

Distributions in a Regression Context

Ashley I Naimi

Summer 2024

Contents

1	Regression Models and Distributions for Simulation: Simple Continuous	2
2	Regression Models and Distributions for Simulation: Binary and Continuous	4
3	Log-Linear and Poisson	8
4	Marginal Standardization	11
5	Inverse Probability Weighting	13

Often, we use regression models to analyze data, and so they are typically implemented in some way in a simulation setting. Here, we'll explore how to integrate some of the distribution functions in the last chapter into a regression modeling framework. We'll start with some very simple regression modeling frameworks and work our way up in complexity.

1 Regression Models and Distributions for Simulation: Simple Continuous

The simplest regression framework we can simulate involves two normally distributed random variables:

```
set.seed(123)

n = 5000

x <- rnorm(n, mean = 0, sd = 1)

y <- 5 + 2*x + rnorm(n, mean = 0, sd = 1)

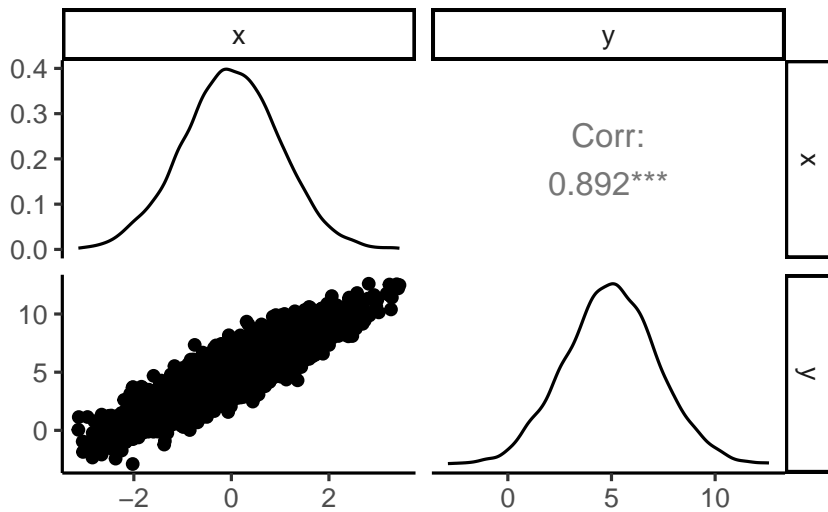
a <- data.frame(x, y)

head(a)
```

```
##           x           y
## 1 -0.56047565 3.384875
## 2 -0.23017749 5.667238
## 3  1.55870831 6.970467
## 4  0.07050839 6.622035
## 5  0.12928774 6.174767
## 6  1.71506499 8.765261
```

We can explore these data using standard methods:

```
GGally::ggpairs(a)
```



```
summary(a$y)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -2.892   3.527   5.007   4.995   6.460   12.599
```

```
summary(a$x)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -3.13739 -0.65514 -0.00757 -0.00057  0.66098  3.44599
```

Because of how we defined the outcome in the simulation, we can analyze these data using a few methods. For example, via `lm()` or `glm()`:

```
mod1 <- lm(y ~ x, data = a)
```

```
# mod1 <- glm(y ~ x, data = a, family = gaussian(link = "identity"))
```

```
summary(mod1)$coefficients
```

```
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.995823  0.01418265  352.249      0
## x            1.993971  0.01426149  139.815      0
```

We can see that the coefficients from this model align almost exactly with the values we used to simulate the outcome variable y .

2 Regression Models and Distributions for Simulation: Binary and Continuous

How should we simulate a binary random variable from a regression model?

