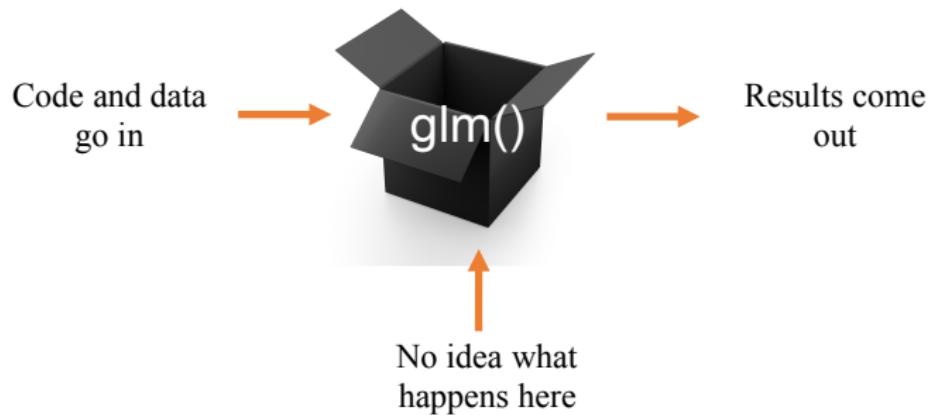
1. The function is `glm()` instead of `lm()`, because logit is a special example of a **generalized linear model** (GLM)
2. The argument `family=binomial(link="logit")` tells R that the particular GLM we want to run is logit. We will discuss the logic of this syntax more detail in a few minutes.

Try running the model yourself.

Climbing out of the Black Box

Great! We've used the right code to run a logit model. If all we care about is **getting coefficient estimates**, we can stop here.

That's an example of something called a **black box approach** to statistics:



Many systems in real life work this way (the **human brain** for example!) but **logit doesn't have to be a black box**.

Climbing out of the Black Box

Logit is a mathematical model of **probability**. If we delve into the underlying math, we will *really* understand how logit works.

Then we can **adapt it to our own problems**. We can even **invent new methods** by changing parts of this construction.

Our plan today:

1. Go over the math, and build the **log-likelihood function**: a blueprint for a logit model
2. Program our own logit models **from scratch** in R, without using any pre-programmed logit function like `glm()`.

Also, there are better ways than reporting coefficients to express the results of a logit model. For details, see the [appendix to these slides](#).

Generalized Linear Models (GLMs)

The purpose of a GLM is to map values of X variables onto probabilities of values of Y.

A GLM has three parts:

The family: a **probability density/mass function** for the outcome

The linear model: a linear combination of the X variables that will be **substituted for a parameter** of the family

A link function: a mathematical function that transforms the linear model so that it has the **same support as the parameter** it's being plugged into

Logit as a GLM

Logit is just **one example** of a GLM. Other famous models (probit, ordered logit, Poisson, etc.) are also GLMs, and differ from logit only in the choice of family and link function.

Logit's **family** is the **Bernoulli distribution** (a special case of the binomial distribution with one trial):

$$f(y_i|p_i) = p_i^{y_i} (1 - p_i)^{1-y_i}$$

There's one parameter p_i , which represents the probability that $y_i = 1$.

The **linear model** is an equation containing all of the Xs. We've seen equations like this before with regression models. I denote the linear model as y_i^* :

$$y_i^* = \alpha + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_k x_{ik}.$$

Logit as a GLM

Family: $f(y_i|p_i) = p_i^{y_i}(1 - p_i)^{1-y_i}$

Linear model: $y_i^* = \alpha + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_k x_{ik}$

The linear model can take on **all real numbers**. And we want to substitute it for the probability parameter p_i , which can only be **between 0 and 1**.

We need a math function that **takes all reals** as inputs, but outputs **numbers between 0 and 1**. Can you think of one?

For logit, we use this particular link function:

$$p_i = \frac{1}{1 + e^{-y_i^*}},$$

which is **CDF of the standard logistic function** – hence the name “logit”.

