A Detour into Nonlinear Models

. . . for Binary Response Variables

Brenton Kenkel — PSCI 8357
February 25, 2016

Today we are going to briefly look away from linear models and become acquainted with their nonlinear cousins. Specifically, we are going to look at the most common use case for nonlinear models in political science: modeling binary response variables.

Let me be up front about my biases here.[1] I think nonlinear models are wildly overused in political science. OLS is easier to interpret, easier to extend, and more robust than nonlinear estimators—or at least the set of nonlinear estimators typically used in political science. I think these benefits outweigh the potential loss of fidelity due to approximating a nonlinear process with a linear model.

That said, my view is not the majority view among political scientists. If you have a binary response variable, your reviewers will expect you to use logistic regression or one of its cousins. So let us get acquainted.

Why Not a Linear Model?

Suppose we have a binary response $Y_i \in \{0, 1\}$, which we want to model as a function of a vector of p covariates x_i. If we use a linear model for this problem, we are essentially treating the probability that $Y_i = 1$ as a linear function of the covariates:

$$\Pr(Y_i = 1 \mid x_i) = E[Y_i \mid x_i] = x_i^\top \beta.$$

What is wrong with doing it this way?

The first and most serious problem is that the assumption of constant marginal changes in conditional expectation (or constant marginal effects, for you causal-

[1] "Biases" isn't the right word. This is just what I think. If anything, I should be biased in favor of wacky nonlinear models, having worked on them in one of my dissertation chapters and written an R package to estimate a particular class of them.

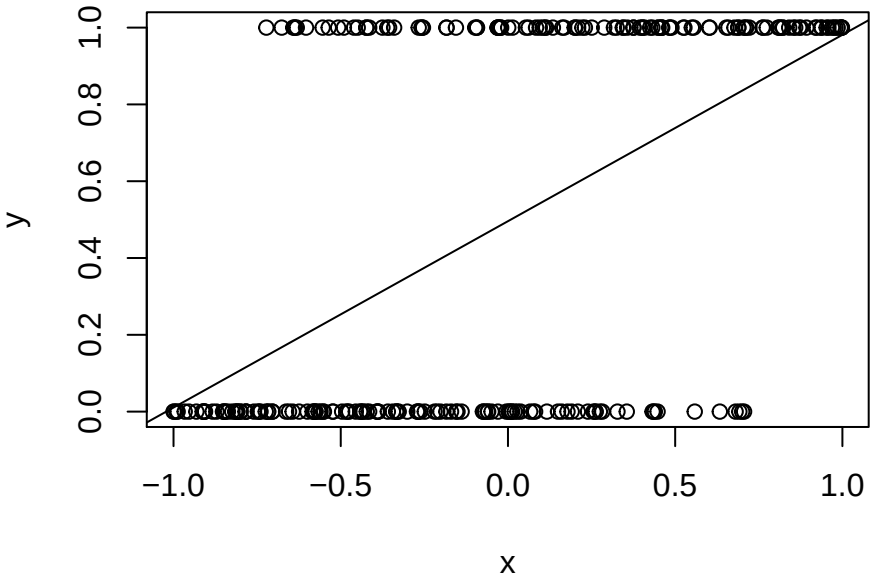
language scofflaws) may be implausible. For example, suppose you want to examine the effect of exposure to a particular advertisement on individuals' probability of voting for a Democrat. You might imagine that for attentive, committed partisans—those whose baseline probability of voting for a Democrat is close to 0 or 1—the ad will have little effect. If the ad has any effect at all, it will probably be greatest for those who would otherwise be close to indifferent.

The second problem is that we might yield nonsense predictions from our model. Suppose we estimate the regression function

$$\Pr(Y_i = 1 \mid x_i) = \hat{\beta}_0 + \hat{\beta}_1 x_{i1},$$

where $\hat{\beta}_1 > 0$. Then we will estimate a nonsensical negative probability for any $x_{i1} < -\hat{\beta}_0/\hat{\beta}_1$ and a nonsensical probability that exceeds 1 for any $x_{i1} > (1 - \hat{\beta}_0)/\hat{\beta}_1$. In practice, though, you'll mostly yield sensible predictions within the range of your data, and nonsense outside of it. You shouldn't be extrapolating outside the range of your data in any case: remember XKCD #605.