We can use a logistic regression model to do this. We won't explain all of this now, but will have an opportunity to look at each of these elements to get a sense of how this code works and why:

```
# define the inverse logistic function
expit <- function(x){
  1/(1 + exp(-x))
}

set.seed(123)

n = 5000

z <- rnorm(n, mean = 0, sd = 1)

x <- rbinom(n, size = 1, p = expit(-1 + log(2)*z))

y <- 100 + 10*x + 3*z + rnorm(n, mean = 0, sd = 10)

# use these variables to construct a dataset:

a <- data.frame(z, x, y)
```

The above code gives us a dataset of 5000 observations with: one continuous covariate z , one binary exposure x , and one continuous outcome y :

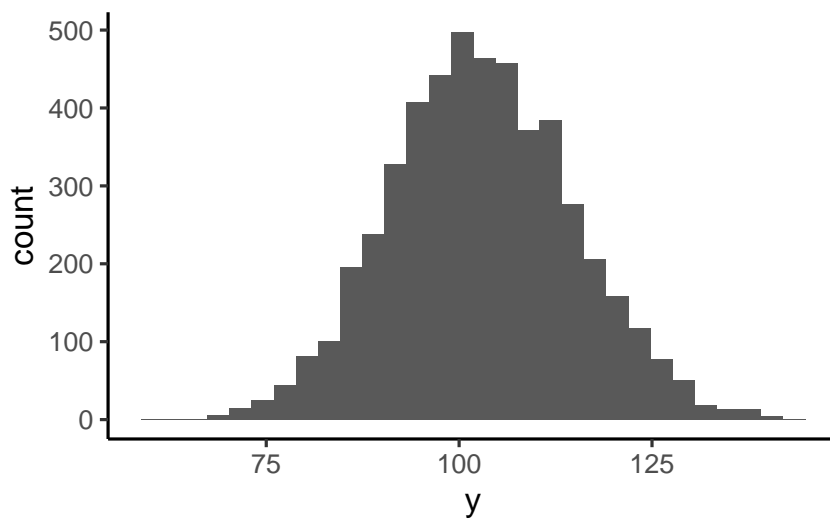
```
head(a)
```

```
##           z x           y
```

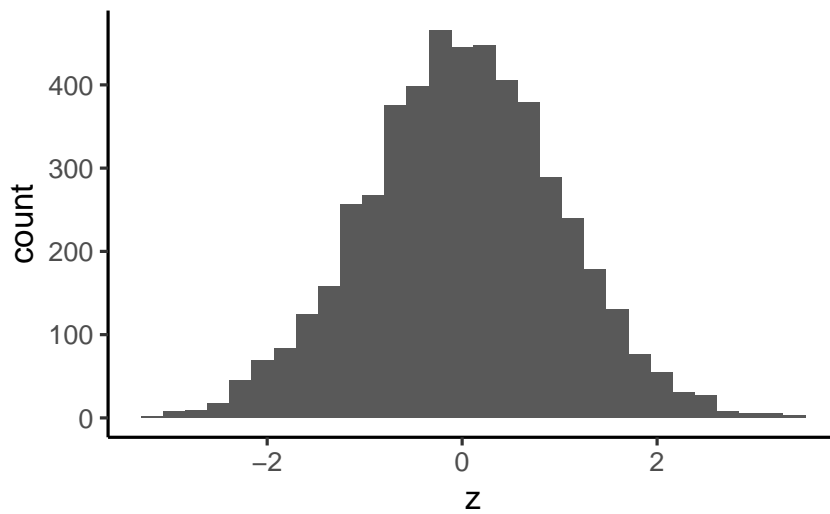
```
## 1 -0.56047565 0 101.81860
## 2 -0.23017749 0 107.45388
## 3  1.55870831 0  99.50946
## 4  0.07050839 0  73.28888
## 5  0.12928774 0  89.41832
## 6  1.71506499 1 102.59044
```

We can do some basic analyses of these data:

```
ggplot(a) + geom_histogram(aes(x = y))
```



```
ggplot(a) + geom_histogram(aes(x = z))
```



```
table(a$x)
```

```
##
##      0      1
## 3517 1483
```

```
mean(a$x)
```

```
## [1] 0.2966
```

Again, we can use a simple regression model to analyze these data, such as:

```
# mod1 <- lm(y ~ x, data = a)

mod1 <- glm(y ~ x + z, data = a, family = gaussian(link = "identity"))

summary(mod1)$coefficients
```

```
##              Estimate Std. Error   t value    Pr(>|t|)
## (Intercept) 99.716726  0.1726271 577.64242 0.000000e+00
## x           10.709244  0.3269832 32.75166 2.603927e-213
## z            2.604083  0.1501827 17.33944 1.888766e-65
```