Logit as a GLM

Family: $f(y_i|p_i) = p_i^{y_i} (1 - p_i)^{1-y_i}$

Linear model: $y_i^* = \alpha + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_k x_{ik}$

Link function: $p_i = \frac{1}{1 + e^{-y_i^*}}$

One way to write the **complete GLM** for logit is

$$f(y_i|y_i^*) = \left(\frac{1}{1 + e^{-y_i^*}} \right)^{y_i} \left(1 - \frac{1}{1 + e^{-y_i^*}} \right)^{1-y_i}$$

where $y_i^* = \alpha + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_k x_{ik}$.

This GLM expresses the probabilities for **one particular observation's outcome**. Next we need to write the *joint distribution* for ALL the observations in the data.

Logit's Likelihood Function

$$f(y_i|y_i^*) = \left(\frac{1}{1 + e^{-y_i^*}} \right)^{y_i} \left(1 - \frac{1}{1 + e^{-y_i^*}} \right)^{1-y_i}$$

We need to assume that all the observations are **independent** and **identically distributed** (they all share this GLM).

Remember, if random variables are independent, then their **joint distribution** is the **product of their marginal distributions**.

So the joint distribution of all observations in the data is the **product of all observations' GLMs**:

$$f(Y|Y^*) = \prod_{i=1}^N \left(\frac{1}{1 + e^{-y_i^*}} \right)^{y_i} \left(1 - \frac{1}{1 + e^{-y_i^*}} \right)^{1-y_i}$$

Logit's Likelihood Function

$$f(Y|Y^*) = \prod_{i=1}^N \left(\frac{1}{1 + e^{-y_i^*}} \right)^{y_i} \left(1 - \frac{1}{1 + e^{-y_i^*}} \right)^{1-y_i}$$

Notice that this function contains both **data** (in the X and Y variables) and **coefficients** (in the linear model).

If we think of the **coefficients as the unknowns** and the data as known, then we call this function a likelihood function, with this notation:

$$L(\alpha, \beta_1, \dots, \beta_k | X, Y) = \prod_{i=1}^N \left(\frac{1}{1 + e^{-y_i^*}} \right)^{y_i} \left(1 - \frac{1}{1 + e^{-y_i^*}} \right)^{1-y_i}$$

A likelihood function is **the DNA of a statistical model**. It's a complete representation of all information in the model, and we need the likelihood function to get coefficient estimates.

Logit's Likelihood Function

$$L(\alpha, \beta_1, \dots, \beta_k | X, Y) = \prod_{i=1}^N \left(\frac{1}{1 + e^{-y_i^*}} \right)^{y_i} \left(1 - \frac{1}{1 + e^{-y_i^*}} \right)^{1-y_i}$$

Here's how likelihood functions work:

1. Plug candidate values for every coefficient into the likelihood function
2. The function outputs a value called a **likelihood**
3. The **higher the likelihood**, the more likely it is that the data could have been generated by a model with the coefficients you've inputted

Our goal: find the coefficient values that MAXIMIZE the likelihood function.

Logit's Log-Likelihood Function

To solve this optimization problem, we will use a **hill-climber algorithm**, which is built into R.

We will get faster and more accurate results, however, if we maximize the **natural logarithm** of the likelihood function. The log has several really useful properties:

- ▶ Taking the log of a function does **not change the location of maximum and minimum points**
- ▶ Exponents inside a log become factors outside the log
- ▶ Multiplication inside a log becomes addition outside the log
- ▶ Division inside a log becomes subtraction outside the log

These properties mean that the hill-climber will have a much easier time maximizing the **log-likelihood function**.

