Statistical Modeling: A Tools Approach

Carlisle Rainey

2022-08-24

Week 1

Class agenda.

Goal of the class Make you competent users and consumers (applied and methods papers) of methods beyond least-squares. I'm deliberately avoiding causal-inference methods (matching, DID, etc) because we have a class that covers those specifically that we're offering regularly. I want you to learn a lot about specific tools, but also develop the skills to go and learn more on your own.

We can deviate into any particular topic you'd find helpful.

Structure of the class

We have three sources of information that we'll learn from:

1. *My lectures* I have a set of tools that I want to introduce you to throughout the semester. I think of the lecture as offering "an overview" as well as "my take" on the tool. I will not supply all the details–we don't have enough time and a lecture isn't the ideal medium for deep and subtle ideas. In the past, I have supplied all of my lecture notes to students. However, the research seems clear that student note-taking boosts learning.
2. *Required readings* For each topic, I have a few readings selected to supply further details or offer a different perspective. I want you to carefully read the required readings, even if they seem familiar.
3. *Suggested and other readings* I encourage you to engage readings beyond the required set. These might be "easier" readings (e.g., FPP) or more difficult readings (e.g., Greene). In this category, I want you to use judgement. If the required readings are easy, then I recommend moving on *after* seriously engaging the required readings. If the required readings are too difficult, then seek out gentler introductions. You should NOT pursue the suggested or other readings at the expense of the required readings.

Assessments

This semester, we have a large set of tools that you must demonstrate that you (1) understand and (2) can implement.

1. Exams: We will have regular exams that require you to implement and explain particular tools. I'm open to suggestions on frequency, but I suggest a *weekly*, open-book, take-home exam with about a one hour time limit. I will grade these as pass/fail. You can re-take a (slightly modified) exam up to three times if you fail.
2. Journal: I want to you to journal throughout the semester. I want you to spend *at least* three hours (hopefully more most weeks) outside of class working on your journal. This journal should have several parts:
   a. Class Notes
   b. Review Exercises
   c. Notes from the required readings, including summaries, reactions, and (especially) questions or flags for ideas you didn't understand. This latter is very important–it will make us all better.
   d. Notes from other readings. I want to give you a bit of space to explore things on your own. You could do a deeper dive on ideas covered carefully in the lectures or readings. Or you could pursue a tangential topic (but keep it somewhat related to class). Again, summaries, reactions, and questions are appropriate. I suggest engaging with reading from substantive course with this class in mind, and record your thoughts in your journal.
   e. Connections throughout the semester.
   f. Explorations of ideas for future projects.

As I see it, "regression modeling" in political science is a several-step process:

You begin with a substantive understanding of the way the world works.

1. Choose a regression model. I introduce many.
2. Fit a regression model. Maximum likelihood and Markov chain Monte Carlo methods are powerful and general.
3. Evaluate the fit. What are the properties of the procedure? How well does the model match the data?
4. Interpret the model. I emphasize quantities of interest and confidence intervals, but also discuss hypothesis tests.

You then update your understanding of the world.

This week, I introduce our first "engine": maximum likelihood. As a starting point, we use ML to estimate the parameters of Bernoulli, Poisson, and beta distributions (without covariates). I introduce the parametric bootstrap as a tool to obtain confidence intervals. I introduce the invariance property and show how we can use the invariance property to transform the estimated parameters into other quantities of interest. To evaluate the models, we use the predictive distribution.

Maximum Likelihood

Suppose we have a random sample from a distribution $f(x; \theta)$. We find the maximum likelihood (ML) estimator $\hat{\theta}$ of θ by maximizing the likelihood of the observed data with respect to θ.

In short, we take the likelihood of the data (given the model and a particular θ) and find the parameter θ that maximizes it.

In practice, to make the math and/or computation a bit easier, we manipulate the likelihood function in two ways:

1. Relabel the likelihood function $f(x; \theta) = L(\theta)$, since it's weird to maximize with respect to a "conditioning variable"fixed" variable. (The notation $f(x; \theta)$ suggests x varies for a particular θ.)
2. Work with $\log L(\theta)$ rather than $L(\theta)$. Because log() is a monotonically increasing function, the θ that maximizes $L(\theta)$ also maximizes $\log L(\theta)$.

Suppose we have samples $x_1, x_2, ..., x_N$ from $f(x; \theta)$. Then the joint density/probability is $f(x; \theta) = \prod_{n=1}^{N} f(x_n; \theta)$ and $\log L(\theta) = \sum_{n=1}^{N} \log \left[f(x_n; \theta) \right]$. The ML estimator $\hat{\theta}$ of θ is $\arg\max \log L(\theta)$.

In applied problems, we might be able to simplify $\log L$ substantially. Occasionally, we can find a nice analytical maximum. In many cases, we have a computer find the parameter that maximizes $\log L$.

Example: Bernoulli Distribution

As a running example, we use the **toothpaste cap problem**:

> We have a toothpaste cap–one with a wide bottom and a narrow top. We're going to toss the toothpaste cap. It can either end up lying on its side, its (wide) bottom, or its (narrow) top.
>
> We want to estimate the probability of the toothpaste cap landing on its top.
>
> We can model each toss as a Bernoulli trial, thinking of each toss as a random variable X where $X \sim \text{Bernoulli}(\pi)$. If the cap lands on its top, we think of the outcome as 1. If not, as 0.

Suppose we toss the cap N times and observe k tops. What is the ML estimate $\hat{\pi}$ of π?

According to the model $f(x_i; \pi) = \pi^{x_i}(1-\pi)^{(1-x_i)}$. Because the samples are iid, we can find the *joint* distribution $f(x) = f(x_1) \times ... \times f(x_N) = \prod_{i=1}^{N} f(x_i)$. We're just multiplying k πs (i.e., each of the k ones has probability π) and $(N-k)$ $(1-\pi)$s (i.e., each of the $N-k$ zeros has probability $1-\pi$), so that the $f(x; \pi) = \pi^k (1-\pi)^{(N-k)}$.

$$\text{the likelihood: } f(x; \pi) = \pi^k (1-\pi)^{(N-k)}, \text{where } k = \sum_{n=1}^{N} x_n$$

Then, we relabel.

$$\text{the likelihood: } L(\pi) = \pi^k (1-\pi)^{(N-k)}$$

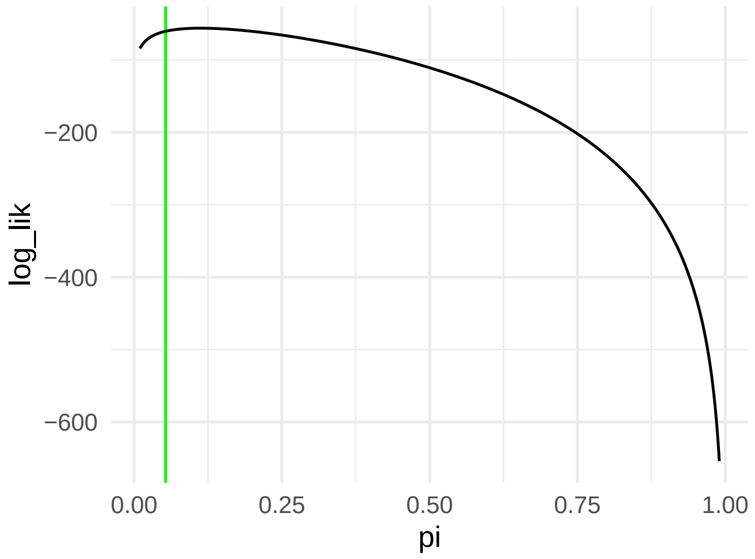
Then, we take the log and simplify.

$$\text{the log-likelihood: } \log L(\pi) = k\log(\pi) + (N-k)\log(1-\pi)$$

To find the ML estimator, we find $\hat{\pi}$ that maximizes $\log L$.