Let's unpack this model. We can formulate it as a generalized linear model with a Gaussian distribution and identity link function:

$$Y_i = \beta_0 + \beta_1 X_i + \beta_2 Z_i + \epsilon_i$$

where $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$

Notice that this equation corresponds to the following model code:

```
mod1 <- glm(y ~ x + z, data = a, family = gaussian(link = "identity"))

summary(mod1)
```

Notice also how this aligned with the code we used to simulate the outcome:

```
y <- 100 + 10 * x + 3 * z + rnorm(n, mean = 0, sd = 10)
```

What about the code that we used to simulate the exposure? This is the propensity score model, and we used logistic regression, defined as:

$$\text{logit } P(X = 1 \mid Z = 1) = \alpha_0 + \alpha_1 Z$$

which is equivalent to:

$$P(X = 1 \mid Z = 1) = \text{expit}\{\alpha_0 + \alpha_1 Z\}$$

In our code, the values of α_0, α_1 were set to -1 and $\log(2)$, respectively. Importantly, the “logit” and “expit” functions are defined as:

$$\text{logit } P(\bullet) = \frac{P(\bullet)}{1 - P(\bullet)}$$

$$\text{expit}(\bullet) = \frac{1}{[1 + \exp(-\bullet)]}$$

These functions are inversely related, meaning:

$$P(\bullet) = \text{expit}(\text{logit } P(\bullet))$$

We can fit this model using the GLM routines in R. For example, if we wanted to fit a propensity score model to the a data, we might use the following code:

```
a$propensity_score <- glm(x ~ z, data = a, family = binomial("logit"))$fitted.values
```

While this is harder to see, it is aligned with the code we used to simulate the exposure. Specifically:

```
x <- rbinom(n, size = 1, p = expit(-1 + log(2)*z))
```

Note that this code simulates a random variable X from the Bernoulli distribution (`rbinom` with `size = 1`), where p is defined as the inverse of the

logit of the regression model. We can write this as:

$$\log \left[\frac{P(X = 1 | Z)}{1 - P(X = 1 | Z)} \right] = -1 + \log(2) \times Z$$

Or, we can also write this as:

$$P(X = 1 | Z) = \frac{1}{[1 + \exp(-[-1 + \log(2) \times Z])]}$$

This $P(X = 1 | Z)$ is the propensity score. It's what we used in the `p =` argument of the `rbinom` function, and it's what we estimated when we fit a `glm` regressing our exposure against Z , appended with the `$fitted.values` operator.



Deeper Dive: The `expit(•)` Function

Consider the probabilities we get from the `expit` function above when we have specific values of Z , which is a continuous (Gaussian) random variable with mean = 0 and standard deviation = 1. When $Z = 0$, we have:

$$\frac{1}{1 + \exp(-[-1])} \approx \frac{1}{1 + 2.718282} \approx 0.27$$

In contrast, when $Z = -1$, we have

$$\frac{1}{1 + \exp(-[-1 + \log(2) \times -1])} \approx \frac{1}{1 + 0.1839397} \approx 0.15$$

But if $Z = 1$, we have:

$$\frac{1}{1 + \exp(-[-1 + \log(2) \times 1])} \approx \frac{1}{1 + 0.7357589} \approx 0.42$$

Each of these gets resolved in the `rbinom` function above, giving us a probability bounded between $[0,1]$ that also depends on Z .