The final problem is heteroskedasticity. Let us simulate some data according to a linear probability model, fit the regression line, and look at the residuals.



The residuals are greatest close to the middle, when the estimated probability is close to 0.5. This is a generic feature of linear probability models, which you

can verify by recalling the mathematics of variance. Since $Y_i^2 = Y_i$ for a binary variable, we have

$$\begin{aligned} V[Y_i \mid x_i] &= E[Y_i^2 \mid x_i] - E[Y_i \mid x_i]^2 \\ &= E[Y_i \mid x_i] - E[Y_i \mid x_i]^2 \\ &= x_i^\top \beta (1 - x_i^\top \beta), \end{aligned}$$

which is greatest when $x_i^\top \beta$ is closest to 0.5 and least when $x_i^\top \beta$ is close to 0 or 1.

An Alternative Model

A perhaps more sensible model of the conditional expectation function for a binary response is the *logistic regression model*,

$$\Pr(Y_i = 1 \mid x_i) = \frac{\exp(x_i^\top \beta)}{1 + \exp(x_i^\top \beta)} = \Lambda(x_i^\top \beta).$$

This model has the nice feature that the conditional expectation always lies between 0 and 1, no matter how extreme the covariates. As $x_i^\top \beta \to \infty$, so does $\exp(x_i^\top \beta)$, and $\Pr(Y_i = 1 \mid x_i) \to 1$. Conversely, as $x_i^\top \beta \to -\infty$, then $\exp(x_i^\top \beta) \to 0$, and $\Pr(Y_i = 1 \mid x_i) \to 0$.

We can obtain this model by assuming there is an unobserved *latent response* Y_i^*,

$$Y_i^* = x_i^\top \beta + \epsilon_i,$$

where ϵ_i is drawn from a logistic distribution. If we assume that we observe

$$Y_i = \begin{cases} 0 & Y_i^* < 0 \\ 1 & Y_i^* \geq 0, \end{cases}$$

then we have the logistic regression model. We can obtain a rather similar model by assuming ϵ_i is drawn from a standard normal distribution, in which case

$$\Pr(Y_i = 1 \mid x_i) = \Phi(x_i^\top \beta),$$

where $\Phi(\cdot)$ is the cumulative distribution of the standard normal distribution. We call this the *probit regression model*. I will only focus on the logistic regression model today, but all the techniques and formulas we will derive for logistic regression (or logit) go through for probit regression (or probit), replacing $\Lambda(\cdot)$ and its derivatives with $\Phi(\cdot)$ and its respective derivatives.

Interpretation

We will skip over the details of estimating logistic regression coefficients. The very short version is that we derive the standard estimator using the *method of maximum likelihood*. Since the maximum likelihood estimator cannot be expressed in closed form (i.e., there is no nice equation like for OLS), we use iterative numerical methods to calculate it. From this point forward, take it as given that we have a coefficient estimate $\hat{\beta}$ and a corresponding estimated variance matrix $\hat{\Sigma}$.

In a couple of important ways, logistic regression coefficients are like linear regression coefficients. A positive coefficient on X_j indicates that greater values of X_j are associated with a greater probability of $Y_i = 1$, whereas a negative coefficient indicates that greater values of X_j are associated with a lower probability of $Y_i = 1$. In addition, hypothesis testing for logistic regression coefficients is essentially the same as for linear model coefficients. To test the null hypothesis that $\beta_j = 0$, we divide the estimated coefficient by its estimated standard error and compare to the relevant Z-statistics. To test composite null hypotheses that are linear functions of β, we use Wald tests, just as we did with linear regression.

So if all you care about is rejecting null hypotheses, logistic regression ain't so hard. But you ought to care also about interpreting the model you have estimated, and this is where it gets tricky. As a study in contrast, let us momentarily recall the linear model. Suppose I told you I ran a linear regression on a sample of Democratic primary voters, where the response variable is voting for Bernie Sanders. Suppose moreover I told you I estimated a coefficient of 0.1 on the "Shops at Whole Foods" variable. You could immediately infer what that means: that if we compared two different Democratic voters who were identical except one shopped at Whole Foods and the other didn't, the Whole Foods shopper would be 10% more likely to Feel the Bern.