The code below plots the log-likelihood function using the 8/150 data.

```
pi <- seq(0.01, 0.99, length.out = 1000)
data <- tibble(pi = pi) %>%
  mutate(log_lik = 18*log(pi) + (150 - 8)*log(1 - pi))
ggplot(data, aes(x = pi, y = log_lik)) +
  geom_vline(xintercept = 8/150, color = "green") +
  geom_line() +
  theme_minimal()
```



In this case, the analytical optimum is easy.

$$\begin{aligned}
\frac{d\log L}{d\hat{\pi}} = k\left(\frac{1}{\hat{\pi}}\right) + (N-k)\left(\frac{1}{1-\hat{\pi}}\right)(-1) &= 0\\
\frac{k}{\hat{\pi}} - \frac{N-y}{1-\hat{\pi}} &= 0\\
\frac{k}{\hat{\pi}} &= \frac{N-y}{1-\hat{\pi}}\\
k(1-\hat{\pi}) &= (N-y)\hat{\pi}\\
k - y\hat{\pi} &= N\hat{\pi} - y\hat{\pi}\\
k &= N\hat{\pi}\\
\hat{\pi} &= \frac{k}{N} = \text{avg}(x)
\end{aligned}$$

The ML estimator of π is the average of the N Bernoulli trials, or, equivalently, the fraction of successes.

The collected data consist of 150 trials and 8 successes, so the ML estimate of π is $\frac{8}{150} \approx 0.053$.

Example: Poisson Distribution

Suppose we collect N random samples $x = \{x_1, x_2, ..., x_N\}$ and model each draw as a random variable $X \sim \text{Poisson}(\lambda)$. Find the ML estimator of λ.

$$\begin{aligned}
\text{Poisson likelihood: } f(x;\lambda) &= \prod_{n=1}^{N} \frac{\lambda^{x_n} e^{-\lambda}}{x_n!} \\
L(\lambda) &= \prod_{n=1}^{N} \frac{\lambda^{x_n} e^{-\lambda}}{x_n!} \\
\log L(\lambda) &= \sum_{n=1}^{N} \log \left[\frac{\lambda^{x_n} e^{-\lambda}}{x_n!} \right] \\
&= \sum_{n=1}^{N} \left[x_n \log \lambda + (-\lambda) \log e - \log x_n! \right] \\
&= \log \lambda \left[\sum_{n=1}^{N} x_n \right] - N\lambda + \sum_{n=1}^{N} \log(x_n!)
\end{aligned}$$

To find the ML estimator, we find $\hat{\lambda}$ that maximizes $\log L$. In this case, the analytical optimum is easy.

$$\begin{aligned}
\frac{d \log L}{d\hat{\lambda}} = \frac{1}{\hat{\lambda}} \left[\sum_{n=1}^{N} x_n \right] - N &= 0 \\
\frac{1}{\hat{\lambda}} \left[\sum_{n=1}^{N} x_n \right] &= N \\
\left[\sum_{n=1}^{N} x_n \right] &= N\hat{\lambda} \\
\hat{\lambda} = \frac{\sum_{n=1}^{N} x_n}{N} &= \text{avg}(x)
\end{aligned}$$

The ML estimator for the Poisson distribution is just the average of the samples.

Remarks

The ML estimator is extremely common in political science because they are general, fast, and work extremely well. Lots of models that you've heard of, such as logistic regression, are estimated with ML.

We can even obtain ML estimates for the linear regression model. We assume that the observed data are samples from a normal distribution with mean $\mu_n = \alpha + \beta x_n$ and variance σ^2. For this model, the least-squares estimate that we learned earlier is also the ML estimate.

Example: Beta Distribution

Questions:

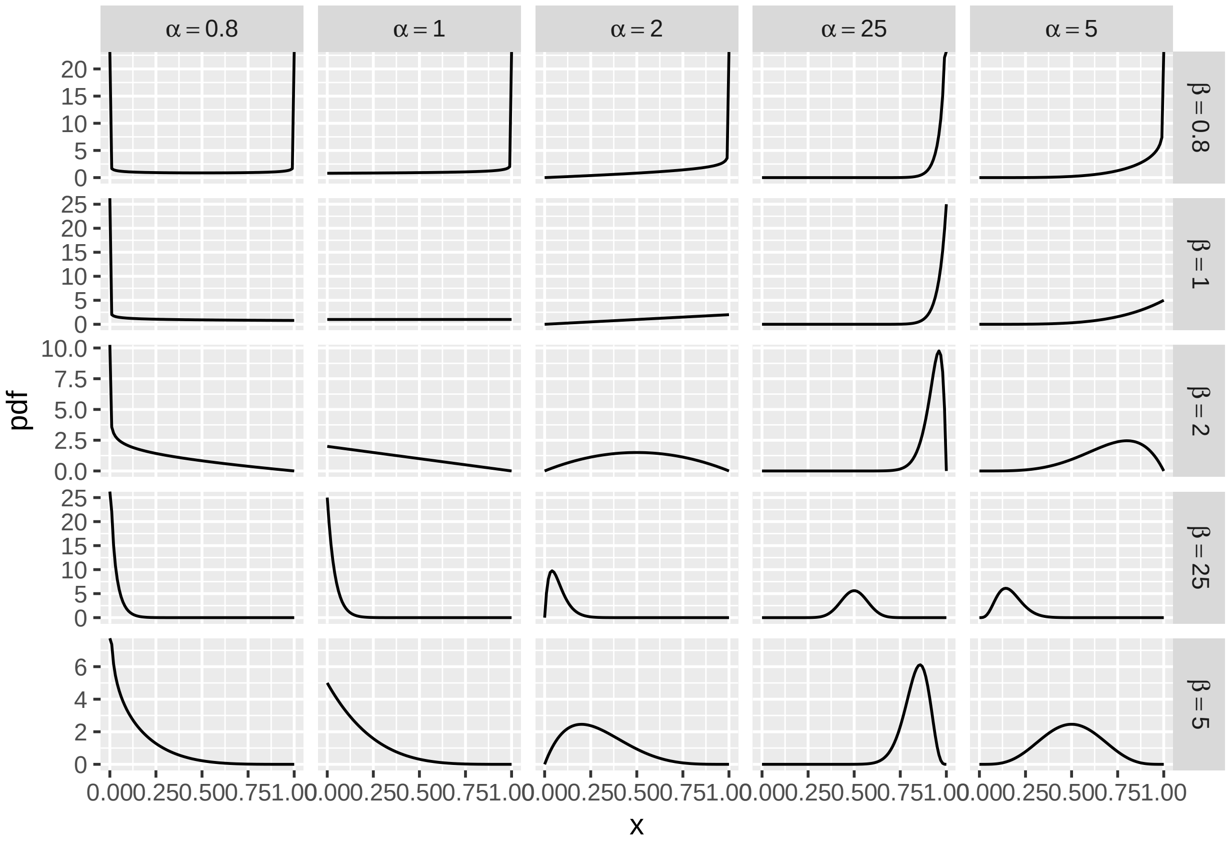
1. What is the *support* of the beta distribution? $[0, 1]$
2. Is y a discrete random variable or a continuous random variable? Continuous.
3. What is the pdf/pmf? $f(y_i; \alpha, \beta) = \dfrac{y_i^{\alpha-1}(1-y_i)^{\beta-1}}{B(\alpha, \beta)}$, where $B(\alpha, \beta) = \displaystyle\int_0^1 t^{\alpha-1}(1-t)^{\beta-1} dt$.