3 Log-Linear and Poisson

We can also simulate from a log-linear model where the outcome is distributed following a Poisson distribution. In this case, we could define a log-linear Poisson regression model as:

$$\log E(Y | X) = \beta_0 + \beta_1 X$$

where $Y \mid X \sim \text{Pois}(\lambda)$, and where $\lambda = \exp(\beta_0 + \beta_1 X)$. In R, this could look like:

```
set.seed(123)

n = 5000

z <- rnorm(n, mean = 0, sd = 1)

x <- rbinom(n, size = 1, p = expit(-1 + log(2)*z))

y <- rpois(n, lambda = exp(2 + log(2)*x + log(1.5)*z))

# use these variables to construct a dataset:

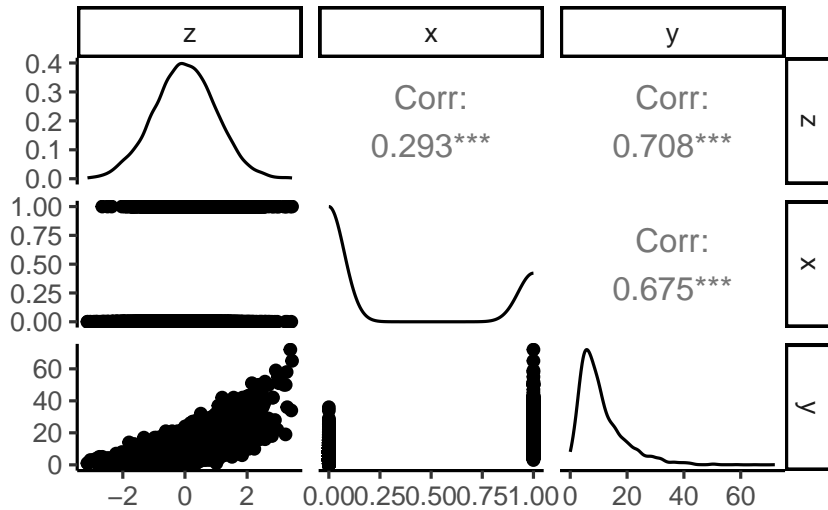
b <- data.frame(z, x, y)

head(b)
```

```
##           z x  y
## 1 -0.56047565 0  7
## 2 -0.23017749 0  3
## 3  1.55870831 0 16
## 4  0.07050839 0  6
## 5  0.12928774 0  8
## 6  1.71506499 1 36
```

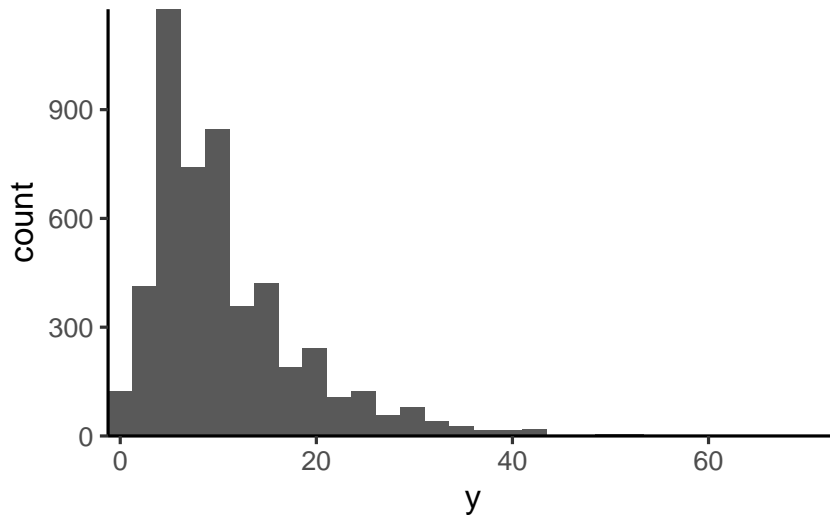
We can explore distributions in this dataset, as we would typically:

```
GGally::ggpairs(b)
```



We could more closely inspect the distribution of the outcome, which follows a Poisson distribution:

```
ggplot(b) +
  geom_histogram(aes(x = y)) +
  scale_x_continuous(expand = c(0,0)) +
  scale_y_continuous(expand = c(0,0))
```



We can then fit a regression model to these simulated data, as follows:

```
mod1 <- glm(y ~ x + z, data = b, family = poisson(link = "log"))

summary(mod1)$coefficients
```

##	Estimate	Std. Error	z value	Pr(> z)
## (Intercept)	1.998889	0.006266520	318.97913	0
## x	0.689279	0.009057425	76.10099	0
## z	0.411776	0.004528943	90.92100	0

Again, this model demonstrates that we can recover the true values, which we used to simulate these data.