Logit's Log-Likelihood Function

Likelihood function:

$$L(\alpha, \beta_1, \dots, \beta_k | X, Y) = \prod_{i=1}^N \left(\frac{1}{1 + e^{-y_i^*}} \right)^{y_i} \left(1 - \frac{1}{1 + e^{-y_i^*}} \right)^{1-y_i},$$

Log-likelihood function (denoted with ℓ instead of L):

$$\ell(\alpha, \beta_1, \dots, \beta_k | X, Y) = \ln \left[\prod_{i=1}^N \left(\frac{1}{1 + e^{-y_i^*}} \right)^{y_i} \left(1 - \frac{1}{1 + e^{-y_i^*}} \right)^{1-y_i} \right]$$

The log-likelihood function algebraically reduces to:

$$\ell(\alpha, \beta_1, \dots, \beta_k | X, Y) = \sum_{i=1}^N -y_i^*(1 - y_i) - \ln(1 + e^{-y_i^*}).$$

(To see the algebra, see the [appendix](#).)

Coding the log-likelihood as an R function

To estimate any model, all you need to do is program the model's **log-likelihood function** into R.

We'll do that by writing a function whose **inputs are possible values of every coefficient**, and whose output is a single number representing the value of the log-likelihood.

We will also use our **ANES data** inside this function.

Here's the code to program the logit log-likelihood into R. We'll go over this code line by line:

```
logit.ll <- function(parameters, formula, data, outcome){  
    Xmat <- model.matrix(terms(formula), data=data)  
    ystar <- Xmat %*% parameters  
    ll <- -ystar*(1 - outcome) - log(1 + exp(-ystar))  
    return(sum(ll))  
}
```

Coding the log-likelihood as an R function

```
logit.ll <- function(parameters, formula, data, outcome){  
    Xmat <- model.matrix(terms(formula), data=data)  
    ystar <- Xmat %*% parameters  
    ll <- -ystar*(1 - outcome) - log(1 + exp(-ystar))  
    return(sum(ll))  
}
```

The function has four arguments:

- ▶ **parameters** – a vector of possible values for the **intercept and each coefficient**
- ▶ **formula** – the **linear model**, using the same syntax we use with the **lm()** or **glm()** functions
- ▶ **data** – the data frame to analyze
- ▶ **outcome** – the dependent variable

Coding the log-likelihood as an R function

```
logit.ll <- function(parameters, formula, data, outcome){  
  Xmat <- model.matrix(terms(formula), data=data)  
  ystar <- Xmat %*% parameters  
  ll <- -ystar*(1 - outcome) - log(1 + exp(-ystar))  
  return(sum(ll))  
}
```

The `model.matrix()` and `terms()` functions together **restrict the data to only the variables we need**. They also **break categorical variables into binary ones**, leaving out the reference category.

The `%*%` syntax is **matrix multiplication**. We multiply the X matrix by the coefficients to get the **linear model**.

Coding the log-likelihood as an R function

```
logit.ll <- function(parameters, formula, data, outcome){  
  Xmat <- model.matrix(terms(formula), data=data)  
  ystar <- Xmat %*% parameters  
  ll <- -ystar*(1 - outcome) - log(1 + exp(-ystar))  
  return(sum(ll))  
}
```

On these two lines of code, we type the **logit log-likelihood** directly into R. Take a look at how the code matches up with:

$$\ell(\alpha, \beta_1, \dots, \beta_k | X, Y) = \sum_{i=1}^N -y_i^*(1 - y_i) - \ln(1 + e^{-y_i^*}).$$

The `sum()` function represents the **summation**.

The `return()` function sets the value of the log-likelihood to be the output of the function.

Using optim()

Now that we've programmed the log-likelihood, we call a **hill-climbing algorithm** by calling the optim() function:

```
our.logit <- optim(par = c(3,rep(0, 12)),
                     fn = logit.ll,
                     formula = as.formula(~ age + marital + education +
                                           union + race + gender),
                     data = anes,
                     outcome = anes$vote,
                     method="BFGS",
                     hessian = TRUE,
                     control=list(fnscale=-1, maxit=5000))
```

We'll go over this code line by line as well.