With the beta distribution, we add two complications that typically occur when using ML.

1. multiple parameters
2. an intractable log-likelihood

Start with the probability model $Y_i \sim f(y_i; \theta)$. In the case of the beta model, we have $Y_i \sim \text{beta}(y_i; \alpha, \beta)$. The α and β here don't have a convenient interpretation. They are "shape" parameters. You can think of α as pushing the distribution to the right and β as pushing the distribution to the left.

```r
alphas <- c(0.8, 1, 2, 5, 25)
betas <- c(0.8, 1, 2, 5, 25)

x <- seq(0, 1, length.out = 100)

pdfs <- crossing(alpha = alphas,
                 beta = betas,
                 x = x) %>%
  mutate(pdf = dbeta(x, alpha, beta)) %>%
  mutate(alpha_lbl = paste0("alpha == ", alpha),
         beta_lbl = paste0("beta == ", beta))

ggplot(pdfs, aes(x = x, y = pdf)) +
  facet_grid(rows = vars(beta_lbl), cols = vars(alpha_lbl),
             labeller = "label_parsed", scales = "free") +
  geom_line()
```



We now have two parameters to estimate and we're going to assume that we have multiple observations, so that $y = [y_1, y_2, , ..., y_n]$.

In general, this is how we do ML:

Step 1 Write down the likelihood function. Recall that we can obtain the joint density of y_1 AND y_2 AND ... AND y_n by multiplying the probabilities of each (assuming independence).

$$L(\alpha, \beta) = \prod_{i=1}^{n} \overbrace{f(y_i; \alpha, \beta)}^{\text{density}} = \prod_{i=1}^{n} \frac{y_i^{\alpha-1}(1-y_i)^{\beta-1}}{B(\alpha, \beta)}$$

We see again, as will be usual, that we have this complicated product that will make our lives difficult.

Step 2 Take the log and simplify.

$$\begin{aligned}
L(\alpha,\beta) &= \prod_{i=1}^{n} \frac{y_i^{\alpha-1}(1-y_i)^{\beta-1}}{B(\alpha,\beta)} \\
\log L(\alpha,\beta) &= \sum_{i=1}^{n} \log \frac{y_i^{\alpha-1}(1-y_i)^{\beta-1}}{B(\alpha,\beta)} \\
&= \sum_{i=1}^{n} \left[\log y_i^{\alpha-1} + \log(1-y_i)^{\beta-1} - \log B(\alpha,\beta)\right] \\
&= \sum_{i=1}^{n} \left[(\alpha-1)\log y_i + (\beta-1)\log(1-y_i) - \log B(\alpha,\beta)\right] \\
&= \sum_{i=1}^{n} \left[(\alpha-1)\log y_i + (\beta-1)\log(1-y_i)\right] - n\log B(\alpha,\beta) \\
\log L(\alpha,\beta) &= (\alpha-1)\sum_{i=1}^{n}\log y_i + (\beta-1)\sum_{i=1}^{n}\log(1-y_i) - n\log B(\alpha,\beta)
\end{aligned}$$

Step 3 Maximize

If we wanted, we could work on this one analytically.

1. Take the derivative w.r.t. α.
2. Take the derivative w.r.t. β.
3. Set both equal to zero and solve. (Two equations and two unknowns.)

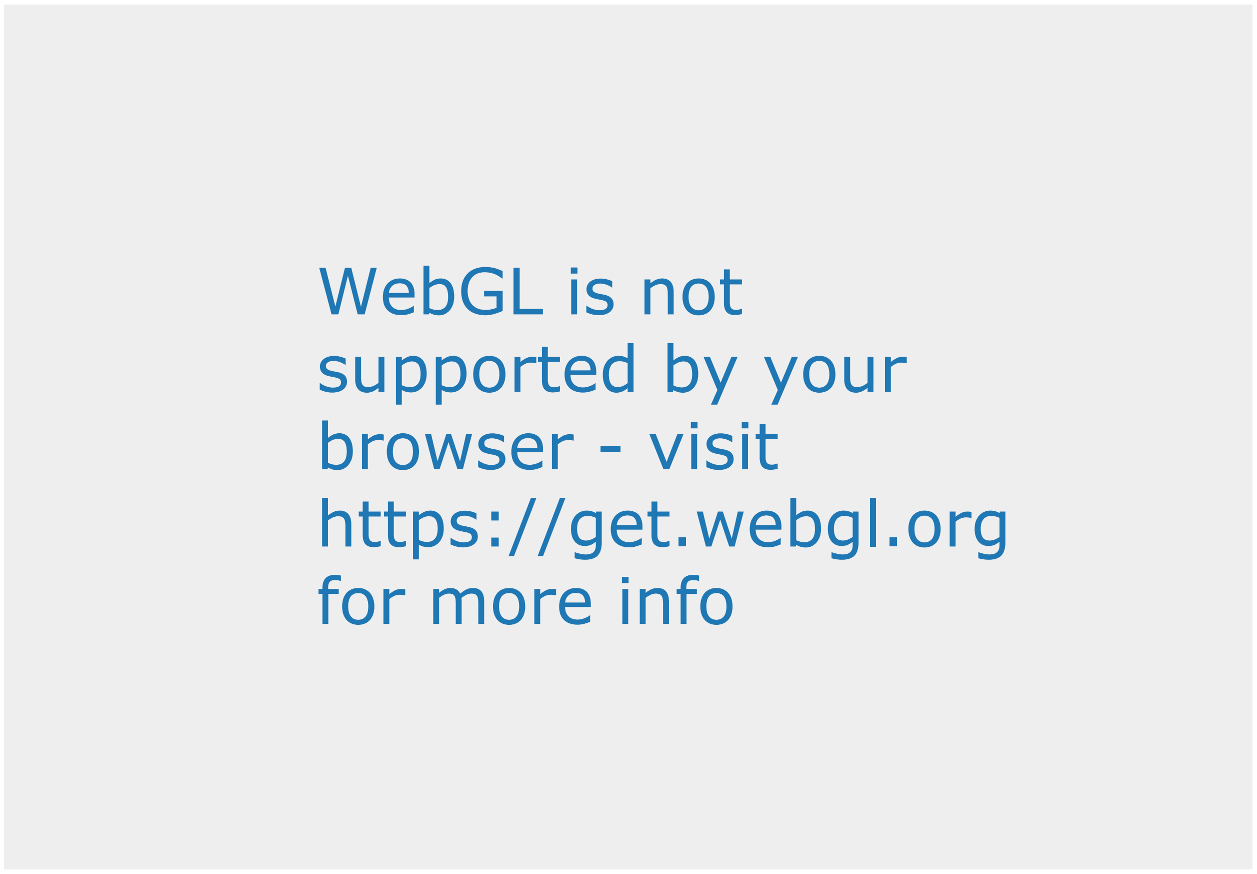
But the last term $B(\alpha,\beta) = \int_0^1 t^{\alpha-1}(1-t)^{\beta-1}dt$ is tricky! So let's do it numerically.