Not so for logistic regression. Now suppose I told you I ran a logit using the same variables, and that my estimated coefficient on "Shops at Whole Foods" were 1. The only thing you could infer from this information alone is that if we compared two otherwise-identical Democrats, the Whole Foods shopper would be more likely to Feel the Bern. You could not say *how much more* likely. Why the difference? In the linear model (specifically, the standard linear model without higher-order terms), we have constant marginal changes in

conditional expectation:

$$\frac{\partial E[Y_i \mid x_i]}{\partial x_{ij}} = \beta_j.$$

But in the logistic regression model, the marginal change in conditional expectation with respect to a particular variable depends on *all of the covariates and coefficients*:

$$\frac{\partial \Pr(Y_i = 1 \mid x_i)}{\partial x_{ij}} = \beta_j \Lambda(x_i^\top \beta)(1 - \Lambda(x_i^\top \beta)).$$

The estimated MCCE may well be different for each observation in the data, even those with the same value of the j'th covariate, depending on its estimated probability of $Y_i = 1$. Remember that this is a feature, not a bug—one of our motivations for ditching the linear probability model was the implausibility of constant marginal effects.

One way people like to interpret logistic regression results is by calculating *predicted probabilities*. This entails fixing $x_{i,-j}$ at some "central" values (usually means or medians) while varying x_{ij} across a grid $x_{ij}^{(1)}, \ldots, x_{ij}^{(M)}$, each time calculating the "predicted probability"

$$\hat{\Pr}(Y_i = 1 \mid x_i^{(m)}) = \Lambda(x_i^{(m)} \hat{\beta}),$$

where $x_i^{(m)} = (x_{i1}, \ldots, x_{ij}^{(m)}, \ldots, x_{ip})$. Does this give us a representative picture of the effect size? That's the supposed point of fixing $x_{i,-j}$ at central values. But remember that, for a nonlinear function $f(\cdot)$, in general $E[f(x)] \neq f(E[x])$. The predicted effect for an "average observation" is therefore a bad estimate of the population average effect (Hanmer and Kalkan 2012). Not to mention that, depending on the distribution of the covariates, there may be no actual observation that resembles the contrived average.

Our goal now is to get estimates of population average effects (or, less casually causally, changes in conditional expectation) akin to what we would get from a linear model. First imagine a continuous covariate X_j. As we have already seen, the estimated MCCE of X_j for the i'th observation is

$$\frac{\partial \hat{\Pr}(Y_i = 1 \mid x_i)}{\partial x_{ij}} = \hat{\beta}_j \Lambda(x_i^\top \hat{\beta})(1 - \Lambda(x_i^\top \hat{\beta})).$$

The estimated population average MCCE is then

$$\frac{\hat{\beta}_j}{N} \sum_{i=1}^{N} \Lambda(x_i^\top \hat{\beta})(1 - \Lambda(x_i^\top \hat{\beta})).$$

If we can interpret the estimates causally, this is an estimate of the average marginal effect of X_j.

For a binary covariate X_j, it makes more sense to work with first differences than to take derivatives. Let the difference in predicted probability between $X_j = 1$ and $X_j = 0$ for the i'th observation be

$$\hat{\Delta}_{ij} = \Lambda(x_{i,-j}^{\top}\hat{\beta}_{-j} + \hat{\beta}_j) - \Lambda(x_{i,-j}^{\top}\hat{\beta}_{-j}).$$

Then we can estimate the population average first difference as

$$\hat{\Delta}_j = \frac{1}{N}\sum_{i=1}^{N}\hat{\Delta}_{ij}.$$

What this estimates is: if we took an individual at random from the population, then compared their probability of $Y_i = 1$ with $X_j = 1$ and $X_j = 0$, what would the difference be on average?

These formulas let us convert our hard-to-interpret logistic regression results into quantities that we would interpret the same way as the corresponding OLS coefficients. But doing all this work to get there once again raises the question: why not just use a linear probability model in the first place? Admittedly, it is unlikely that the "true" data-generating process is linear, given the considerations I laid out before. But it is also unlikely that the "true" data-generating process is a logistic regression, or a probit regression, or a cloglog or a scobit or any of the other numerous binary response models—we're always making an approximation, and the approximation we choose has trade-offs. Sometimes it is worth sacrificing a bit of fidelity for the sake of interpretability.