Using optim()

```
our.logit <- optim(par = c(3,rep(0, 12)),
                    fn = logit.ll,
                    formula = as.formula(~ age + marital + education +
                                         union + race + gender),
                    data = anes,
                    outcome = anes$vote,
                    method="BFGS",
                    hessian = TRUE,
                    control=list(fnscale=-1, maxit=5000))
```

Hill-climbing algorithms have to start somewhere. The `par` argument sets the **starting values** of the parameters to be estimated.

Our logit model has an **intercept and 12 coefficients**. I set the starting value of the intercept to 3, and to 0 for the 12 coefficients.

Using optim()

```
our.logit <- optim(par = c(3,rep(0, 12)),  
                    fn = logit.ll,  
                    formula = as.formula(~ age + marital + education +  
                                         union + race + gender),  
                    data = anes,  
                    outcome = anes$vote,  
                    method="BFGS",  
                    hessian = TRUE,  
                    control=list(fnscale=-1, maxit=5000))
```

fn is the function we created to express the **log-likelihood function**.

formula, data, and outcome are the same as defined in the logit.ll() function.

Using optim()

```
our.logit <- optim(par = c(3,rep(0, 12)),  
                    fn = logit.ll,  
                    formula = as.formula(~ age + marital + education +  
                                         union + race + gender),  
                    data = anes,  
                    outcome = anes$vote,  
                    method="BFGS",  
                    hessian = TRUE,  
                    control=list(fnscale=-1, maxit=5000))
```

method allows us to choose between different [hill-climbing algorithms](#). BFGS is the “Broyden-Fletcher-Goldfarb-Shanno” algorithm. (To see other options, type ?optim and read under Details.)

hessian=TRUE estimates variances and covariances for our estimates, and allows us to extract [standard errors](#).

Using optim()

```
our.logit <- optim(par = c(3,rep(0, 12)),  
                    fn = logit.ll,  
                    formula = as.formula(~ age + marital + education +  
                                         union + race + gender),  
                    data = anes,  
                    outcome = anes$vote,  
                    method="BFGS",  
                    hessian = TRUE,  
                    control=list(fnscale=-1, maxit=5000))
```

By default, `optim()` **minimizes** functions instead of maximizing. Setting `fnscale=-1` tells `optim()` to maximize the log-likelihood function.

`maxit=5000` sets a maximum of 5000 iterations to **arrive at the maximum** of the log-likelihood function. Most of the time, however, we only need a couple dozen iterations.

Using optim()

```
our.logit <- optim(par = c(3,rep(0, 12)),  
                    fn = logit.ll,  
                    formula = as.formula(~ age + marital + education +  
                                         union + race + gender),  
                    data = anes,  
                    outcome = anes$vote,  
                    method="BFGS",  
                    hessian = TRUE,  
                    control=list(fnscale=-1, maxit=5000))
```

Try running the code!

- ▶ `our.logit$par` shows our **coefficients**.
- ▶ `sqrt(diag(solve(-our.logit$hessian)))` provides the **standard errors**.

How do these compare to the results we get by using the canned `glm()` function?

You are a Craftsman, not just a User

To recap: we **rebuilt logit** from a theoretically chosen **family**, **linear model**, and **link function**. You can change these if you want!

We used these elements to construct logit's **log-likelihood function**. And we passed the log-likelihood to a hill-climber in R to estimate the model.

These steps are important for getting a complete understanding of an important method like logit. But more than that:

You have the tools to build and estimate your own original probability models.

As such, you should think of yourselves as people who can craft methods tailored to a problem at hand.

Thank you!

Interpreting results

For logit models, the interpretation of β you learned in regression class **no longer applies**. My personal rule is:

If all you can do is interpret the sign and significance of an estimate, you've done it wrong.

We need to be able to interpret the **magnitude** of an effect in order to make results substantively meaningful.

There are three ways to interpret binary regression results:

1. **Odds ratios** (going out of style fast)
2. **Predicted probabilities** (and differences in probability)
3. **Marginal changes in probability** (the first derivative of probability)

Odds

Odds: The probability **for** an event divided by the probability **against** the event.