To perform the optimization, we need a data set. For now, let's simulate a fake data set with known parameters

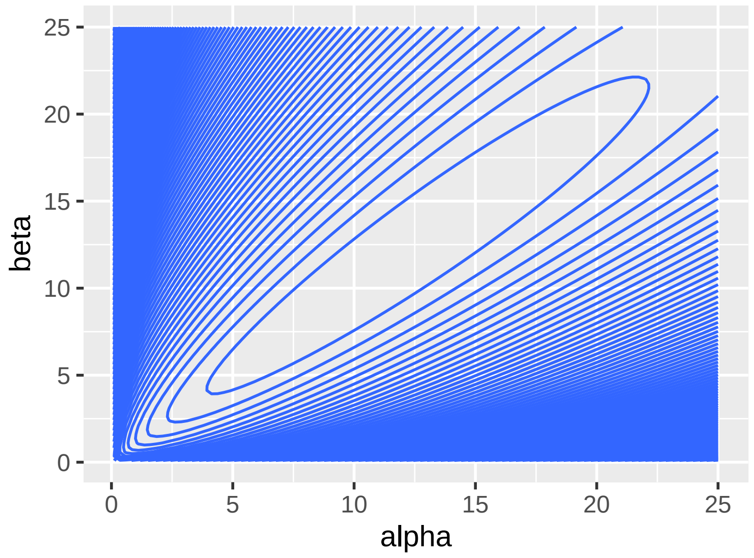
```r
y <- rbeta(1000, shape1 = 10, shape2 = 10)
```

Let's plot the log-likelihood function to see what we're dealing with.

```r
library(plotly)

alpha <- seq(0.1, 25, length.out = 100)
beta  <- seq(0.1, 25, length.out = 100)
data <- crossing(alpha, beta) %>%
  mutate(log_lik = alpha*sum(log(y)) + beta*sum(log(1 - y)) -
           length(y)*log(beta(alpha, beta)))

plot_ly(x = ~alpha, y = ~beta, z = ~log_lik, data = data) %>%
  add_mesh(labels = c("alpha", "beta", "log-likelihood"))
```


WebGL is not supported by your browser - visit https://get.webgl.org for more info

```
ggplot(data, aes(x = alpha, y = beta, z = log_lik)) +
  geom_contour(bins = 100)
```



Now let's program the log-likelihood function in R to handle the optimization numerically.

```
ll_fn <- function(theta, y) {
  alpha <- theta[1]  # optim() requires a single parameter vector
  beta <- theta[2]
  ll <- alpha*sum(log(y)) + beta*sum(log(1 - y)) -
           length(y)*log(beta(alpha, beta))
  return(ll)
}
```

Now let's use `optim()` to do the maximization.

```
est <- optim(par = c(1, 1), fn = ll_fn, y = y,
               control = list(fnscale = -1),
               method = "Nelder-Mead")

print(est$par, digits = 3)
```

```
## [1] 10.5 10.5
```

We can also wrap the `optim()` in a function, to make obtaining the estimates a little bit easier.

```
est_beta <- function(y) {
  est <- optim(par = c(1, 1), fn = ll_fn, y = y,
               control = list(fnscale = -1),
               method = "Nelder-Mead") # for >1d problems
  if (est$convergence != 0) print("Model did not converge!")
  res <- list(est = est$par)
  return(res)
}

ml_est <- est_beta(y)
print(ml_est, digits = 3)
```

```
## $est
## [1] 10.5 10.5
```

The Invariance Property

The parameter π has a nice interpretation–it's a probability or the expected fraction of 1s in the long-run. However, the model parameters might not always have nice interpretation. (See the "shape" parameters of the beta distribution.) Fortunately, it's easy to transform the ML estimates of the model parameters into ML estimates of a quantity of interest.

Suppose we obtain an ML estimate $\hat{\theta}$ of a parameter θ. But we also (or instead) want to estimate a transformation $\tau(\theta)$. The we can estimate $\tau(\theta)$ by applying the transformation τ to the ML estimate $\hat{\theta}$, so that $\widehat{\tau(\theta)} = \hat{\tau} = \tau(\hat{\theta})$.

Example: Bernoulli Odds

Suppose that we want an ML estimator of the *odds* of getting a top for the toothpaste cap problem. We already used ML to estimate the *probability* π of getting a top and came up with $\frac{8}{150} \approx 0.053$. We can directly transform a probability into odds using odds $= \frac{\pi}{1-\pi}$. This has a nice interpretation: odds $= 2$ means that a top is twice as likely as not; odds $= 0.5$ means that a top is half as likely as not.

In our case, we can plug our ML estimate of π into the transformation to obtain the ML estimate of the odds.

$$\begin{aligned}
\widehat{\text{odds}} &= \frac{\hat{\pi}}{1-\hat{\pi}} \\
&= \frac{\frac{8}{150}}{1-\frac{8}{150}} \\
&= \frac{\frac{8}{150}}{\frac{150}{150}-\frac{8}{150}} \\
&= \frac{\frac{8}{150}}{\frac{142}{150}} \\
&= \frac{8}{142} \\
&\approx 0.056
\end{aligned}$$

This means that tops are about 0.06 times as likelihood as not-tops. Inverted, you're about $\frac{142}{8} \approx 18$ times more likely to not get a top than get a top.