4 Marginal Standardization

Marginal standardization is equivalent to g computation (aka the parametric g formula) when the exposure is measured at a single time point (Naimi et al., 2017). This process can be implemented by fitting a single regression model, regressing the outcome against the exposure and all confounding variables. But instead of reading the coefficients the model, one can obtain parameter estimates of interest by using this model to generate predicted risks for each individual under “exposed” and “unexposed” scenarios in the dataset. To obtain standard errors, the entire procedure must be bootstrapped.

Here is some code to implement this marginal standardization in the above dataset:

```
library(boot)

# 'Regress the outcome against the exposure and covariate
ms_model <- glm(y ~ x + z, data = a, family = gaussian(link = "identity"))

## 'Generate predictions for everyone in the sample to obtain
## 'unexposed (mu0 predictions) and exposed (mu1 predictions) risks.
mu1 <- predict(ms_model, newdata = transform(a, x=1), type="response")
mu0 <- predict(ms_model, newdata = transform(a, x=0), type="response")

# 'Mean difference in predicted outcomes
marg_stand_MD <- mean(mu1) - mean(mu0)

# 'Using the bootstrap to obtain confidence intervals for the marginally adjusted
# 'mean difference.
```

```

bootfunc <- function(data,index){
  boot_dat <- data[index,]
  ms_model <- glm(y ~ x + z, data=boot_dat, family = gaussian(link = "identity"))
  mu1 <- predict(ms_model, newdata = transform(boot_dat,x=1), type="response")
  mu0 <- predict(ms_model, newdata = transform(boot_dat,x=0), type="response")

  marg_stand_MD_ <- mean(mu1) - mean(mu0)

  return(marg_stand_MD_)
}

#' Run the boot function. Set a seed to obtain reproducibility
set.seed(123)
boot_res <- boot(a, bootfunc, R=2000)

boot_MD <- boot.ci(boot_res, type = "norm")

marg_stand_MD

```

```
## [1] 10.70924
```

```
boot_MD
```

```

## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 2000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = boot_res, type = "norm")
##
## Intervals :
## Level      Normal
## 95%      (10.07, 11.36 )
## Calculations and Intervals on Original Scale

```

5 Inverse Probability Weighting

We can also estimate the exposure-outcome association using inverse probability weighting. Inverse probability weighting is the commonly employed propensity score adjustment technique. The simple heuristic often used to describe the way IP-weighting works is that, when applied to data, they yield a “pseudo population” where there is no longer an association between the covariate (i.e., confounder) on the exposure.

The weights for each individual needed to create this pseudo-population are defined as the inverse of the probability of receiving their observed exposure. However, simply taking the inverse of the probability of the observed exposure, while valid, is not the usual strategy for implementing inverse probability weights. In practice, one will often use stabilized weights, stabilized normalized weights, potentially with some degree of “truncation” or, more accurately, trimming of the weights.¹

The simplest type of weight used in practice is the stabilized inverse probability weight. These are often defined as:

$$sw = \begin{cases} \frac{P(X=1)}{P(X=1|Z)} & \text{if } X=1 \\ \frac{P(X=0)}{P(X=0|Z)} & \text{if } X=0 \end{cases}$$

Let’s use the simulated data again to construct the stabilized weights and apply them to estimate the mean difference. We start by fitting a propensity score model to construct our weights:

```
# create the propensity score in the dataset
a$propensity_score <- glm(x ~ z, data = a, family = binomial("logit"))$fitted.values

# stabilized inverse probability weights
a$sw <- (mean(a$x)/a$propensity_score)*a$x +
  ((1-mean(a$x))/(1-a$propensity_score))*(1-a$x)

summary(a$sw)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.3638  0.8246  0.9308  1.0002  1.0914  5.3677
```