Example: probability of $y = 1$ is .6. Odds are

$$\frac{.6}{.4} = \frac{6}{4} = \frac{3}{2} = \text{"3 to 2 for"}$$

If we flip the fraction, it's called **"2 to 3 against"**.

Usually, when a gambler talks about odds, it's the odds **against**.
Odds are useful in gambling because they are interpreted as payouts. **The second number is the bet required to get the first number in profit.**

So in the above example, if I bet \$3 on an event and it happens, I get \$5: my \$3 returned and **\$2 in profit**.

Odds ratios from a logit model

The sizes of the coefficients for probit and c-log-log mean nothing.
But **logit** coefficients can be interpreted as “**log-odds**”:

$$p_i = \frac{1}{1 + e^{-y_i^*}}, \quad (1 + e^{-y_i^*})p_i = 1,$$

$$p_i + p_i e^{-y_i^*} = 1, \quad p_i e^{-y_i^*} = 1 - p_i,$$

$$e^{-y_i^*} = \frac{1 - p_i}{p_i}, \quad e^{y_i^*} = \frac{p_i}{1 - p_i},$$

$$y_i^* = \ln \left(\frac{p_i}{1 - p_i} \right),$$

$$\alpha + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_k x_{ik} = \ln \left(\frac{p_i}{1 - p_i} \right)$$

Odds ratios from a logit model

$$\alpha + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_k x_{ik} = \ln \left(\frac{p_i}{1 - p_i} \right)$$

Since we take the log, $\ln \left(\frac{p_i}{1 - p_i} \right)$ called the “log-odds”.

Direct interpretation of a logit coefficient: one-unit increase in x_{ik} is associated with a β_k change in the log-odds, on average, after controlling for the other x variables.

But this statement doesn't really tell us anything about the substance of an effect.

To express the odds, exponentiate both sides:

$$e^{\alpha + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_k x_{ik}} = \frac{p_i}{1 - p_i}.$$

Odds ratios from a logit model

Odds Ratio: How do the odds change (**multiplicatively**) for a one-unit increase in x ?

For logit, the odds ratio is

$$\begin{aligned}\frac{\text{Odds with } (x_{i1} + 1)}{\text{Odds with } (x_{i1})} &= \frac{e^{\alpha + \beta_1(\textcolor{red}{x_{i1}}+1) + \beta_2 x_{i2} + \dots + \beta_k x_{ik}}}{e^{\alpha + \beta_1 \textcolor{red}{x_{i1}} + \beta_2 x_{i2} + \dots + \beta_k x_{ik}}} \\ &= \frac{e^{\alpha + \beta_1 \textcolor{red}{x_{i1}} + \beta_1 + \beta_2 x_{i2} + \dots + \beta_k x_{ik}}}{e^{\alpha + \beta_1 \textcolor{red}{x_{i1}} + \beta_2 x_{i2} + \dots + \beta_k x_{ik}}} \\ &= \frac{e^{\beta_1} e^{\alpha + \beta_1 \textcolor{red}{x_{i1}} + \beta_2 x_{i2} + \dots + \beta_k x_{ik}}}{e^{\alpha + \beta_1 \textcolor{red}{x_{i1}} + \beta_2 x_{i2} + \dots + \beta_k x_{ik}}} = e^{\beta_1}.\end{aligned}$$

Odds ratio interpretation of a logit coefficient: “a one-unit increase in x_{ik} is associated with **multiplying the odds** by e^{β_k} , on average, after controlling for the other x variables.”

Odds ratios from a logit model

Example: logit coefficient is $\beta = 0.25$.

A one-unit increase in x_i is associated with multiplying the odds by $e^{0.25} = 1.28$, on average, after controlling for the other x variables.