Example: Poisson SD

In this example, we use real data from Hultman, Kathman, and Shannon (2013). They are interested in civilian casualties during civil wars. They write:

> To gauge the effectiveness of peacekeeping, we explore all intrastate armed conflicts in sub-Saharan Africa from 1991 to 2008 with monthly observations. Conflicts are identified using the Uppsala Conflict Data Program/Peace Research Institute, Oslo (UCDP/PRIO) Armed Conflict Dataset v.4–2010 (Gleditsch et al. 2002; Harbom and Wallensteen 2009), which employs a threshold of 25 battle deaths per year. The dataset covers 36 conflicts, 12 of which have a PKO present at some time. Consistent with previous research, we add two years of observations to the end of each conflict episode, as the theoretical processes associated with victimization may continue after the cessation of hostilities (Cunningham, Gleditsch, and Salehyan 2009).

Below are a random sample of 250 observations from their 3,972 monthly observations.

```r
civilian_casualties <- c(0, 0, 0, 0, 0, 13, 0, 0, 61, 0, 0, 0, 0, 0, 0, 0, 0,
                         0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 19, 0, 0, 12, 0, 0, 4, 147, 0, 934, 0, 0,
                         42, 0, 24, 124, 0, 1, 0, 0, 0, 145844, 0, 0, 44, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                         0, 0, 2, 10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                         7971, 0, 0, 0, 0, 72, 0, 40, 0, 0, 444, 0, 0, 0, 0, 48, 109, 33, 0, 0, 0, 0,
                         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 15, 41, 0, 0, 0, 0, 84, 0, 34, 0, 0, 0,
                         0, 0, 0, 0, 1, 0, 15, 0, 0, 15, 0, 104, 0, 24, 0, 0, 104, 0, 0, 4, 0, 0, 0, 0,
                         0, 12, 41, 0, 0, 37, 0, 0, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                         0, 0, 12, 0, 4, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 8, 21, 0, 0, 0, 0, 25, 0, 0, 0,
                         3, 0, 0, 27, 0, 0, 576, 3, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, 32, 0, 0, 0, 0,
                         0, 0, 0, 94, 42, 0, 30, 0, 2, 12, 0, 0, 5, 5 )
```

We can estimate a single-parameter Poisson model to estimate a mean λ and a rate $\frac{1}{\lambda}$. In the case of the Poisson model, the ML estimate $l a \hat{m} b d a$ of λ is $\text{avg}(y)$.

```r
ml_est <- mean(civilian_casualties)
print(ml_est, digits = 3)
```

```
## [1] 630
```

The mean is a nice, interpretable parameter, but we might want also want the SD. For the Poisson distribution, the variance equals the mean, so $\text{Var}(y) = \text{E}(y) = \lambda$. Therefore, the SD is $\sqrt{\lambda}$.

```
# ML estimate of SD
ml_est <- sqrt(630)
print(ml_est, digits = 2)
```

```
## [1] 25
```

This is the ML estimate of the SD of the data, and it carries all the properties of ML estimators. We're using the invariance property to move from the mean to the SD by a simple transformation.

Example: Beta Mean and Variance

Now let's see an example of the beta distribution $Y \sim \text{beta}(\alpha, \beta)$. The beta distribution does not have parameters that are easily interpretable in terms of mean and variance. Instead, it has two "shape" parameters α and β that are in tension—one pulls the distribution to the left and the other pulls the distribution to the right.

For this example, I use opinion data from the 50 states from Barrilleaux and Rainey (2014). You can find the data here: https://github.com/carlislerainey/aca-opinion/blob/master/Data/mrp_est.csv

To make these data suitable for the beta distribution, I rescaled the observations from a percent to a proportion that ranges from 0 to 1.

```
br <- tibble::tribble(
  ~state_abbr, ~prop_favorable_aca,
         "AL",   0.382711108911823,
         "AK",   0.374428493677838,
         "AZ",   0.396721609154912,
         "AR",   0.361623814680961,
         "CA",   0.560999240847165,
         "CO",   0.450011650633043,
         "CT",   0.522239143634457,
         "DE",   0.524637037667977,
         "DC",   0.853595690161985,
         "FL",    0.47022917052716,
         "GA",   0.460216990024346,
         "HI",    0.61965456264517,
         "ID",   0.282992730179373,
         "IL",   0.550517975187469,
         "IN",   0.421854785281297,
         "IA",   0.454007062646206,
         "KS",   0.394817640911206,
         "KY",   0.336156662764729,
         "LA",   0.425588396620569,
         "ME",   0.472319257331465,
         "MD",   0.583719023711148,
         "MA",   0.531871146279692,
         "MI",   0.509096426714406,
         "MN",   0.497981331879903,
         "MS",   0.468038078521612,
         "MO",   0.420161837905426,
         "MT",   0.351773944902139,
         "NE",   0.365225584190989,
         "NV",   0.459026605256376,
         "NH",    0.43886275738451,
         "NJ",   0.531656835425683,
         "NM",   0.528461049175538,
         "NY",     0.6010574821094,
         "NC",   0.452240849305449,
         "ND",   0.367690453757597,
         "OH",   0.456298880813516,
         "OK",   0.309578750918355,
         "OR",   0.455832591683007,
         "PA",    0.45819440292365,
         "RI",   0.536978574569609,
         "SC",   0.444870259057071,
         "SD",   0.377170366708612,
         "TN",   0.368615233253355,
         "TX",   0.428407014559672,
         "UT",   0.248496577141183,
         "VT",   0.553042362822573,
         "VA",   0.470739058046787,
         "WA",   0.496133477680592,
         "WV",   0.295062675817918,
         "WI",   0.489912969415965,
         "WY",   0.263567780036879
  )
```

Now let's find the ML estimates of the two shape parameters of the beta distribution.

```
# obtain ml estimates
log_lik_fn <- function(par = c(2, 2), y) {
  a <- par[1]  # pulling these out makes the code a bit easier to follow
  b <- par[2]
  log_lik_i <- dbeta(y, shape1 = a, shape2 = b, log = TRUE)
  log_lik <- sum(log_lik_i)
  return(log_lik)
}
opt <- optim(par = c(2, 2), fn = log_lik_fn, y = br$prop_favorable_aca,
             control = list(fnscale = -1))
ml_est <- opt$par
print(ml_est, digits = 3)
```

```
## [1]  9.56 11.49
```

The mean is given by $\frac{\alpha}{\alpha+\beta}$ and the variance is given by $\frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$.

We can use the invariance property to obtain ML estimates of the mean and variance using our ML estimates of α and β.

```
a <- ml_est[1]
b <- ml_est[2]

a/(a + b)  # mean
```

```
## [1] 0.4542508
```

```
(a * b)/((a + b)^2 * (a + b + 1))  # var
```

```
## [1] 0.01123986
```

It's worth noting that these correspond closely, *but not exactly* to the observed mean and variance.

```
mean(br$prop_favorable_aca)
```

```
## [1] 0.4524527
```

```
var(br$prop_favorable_aca)
```

```
## [1] 0.01073633
```

The Parametric Bootstrap

The parametric bootstrap is a powerful, general tool to obtain confidence intervals for estimates from parametric models.