Inference

As I already mentioned, inference about coefficients is essentially no different for logistic regression (and friends) than it is for OLS coefficients. Z tests for individual coefficients (we don't have finite-sample distributional results, so no t tests), Wald tests for linear hypotheses about multiple coefficients.

If you want to estimate standard errors or confidence intervals for predicted probabilities, pointwise MCCEs and first differences, or sample average MCCEs

and first differences, the easiest way is the "Clarify" (King, Tomz, and Wittenberg 2000) method. Let $g(\hat{\beta})$ denote your quantity of interest, as a function of the estimated parameters. The procedure is as follows:

1. Draw B values from a multivariate normal distribution with mean $\hat{\beta}_{\text{MLE}}$ and variance $\hat{\Sigma}_{\text{MLE}}$
2. For each $b = 1, \ldots, B$, calculate $g^{(b)} = g(\hat{\beta}^{(b)})$.
3. Estimate the standard error as the standard deviation of $g^{(1)}, \ldots, g^{(b)}$.
4. Estimate confidence intervals as $g(\hat{\beta}_{\text{MLE}}) \pm Z_\alpha \times \text{std.error}$, where Z_α is the Z-score associated with the α significance level.

   Alternatively, estimate the confidence interval by taking the $100(\frac{\alpha}{2})$ and $100(1 - \frac{\alpha}{2})$ percentiles of $g^{(1)}, \ldots, g^{(b)}$.

Summary

- Political scientists usually use logistic or probit regression to model binary response variables.
- The assumptions of these models are better suited to binary data than the linear model, but their parameters are harder to interpret on their own.
- You can go through a lot of work to convert logistic/probit regression coefficients into interpretable quantities of interest, or you can just run a linear probability model.

Appendix: Maximum Likelihood Estimation

Consider the problem of estimating the coefficient vector β from the logistic regression model,

$$\Pr(Y_i = 1 \mid x_i) = \Lambda(x_i^\top \beta).$$

As you know well by now, there are lots of potential estimators of β. Any function of the sample data $(Y, \mathbf{X})$ that yields a $p \times 1$ vector is an estimator of β. The problem is coming up with a *good* estimator.

The usual method of obtaining a good estimator for a nonlinear model is *maximum likelihood*. Loosely speaking, the goal of maximum likelihood estimation is to find the set of parameters under which the probability of observing data like ours is greatest. To do this, we set up the likelihood function: the probability of observing our data, given a particular set of parameters.[2] For example, the likelihood function for logistic regression is

$$\begin{aligned}\ell(\beta \mid Y, \mathbf{X}) &= \prod_{i:Y_i=1} \Lambda(x_i^\top \beta) \prod_{i:Y_i=0} (1 - \Lambda(x_i^\top \beta)) \\ &= \prod_{i=1}^{N} \left[Y_i \Lambda(x_i^\top \beta) + (1 - Y_i)(1 - \Lambda(x_i^\top \beta)) \right]\end{aligned}$$

Any value of β that maximizes $\ell(\beta \mid Y, \mathbf{X})$ also maximizes the natural logarithm of this expression, $\log \ell(\beta \mid Y, \mathbf{X})$. Because sums are easier to compute with than products, we usually work with the *log-likelihood* function,

$$\log \ell(\beta \mid Y, \mathbf{X}) = \sum_{i=1}^{N} \left[Y_i \Lambda(x_i^\top \beta) + (1 - Y_i)(1 - \Lambda(x_i^\top \beta)) \right]$$

The maximum likelihood estimator of the coefficients β of the logistic regression model is the solution to this maximization problem,

$$\hat{\beta}_{\text{MLE}}(Y, \mathbf{X}) = \underset{\beta \in \mathbb{R}^p}{\operatorname{argmax}} \log \ell(\beta \mid Y, \mathbf{X}).$$

Lest ye be concerned about multiplicity, the log-likelihood function is strictly concave and thus has a unique maximizer, so the estimator is well-defined.

Unlike the OLS or WLS estimators of the linear regression parameters, there is no closed-form expression for the MLE estimator of the logistic regression parameters. In practice, we use iterative computational methods to find an approximate solution to the maximization problem above. These methods are beyond the scope of this class, but are implemented via the R function `optim()` and in the **maxLik** package.