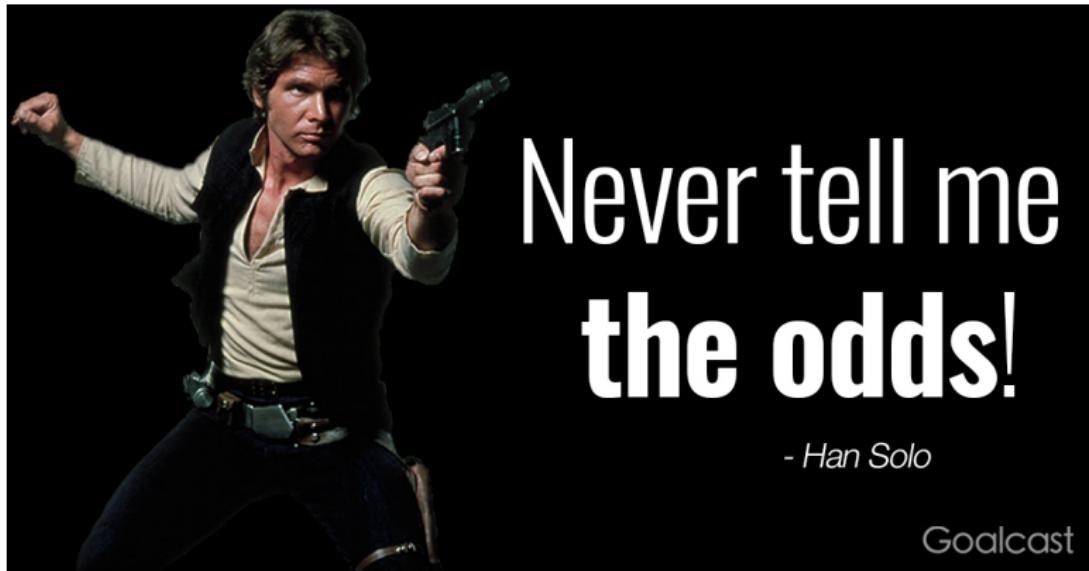
A one-unit increase in x_i is associated with the outcome becoming 28% “more likely”, on average, after controlling for the other x variables.

Example: logit coefficient is $\beta = -0.5$.

A one-unit increase in x_i is associated with multiplying the odds by $e^{-0.5} = .61$, on average, after controlling for the other x variables.

A one-unit increase in x_i is associated with the outcome becoming 39% “less likely”, on average, after controlling for the other x variables.

Why odds ratios are going out of style



- Han Solo

Goalcast

Why odds ratios are going out of style

The odds are odd!

Odds are used in things like horse racing, but social scientists **NEVER** talk about probability in this weird way.

Which makes more sense?

.25 probability or $\frac{p}{1-p} = \frac{.25}{.75} = 3 \text{ to } 1 \text{ odds against}$

.8 probability or $\frac{p}{1-p} = \frac{.8}{.2} = 4 \text{ to } 1 \text{ odds for}$

Why odds ratios are going out of style

The odds can be misleading!

Example: Suppose when $X = c$, $p = .4$, and the odds are

$$\frac{.4}{.6} = .667.$$

Also suppose when $X = c + 1$, $p = .95$, then the odds are

$$\frac{.95}{.05} = 19.$$

Then the odds ratio (**multiplicative change in the odds**) is

$$\frac{19}{.667} = 28.5.$$

So a one-unit increase in X is associated with a **2750% increase** in the odds that $y = 1$. That's not wrong, *but it is a misleading way* to communicate this finding. It's much better to report that we go from a .4 probability to a .95 probability.

Why odds ratios are going out of style

Odds are easy to misunderstand and incorrectly interpret!

Odds ratios are the multiplicative change in the **odds for** an event.

They are **NOT** multiplicative changes in probability.

$$\text{Mult. change in odds} = \frac{.95/.05}{.4/.6} = 28.5$$

$$\text{Mult. change in probability} = \frac{.95}{.4} = 2.38$$

When a researcher says “more likely” it’s not clear if that refers to **odds or probability**. *Is the event 28.5 times more likely, or just 2.38 times more likely?*

Instead, let’s compute **predicted probability** and **marginal changes in probability**.