Importantly, we are going to lean *pretty heavily* on the assumption that we have a good model of the distribution of the data. (The predictive distribution below allows us to assess this.) There's also a ***non****parametric* bootstrap, which is much more popular. We consider that later in the semester.

Suppose we have a sample y from some known distribution $f(y;\theta)$ and use y to estimate the model parameter(s) θ or some quantity of interest $\tau(\theta)$. Remember, we can use ML to estimate either.

To compute a confidence interval, we can use a *parametric* bootstrap. To do implement the parametric bootstrap, do the following:

1. Approximate $f(y;\theta)$ with $\hat{f} = f(y;\hat{\theta})$. Simulate a new outcome y^{bs} from the estimated distribution.
2. Re-compute the estimate of interest $\hat{\theta}^{\text{bs}}$ or $\hat{\tau}^{\text{bs}}$ using the bootstrapped outcome variable y^{bs} rather than the observed outcome y.
3. Repeat 1 and 2 many times (say 2,000) to obtain many bootstrapped estimates. To obtain the 95% confidence interval, take the 2.5th and 97.5th percentiles of the estimates. This is known as the percentile method.

Example: Toothpaste Cap Problm

The code below implements the parametric bootstrap for the toothpaste cap problem. For 2,000 iterations, it draws 150 observations from $Y \sim \text{Bernoulli}(\hat{\pi} = \frac{8}{150})$. For each iteration, it computes the ML estimate of π for the bootstrapped data set. Then it computes the percentiles to obtain the confidence interval.

```
n_bs <- 2000
bs_est <- numeric(n_bs)  # a container for the estimates
for (i in 1:n_bs) {
  bs_y <- rbinom(150, size = 1, prob = 8/150)
  bs_est[i] <- mean(bs_y)
}
print(quantile(bs_est, probs = c(0.025, 0.975)), digits = 2)  # 95% ci
```

```
##  2.5% 97.5%
## 0.020 0.093
```

We leave an evaluation of this confidence interval (i.e., Does it capture θ 95% of the time?) to later in the semester.

Example: Beta Distribution

Now let's apply the parametric bootrap to a two-parameter model: the beta distribution.

First, let's simulate a (fake) data set to use.

```
# set parameters
alpha <- 5
beta <- 2

# simulate data
set.seed(1234)
n <- 100
y <- rbeta(n, alpha, beta)
```

Now let's find the ML estimates of the two shape parameters.

```
# obtain ml estimates
log_lik_fn <- function(par = c(2, 2), y) {
  a <- par[1]  # pulling these out makes the code a bit easier to follow
  b <- par[2]
  log_lik_i <- dbeta(y, shape1 = a, shape2 = b, log = TRUE)
  log_lik <- sum(log_lik_i)
  return(log_lik)
}
opt <- optim(par = c(3, 3), fn = log_lik_fn, y = y,
             control = list(fnscale = -1))
ml_est <- opt$par
print(ml_est, digits = 3)
```

```
## [1] 5.46 1.91
```

Now let's use those ML estimates to perform a parametric bootstrap and find 95% CIs for the shape parameters.

```
# obtain parametric bootstrap 95% ci for alpha and beta
n_bs <- 2000
bs_est <- matrix(NA, nrow = n_bs, ncol = 2)  # a container for the estimates
for (i in 1:n_bs) {
  bs_y <- rbeta(n, shape1 = ml_est[1], shape2 = ml_est[2])
  bs_opt <- optim(par = c(3, 3), fn = log_lik_fn, y = bs_y,
             control = list(fnscale = -1))
  bs_est[i, ] <- bs_opt$par
}
ci <- apply(bs_est, MARGIN = 2, quantile, probs = c(0.025, 0.975))
print(ci, digits = 3)  # 95% ci
```

```
##       [,1] [,2]
## 2.5%  4.25 1.52
## 97.5% 7.52 2.58
```

If instead we cared about the mean of the beta distribution (which is $\frac{\alpha}{\alpha+\beta}$), we can use the parametric bootstrap to obtain a confidence interval for that quantity as well.

```
# obtain parametric bootstrap 95% ci for mean
n_bs <- 2000
bs_est <- numeric(n_bs)  # a container for the estimates
for (i in 1:n_bs) {
  bs_y <- rbeta(n, shape1 = ml_est[1], shape2 = ml_est[2])
  bs_opt <- optim(par = c(3, 3), fn = log_lik_fn, y = bs_y,
             control = list(fnscale = -1))
  bs_alpha <- bs_opt$par[1]
  bs_beta <- bs_opt$par[2]
  bs_est[i] <- bs_alpha/(bs_alpha + bs_beta)
}
print(quantile(bs_est, probs = c(0.025, 0.975)), digits = 2)  # 95% ci
```

```
##  2.5% 97.5%
##  0.71  0.77
```

```
# true mean
print(alpha/(alpha + beta), digits = 2)
```

```
## [1] 0.71
```

Sampling Distribution

What's the most important concept in statistical inference? I don't know, but it could be **the sampling distribution**. For effect, let me back off the hedge.

> The most important concept in statistical inference is the **sampling distribution**.

To define a sampling distribution, you need to imagine repeating a study over and over. If each study has a random component (perhaps random sampling or random assignment to treatment and control), then the estimate will differ from study to study. The distribution of the estimates across the studies is called the sampling distribution.

Example: The Toothpaste Cap Problem

For a given sample of 150 tosses, we recognize the the ML estimate $\hat{\pi}$ does not (usually) exactly equal the parameter π. Instead, the particular $\hat{\pi}$ that the study produces is draw from a distribution.

Let's illustrate that with a simulation. For these simulations, I suppose that we toss the toothpaste cap 150 times and the chance of a head is 5%.

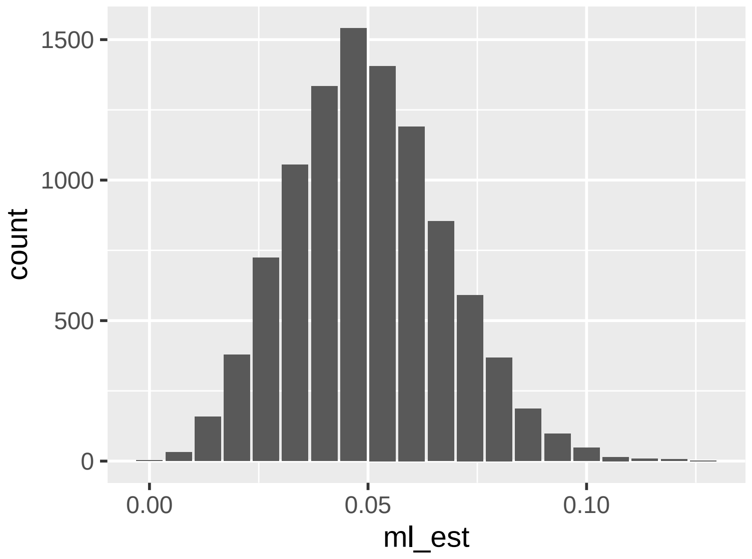
```r
n_sims <- 10
ml_est <- numeric(n_sims)  # a container for the estimates
for (i in 1:n_sims) {
  y <- rbinom(150, size = 1, prob = 0.05)
  ml_est[i] <- mean(y)
}
print(ml_est, digits = 2)
```

```
##  [1] 0.027 0.060 0.060 0.053 0.073 0.093 0.040 0.053 0.027 0.033
```

As you can see, the ML estimates vary to from sample to sample–different data sets produce different ML estimates. We need a way to create a confidence interval that consistently captures θ.