¹ In contrast to our emphasis of the usage of the word “truncation” which refers to the removal of observations from the dataset, researchers will often refer to “truncating” the weights, which sets the largest value to be equal to the 99th or 95th percentile values. This is more accurately referred to as “trimming” the weights, since no truncation is occurring.

```
head(a)
```

```
##           z x           y propensity_score           sw
## 1 -0.56047565 0 101.81860           0.2054964 0.8853327
## 2 -0.23017749 0 107.45388           0.2463131 0.9332788
## 3  1.55870831 0  99.50946           0.5370583 1.5194138
## 4  0.07050839 0  73.28888           0.2879360 0.9878326
## 5  0.12928774 0  89.41832           0.2965456 0.9999226
## 6  1.71506499 1 102.59044           0.5644486 0.5254686
```

As we can see from the output above, the stabilized weights are, in fact, well behaved, with a mean of one and a max value that is small relative to the overall sample size.

```
mod_MD_weighted <- glm(y ~ x, data = a, weights=sw, family = gaussian("identity"))
```

```
summary(mod_MD_weighted)$coefficients
```

```
##           Estimate Std. Error   t value    Pr(>|t|)
## (Intercept) 99.73286  0.1754109 568.56702 0.000000e+00
## x           10.72026  0.3223223  33.25944 2.693767e-219
```

To get appropriate standard errors for this model, there are a few options we can use. Importantly, the model-based standard errors are no longer valid when weighting is used. One must instead use the robust (sandwich) variance estimators, or the bootstrap.

For example, the robust variance approach could be deployed using the `lmtest` and `sandwich` packages:

```
library(lmtest)
```

```
library(sandwich)
```

```
coeftest(mod_MD_weighted,
         vcov. = vcovHC(mod_MD_weighted, type = "HC3"))[2,]
```

```
##           Estimate   Std. Error   z value    Pr(>|z|)
## 1.072026e+01  3.545264e-01  3.023825e+01 7.447026e-201
```

```
coefci(mod_MD_weighted,
       level = 0.95,
       vcov. = vcovHC(mod_MD_weighted, type = "HC3"))[2,]
```

```
##      2.5 %    97.5 %
## 10.02540 11.41512
```

One can then construct CIs in the standard way using the estimated standard error in the output above, or using the `coefci` function in the `lmtest` package. Alternatively, we can use the bootstrap to get standard errors for IP weighted models. The key to the bootstrap here (as in all cases) is to capture all models within the bootstrap function. In the IP weighting case, this includes the propensity score model and the weighted regression model:

```
## Using the bootstrap to obtain confidence intervals for the IP weighted
## mean difference.
bootfunc <- function(data,index){

  boot_dat <- data[index,]

  boot_dat$propensity_score <- glm(x ~ z, data = boot_dat, family = binomial("logit"))$fitted.values

  # stabilized inverse probability weights
  boot_dat$sw <- (mean(boot_dat$x)/boot_dat$propensity_score)*boot_dat$x +
    ((1-mean(boot_dat$x))/(1-boot_dat$propensity_score))*(1-boot_dat$x)

  mod_MD_weighted_ <- glm(y ~ x, data = boot_dat, weights=sw, family = gaussian("identity"))

  res <- summary(mod_MD_weighted_)$coefficients[2,1]

  return(res)

}

## Run the boot function. Set a seed to obtain reproducibility
```

```

set.seed(123)
boot_res <- boot(a, bootfunc, R = 2000)

boot_IP_weight <- boot.ci(boot_res, type = "norm")

summary(mod_MD_weighted)$coefficients[2,1]

## [1] 10.72026

```

```
boot_IP_weight
```

```

## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 2000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = boot_res, type = "norm")
##
## Intervals :
## Level      Normal
## 95%      (10.05, 11.40 )
## Calculations and Intervals on Original Scale

```

Here's a table comparing the results we've obtained so far:

	Estimate	LCL	UCL
marg_stand_res	10.70924	10.07334	11.35882
ip_weighted_res	10.72026	10.02540	11.41512
ip_weighted_boot	10.72026	10.04892	11.39527

References

Ashley I Naimi, Stephen R Cole, and Edward H Kennedy. An Introduction to G Methods. *Int J Epidemiol*, 46(2):756–62, 2017.