Predicted Probability

For logit, predicted probability is

$$P(y_i = 1) = \pi_i = \frac{1}{1 + e^{-(\hat{\alpha} + \hat{\beta}_1 x_{i1} + \hat{\beta}_2 x_{i2} + \dots + \hat{\beta}_k x_{ik})}}$$

where $\hat{\cdot}$ indicates the ML estimate for the parameter.

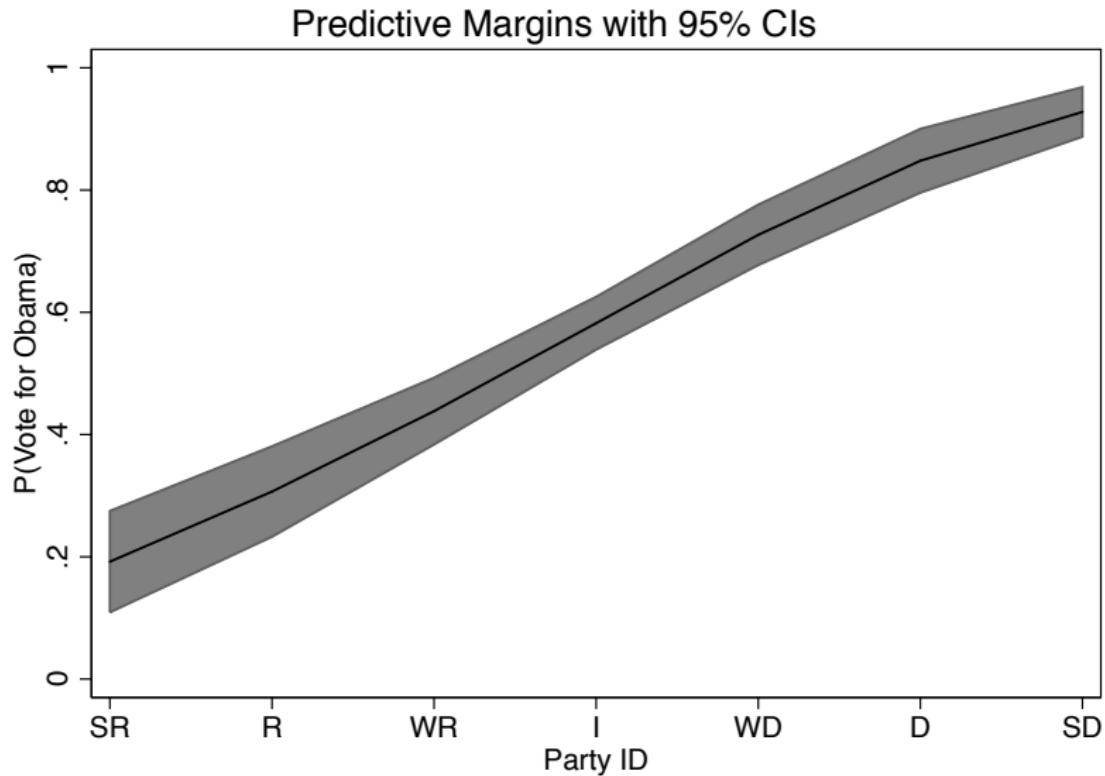
For probit, predicted probability is

$$P(y_i = 1) = \pi_i = \Phi(\hat{\alpha} + \hat{\beta}_1 x_{i1} + \hat{\beta}_2 x_{i2} + \dots + \hat{\beta}_k x_{ik})$$

These probabilities are **different** for every observation, depending on the covariates.

Strategy: plot the predicted probabilities over the x of interest to show what's really going on with the model.

Predicted Probability



Predicted Probability

Plotting predicted probabilities is a **wonderful illustration** of what's really going on with the model results. But they don't directly **test hypotheses**.

For a direct hypothesis test, try one of these approaches:

1. Calculate the predicted probability at two **meaningful and distinct** values of the x of interest. Take the difference in the probabilities.
2. Compute the **marginal change in probability**, which is the first derivative of probability with respect to the x of interest.