If we repeat the simulations a large number of times, we can see an accuracy picture of the sampling distribution via histogram.

```r
n_sims <- 10000
ml_est <- numeric(n_sims)  # a container for the estimates
for (i in 1:n_sims) {
  y <- rbinom(150, size = 1, prob = 0.05)
  ml_est[i] <- mean(y)
}

gg_data <- data.frame(ml_est = ml_est)
ggplot(gg_data, aes(x = ml_est)) +
  geom_bar()
```



Many of our methods of evaluating an estimator are statements about the sampling distribution of that estimator. In general, we'd like the sampling distribution to be centered over the true parameter of interest and tightly dispersed.

Bias

Imagine repeatedly sampling and computing the estimate $\hat{\theta}$ of the parameter θ for each sample. In this thought experiment, $\hat{\theta}$ is a random variable. We say that $\hat{\theta}$ is **biased** if $E(\hat{\theta}) \neq \theta$. We say that $\hat{\theta}$ is **unbiased** if $E(\hat{\theta}) = \theta$. We say that the **bias** of $\hat{\theta}$ is $E(\hat{\theta}) - \theta$.

Importantly, **ML estimators are not necessarily unbiased**. Of the models we will see in this course, *most* are biased.

Example: Bernoulli Distribution

For example, we can compute the bias of our ML estimator of π in the toothpaste cap problem.

$$E\left[\frac{k}{N}\right] = \frac{1}{N}E(k) = \frac{1}{N}E\overbrace{\left(\sum_{n=1}^{N} x_n\right)}^{\text{recall } k=\sum_{n=1}^{N} x_n} = \frac{1}{N}\sum_{n=1}^{N} E(x_n) = \frac{1}{N}\sum_{n=1}^{N} \pi = \frac{1}{N}N\pi$$
$$= \pi$$

Thus, $\hat{\pi}^{ML}$ is an unbiased estimator of π in the toothpaste cap problem.

We can use a Monte Carlo simulation to check this analytical result.

```r
set.seed(1234)
n_mc_sims <- 100000
pi_hat <- numeric(n_mc_sims)
for (i in 1:n_mc_sims) {
  y <- rbinom(150, size = 1, prob = 0.05)
  pi_hat[i] <- mean(y)
}

# expected value of pi-hat
mean(pi_hat)
```

```
## [1] 0.05006227
```

```
# estimated monte carlo error
sd(pi_hat)/sqrt(n_mc_sims)
```

```
## [1] 5.631271e-05
```

But notice that the property of unbiasedness does not follow the estimate through transformation. Because the sample is relatively large in this case (150 tosses), the bias is small, but detectable with 100,000 Monte Carlo simulations

```
odds_hat <- pi_hat/(1 - pi_hat)

# actual odds
0.05/(1 - 0.05)
```

```
## [1] 0.05263158
```

```
# expected value of odds-hat
mean(odds_hat)
```

```
## [1] 0.05307323
```

```
# estimated monte carlo error
sd(odds_hat)/sqrt(n_mc_sims)
```

```
## [1] 6.288517e-05
```

```
# the z-statistic
(mean(odds_hat) - 0.05/0.95)/(sd(odds_hat)/sqrt(n_mc_sims))
```

```
## [1] 7.023072
```

Example: Poisson Distribution

Using math almost identical to the toothpaste cap problem, we can show that the ML estimator $\hat{\lambda} = \text{avg}(x)$ is an unbiased estimator of λ.

We can also illustrate the unbiasedness with a computer simulation.

```
lambda <- 4.0      # the parameter we're trying to estimate
sample_size <- 10  # the sample size we're using in each "study"

n_mc_sims <- 10000  # the number of times we repeat the "study"
lambda_hat <- numeric(n_mc_sims)  # a container
for (i in 1:n_mc_sims) {
  x <- rpois(sample_size, lambda = lambda)
  lambda_hat[i] <- mean(x)
}

# expected value of lambda-hat
mean(lambda_hat)
```

```
## [1] 3.99397
```

```
# estimated monte carlo error
sd(lambda_hat)/sqrt(n_mc_sims)
```

```
## [1] 0.006300177
```

Consistency

Imagine taking a sample of size N and computing the estimate $\hat{\theta}_N$ of the parameter θ. We say that $\hat{\theta}$ is a **consistent** estimator of θ if $\hat{\theta}$ converges in probability to θ.

Intuitively, this means the following:

1. For a large enough sample, the estimator returns the exact right answer.
2. For a large enough sample, the estimate $\hat{\theta}$ does not vary any more, but collapses onto a single point and that point is θ.

Under weak, but somewhat technical, assumptions that usually hold, ML estimators are consistent.