Under certain reasonable regularity conditions I won't detail here, maximum likelihood estimators have good asymptotic properties:

[2] With continuous outcomes, the likelihood function is instead the *density* of the observed outcomes, given a particular set of parameters.

- Consistency.
- Asymptotic normality.
- Asymptotic efficiency: no other consistent estimator has lower asymptotic variance.
- Invariance: if $\hat{\beta}$ is a maximum likelihood estimator of β, then $f(\hat{\beta})$ is a maximum likelihood estimator of $f(\beta)$.

The finite-sample properties are generally unknown, though. In particular, maximum likelihood estimators may be biased, though the bias disappears as the sample size grows large enough.

The usual standard error estimates that you get from maximum likelihood procedures are derived from an approximation to the asymptotic variance. This means you should be very wary of the standard error estimates from a maximum likelihood procedure when your sample size is small, even more so when you are estimating many parameters. In situations like this, you can employ *bootstrap* or *profile likelihood* methods for better inferences; we will cover bootstrapping later in the course, but profile likelihood is beyond the scope of Stat II.

Appendix: Implementation

We'll be using the following packages:

- **erer** for automatic estimation of average MCCEs and their standard errors
- **MASS** for drawing from a multivariate normal distribution
- **pscl** for data on vote choices in 1992

```
library("erer")
library("MASS")
library("pscl")
```

Begin by loading up the data. We have a sample of 909 voters, and we are interested in modeling a voter's probability of voting for Ross Perot. We'll need to create a dummy variable for voting for Perot.

```
data(vote92)
head(vote92)
```

```
##      vote dem rep female persfinance natlecon clintondis bushdis
## 1    Bush   0   1      1           1        0     4.0804   0.102
## 2    Bush   0   1      1           0       -1     4.0804   0.102
## 3 Clinton   1   0      1           0       -1     1.0404   1.742
## 4    Bush   0   1      0           0       -1     0.0004   5.382
## 5 Clinton   0   0      1           0       -1     0.9604  11.022
## 6 Clinton   1   0      1          -1       -1     3.9204  18.662
##   perotdis
## 1     0.26
## 2     0.26
## 3     0.24
## 4     2.22
## 5     6.20
## 6    12.18
```

```
vote92$votePerot <- as.numeric(vote92$vote == "Perot")
```

We run logistic regressions in R with the `glm()` command, where GLM stands for *generalized linear model*. The syntax of `glm()` is similar to that of `lm()`, except that you must specify a `family` of models. For binary response models, the relevant family is `binomial()`, specifically `binomial(link = "logit")` for logit models and `binomial(link = "probit")` for probit models. Our running example will be a logit of voting for Perot on three variables:

- Whether the respondent is a Republican
- Whether the respondent is female
- The respondent's squared ideological distance from Perot

```
logit_perot <- glm(votePerot ~ rep + female + perotdis,
                   data = vote92,
                   family = binomial(link = "logit"))
summary(logit_perot)
```

```
##
## Call:
## glm(formula = votePerot ~ rep + female + perotdis, family = binomial(link = "logit"),
##     data = vote92)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -0.803  -0.730  -0.647  -0.481   2.370
```

```
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -1.0353     0.1583   -6.54  6.1e-11
## rep           0.0946     0.1699    0.56   0.5774
## female       -0.3950     0.1703   -2.32   0.0203
## perotdis     -0.1080     0.0380   -2.85   0.0044
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 913.05  on 908  degrees of freedom
## Residual deviance: 896.50  on 905  degrees of freedom
## AIC: 904.5
##
## Number of Fisher Scoring iterations: 4
```

We had no especially good reason to run a logit, so we could also run a probit.

```
probit_perot <- update(logit_perot,
                       family = binomial(link = "probit"))
summary(probit_perot)
```

```
##
## Call:
## glm(formula = votePerot ~ rep + female + perotdis, family = binomial(link = "probit"),
##     data = vote92)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -0.798  -0.733  -0.644  -0.484   2.377
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.6500     0.0920   -7.07  1.6e-12
## rep           0.0588     0.0977    0.60   0.5475
## female       -0.2224     0.0967   -2.30   0.0214
## perotdis     -0.0565     0.0203   -2.78   0.0055
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 913.05  on 908  degrees of freedom
```

```
## Residual deviance: 896.99  on 905  degrees of freedom
## AIC: 905
##
## Number of Fisher Scoring iterations: 4
```

Notice that the coefficients and standard errors are different, but the Z statistics are almost identical. This is typical: it is rare for a logit model and a probit model to give you substantively different results from the same data.