Then use the **delta method**, simulation, or another method to compute standard errors. Use the standard errors to compute **p-values and confidence intervals** to test whether the difference/derivative is different from 0.

Marginal Change in Probability

For logit, the **marginal change in probability** (or just **marginal effect**) of x is the derivative of probability:

$$\begin{aligned}\frac{\partial \pi_i}{\partial x_{ik}} &= \frac{\partial}{\partial x_{ik}} \left(\frac{1}{1 + e^{-(\hat{\alpha} + \hat{\beta}_1 x_{i1} + \hat{\beta}_2 x_{i2} + \dots + \hat{\beta}_k x_{ik})}} \right) \\ &= \frac{\hat{\beta}_k e^{-(\hat{\alpha} + \hat{\beta}_1 x_{i1} + \hat{\beta}_2 x_{i2} + \dots + \hat{\beta}_k x_{ik})}}{(1 + e^{-(\hat{\alpha} + \hat{\beta}_1 x_{i1} + \hat{\beta}_2 x_{i2} + \dots + \hat{\beta}_k x_{ik})})^2}.\end{aligned}$$

For probit, the **marginal effect** is

$$\begin{aligned}\frac{\partial}{\partial x_{ik}} P(y_i = 1) &= \frac{\partial}{\partial x_{ik}} \Phi(\hat{\alpha} + \hat{\beta}_1 x_{i1} + \hat{\beta}_2 x_{i2} + \dots + \hat{\beta}_k x_{ik}) \\ &= \hat{\beta}_k \phi(\hat{\alpha} + \hat{\beta}_1 x_{i1} + \hat{\beta}_2 x_{i2} + \dots + \hat{\beta}_k x_{ik})\end{aligned}$$

where $\phi()$ denotes the standard normal PDF instead of the CDF.

Logit's Log-Likelihood Function

Here's the algebraic simplification of the log-likelihood function:

$$\begin{aligned}\ell(\alpha, \beta_1, \dots, \beta_k | X, Y) &= \ln \left[\prod_{i=1}^N \left(\frac{1}{1 + e^{-y_i^*}} \right)^{y_i} \left(1 - \frac{1}{1 + e^{-y_i^*}} \right)^{1-y_i} \right] \\ &= \sum_{i=1}^N \ln \left[\left(\frac{1}{1 + e^{-y_i^*}} \right)^{y_i} \left(1 - \frac{1}{1 + e^{-y_i^*}} \right)^{1-y_i} \right] \\ &= \sum_{i=1}^N \ln \left[\left(\frac{1}{1 + e^{-y_i^*}} \right)^{y_i} \right] + \ln \left[\left(1 - \frac{1}{1 + e^{-y_i^*}} \right)^{1-y_i} \right] \\ &= \sum_{i=1}^N y_i \ln \left[\frac{1}{1 + e^{-y_i^*}} \right] + (1 - y_i) \ln \left[\left(1 - \frac{1}{1 + e^{-y_i^*}} \right) \right]\end{aligned}$$

Logit's Log-Likelihood Function

$$= \sum_{i=1}^N y_i \ln \left[\frac{1}{1 + e^{-y_i^*}} \right] + (1 - y_i) \ln \left[\left(\frac{e^{-y_i^*}}{1 + e^{-y_i^*}} \right) \right]$$

$$= \sum_{i=1}^N -y_i \ln(1 + e^{-y_i^*}) + (1 - y_i) \ln(e^{-y_i^*}) - (1 - y_i) \ln(1 + e^{-y_i^*})$$

$$= \sum_{i=1}^N -y_i \ln(1 + e^{-y_i^*}) - y_i^* (1 - y_i) - \ln(1 + e^{-y_i^*}) + y_i \ln(1 + e^{-y_i^*})$$

$$\ell(\alpha, \beta_1, \dots, \beta_k | X, Y) = \sum_{i=1}^N -y_i^* (1 - y_i) - \ln(1 + e^{-y_i^*})$$