We can calculate the "predicted probabilities" for the data used to fit the model with the `predict()` command. We must specify `type = "response"` to get the predicted probability $\Lambda(x_i^\top \hat{\beta})$; the default just gives us the "linear link" $x_i^\top \hat{\beta}$.

```
pp_obs <- predict(logit_perot, type = "response")
quantile(pp_obs)
```

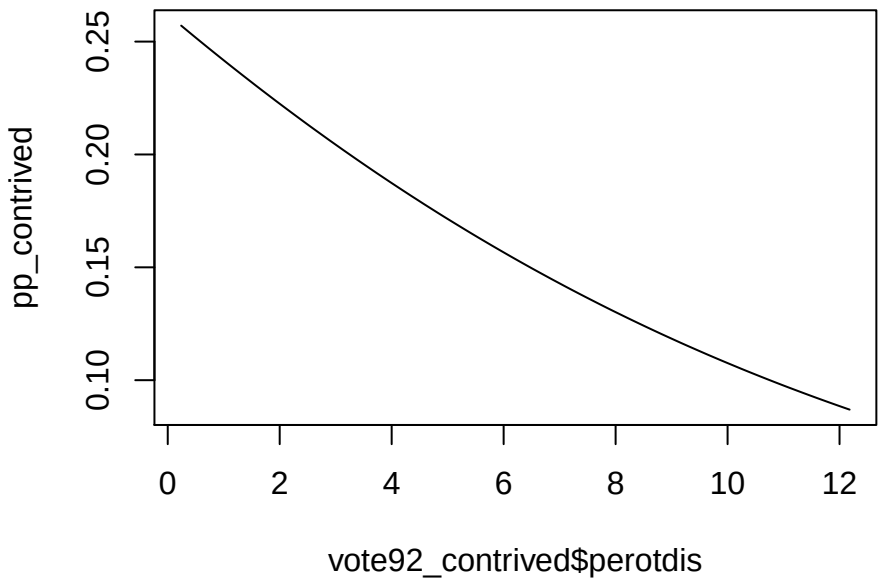
```
##     0%    25%    50%    75%   100%
## 0.0603 0.1650 0.2036 0.2567 0.2756
```

We can also use `predict()` for out-of-sample or hypothetical predictions. To do that, we construct a data frame with the same covariates as we used to fit the model, then feed it to the `newdata` argument of `predict()`. For example, let's do the traditional predicted probability exercise with ideological distance, varying it across its observed range while holding the other covariates at their medians.

```
vote92_contrived <- data.frame(
    perotdis = seq(min(vote92$perotdis),
                   max(vote92$perotdis),
                   length.out = 100),
    rep = median(vote92$rep),
    female = median(vote92$female)
)
pp_contrived <- predict(logit_perot,
                        newdata = vote92_contrived,
                        type = "response")
```

We can now look at how the predicted probability of voting for Perot varies with ideological distance from Perot. Shockingly, those closer to him are more likely to vote for him.

```
plot(vote92_contrived$perotdis,
     pp_contrived,
     type = "l")
```



We can also calculate the MCCE (or marginal effect) of ideological distance for each voter in the dataset. Remember that the MCCE of ideological distance depends on the individual's baseline probability of voting for Perot, being greatest when the baseline probability is closest to 0.5.

```
beta_perotdis <- coef(logit_perot)["perotdis"]
mcce_perotdis <- beta_perotdis * pp_obs * (1 - pp_obs)
range(mcce_perotdis)
```

```
## [1] -0.02157 -0.00612
```

```
mean(mcce_perotdis)
```

```
## [1] -0.0171
```

For binary covariates, we can calculate average first differences. We'll take each observation in the dataset, set `female` to 1 while holding all other covariates fixed, set `female` to 0 while holding all else fixed, and calculate the average difference in predicted probabilities.

```
vote92_male <- vote92_female <- vote92
vote92_male$female <- 0
vote92_female$female <- 1
pp_male <- predict(logit_perot,
                   newdata = vote92_male,
                   type = "response")
pp_female <- predict(logit_perot,
                     newdata = vote92_female,
                     type = "response")
range(pp_female - pp_male)
```

```
## [1] -0.0716 -0.0267
```

```
mean(pp_female - pp_male)
```

```
## [1] -0.062
```

Estimating the standard errors of MCCEs by hand is a bit tougher. We'll take the following steps:

1. Draw B new values of $\hat{\beta}$ from the estimated sampling distribution of β. We've done this before, using `mvrnorm()`.
2. Calculate the sample average MCCE for each value of $\hat{\beta}^{(b)}$. Unfortunately, we can't use the `predict()` function for this part[3] — we'll need to calculate the predicted probabilities by hand.
3. Take the standard deviation across our B estimated sample average MCCEs.

```
## Draw betas from estimated sampling distribution
n_sim <- 100
beta_dist <- mvrnorm(n_sim,
                     mu = coef(logit_perot),
                     Sigma = vcov(logit_perot))

## Construct an n_obs x n_sim matrix whose ij'th entry is the
## predicted probability of observation i under coefficients j
X <- model.matrix(logit_perot)
```

[3] You might be able to trick R by switching out the `$coefficients` element of `logit_perot` with each row of the matrix you draw from `mvrnorm()`, but I wouldn't want to say for sure.

```
Xb <- X %*% t(beta_dist)
Xb <- exp(Xb) / (1 + exp(Xb))
dim(Xb)
```

```
## [1] 909 100
```

```
## Calculate the average of p (1 - p) for each set of coefficients
var_average <- colMeans(Xb * (1 - Xb))

## Multiply with the corresponding coefficient to get the estimated
## average MCCE for each set of coefficients
mcce_dist <- var_average * beta_dist[, "perotdis"]

## Standard error
sd(mcce_dist)
```

```
## [1] 0.00672
```

```
## Confidence interval
mean(mcce_perotdis) + c(-1.96, 1.96) * sd(mcce_dist)
```

```
## [1] -0.0302 -0.0039
```

```
quantile(mcce_dist, probs = c(0.025, 0.975))
```

```
##     2.5%    97.5%
## -0.02945 -0.00287
```

Fortunately, there is a function `maBina()` in the **erer** package that automates the process of calculating average marginal effects and first differences, along with their standard errors. It requires that the original `glm()` be run with the argument `x = TRUE` (which tells `glm()` to save the design matrix $\mathbf{X}$ as part of the output). The argument `x.mean = FALSE` tells it to calculate the observationwise average instead of holding the covariates at their means.

```
logit_perot <- update(logit_perot, x = TRUE)
maBina(logit_perot, x.mean = FALSE)
```

```
##             effect error t.value p.value
## (Intercept) -0.164 0.023  -7.125   0.000
## rep          0.015 0.027   0.555   0.579
## female      -0.062 0.026  -2.347   0.019
```

```
## perotdis    -0.017 0.006  -2.895   0.004
```

`maBina()` automatically knows to compute first differences instead of first derivatives for binary covariates, though there's an argument to turn off this behavior if you reeeeeally want to. Notice that the average MCCE of ideological distance and average first difference of female are the same as what we calculated by hand. However, the estimated standard errors differ—that's because `maBina()` uses an analytical approximation instead of the simulation approach.

NOTE: I have not vetted this package carefully. Its author has not written other packages I am familiar with. I have looked at the code, and I suspect (but am not sure) it will *not* give you the right MCCEs for variables with higher-order terms included. As with any package that is not part of base R, use with caution!

If you just want predicted probabilities and their confidence intervals, the **Zelig** package makes it easy to get them. But you don't want predicted probabilities—you want marginal effects.

References

Hanmer, Michael J., and Kerem Ozan Kalkan. 2012. "Behind the Curve: Clarifying the Best Approach to Calculating Predicted Probabilities and Marginal Effects from Limited Dependent Variable Models." *American Journal of Political Science* 57 (1): 263–77. http://dx.doi.org/10.1111/j.1540-5907.2012.00602.x.

King, Gary, Michael Tomz, and Jason Wittenberg. 2000. "Making the Most of Statistical Analyses: Improving Interpretation and Presentation." *American Journal of Political Science* 44 (2): 341–55. http://gking.harvard.edu/files/making.pdf.