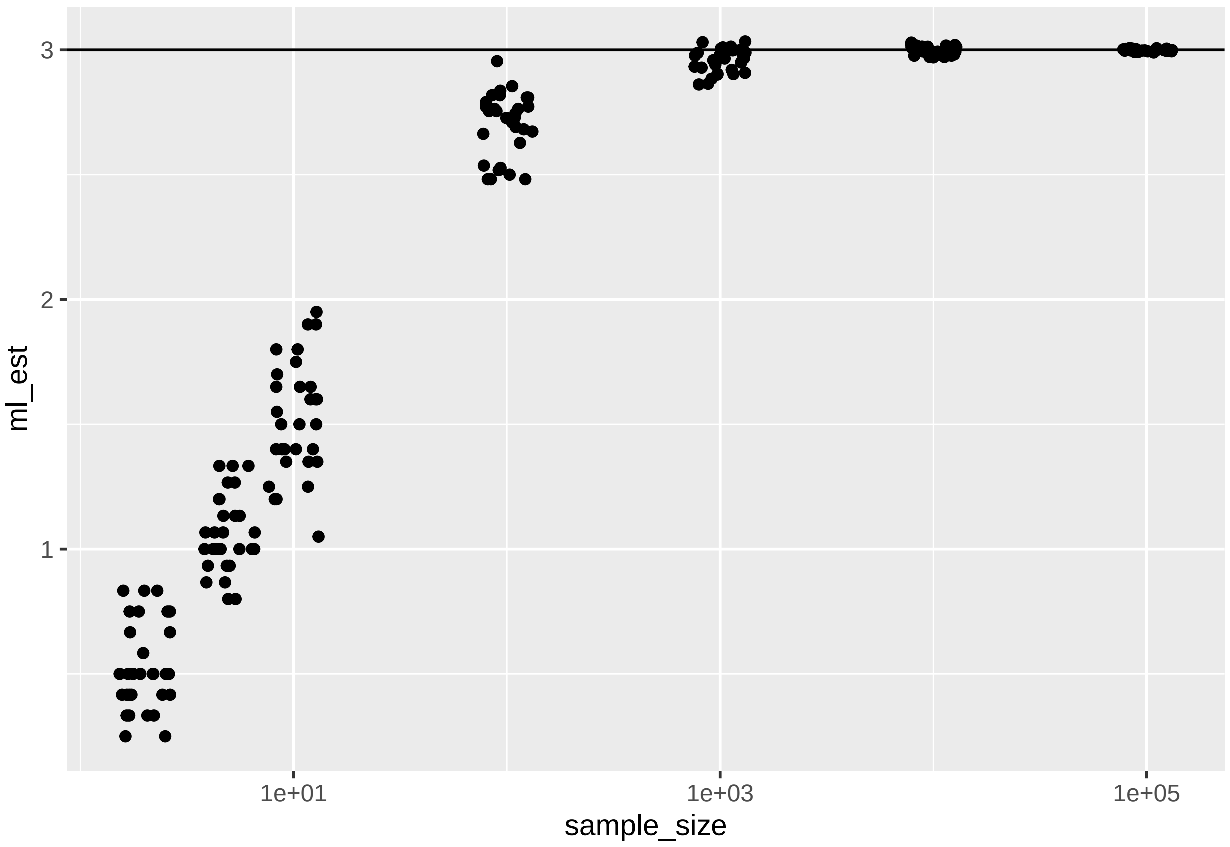
Given that we always have finite samples, why is consistency valuable? In short, it's not valuable, directly. However, consistent estimators tend to be decent with small samples.

But it does not follow that consistent estimators work well in small samples. However, as a rough guideline, consistent estimators work well for small samples. However, whether they actually work well in any particular situation needs a more careful investigation.

Example: Illustrative

To illustrate the concept of consistency, consider this estimator of the population mean $\hat{\mu}^{\text{silly}} = \frac{\sum_{i=1}^{N} x_i}{N+10}$. While this estimator is biased, it is a consistent estimator.

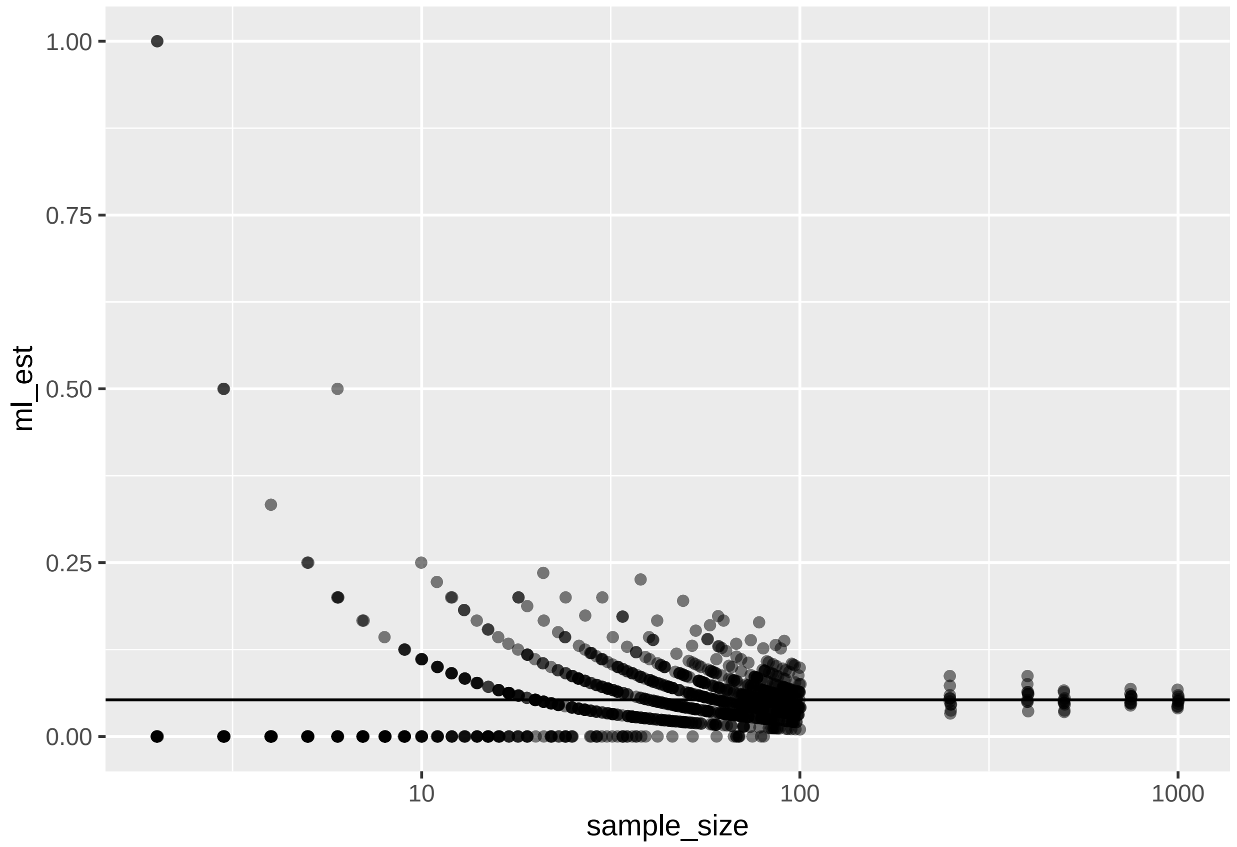
```
population <- c(1, 2, 3, 4, 5)
sample_sizes <- c(2, 5, 10, 100, 1000, 10000, 100000)
n_mc_sims <- 30  # for each sample size
results_list <- list()  # grow with each iteration; slow, but easy
for (i in 1:length(sample_sizes)) {
  ml_est_i <- numeric(n_mc_sims)
  for (j in 1:n_mc_sims) {
    x <- sample(population, sample_sizes[i], replace = TRUE)
    ml_est_i[j] <- sum(x)/(sample_sizes[i] + 10)
  }
  results_list[[i]] <- data.frame(sample_size = sample_sizes[i],
                                  ml_est = ml_est_i)
}
results <- dplyr::bind_rows(results_list)

ggplot(results, aes(x = sample_size, y = ml_est)) +
  geom_hline(yintercept = mean(population)) +
  geom_jitter() +
  scale_x_log10()
```



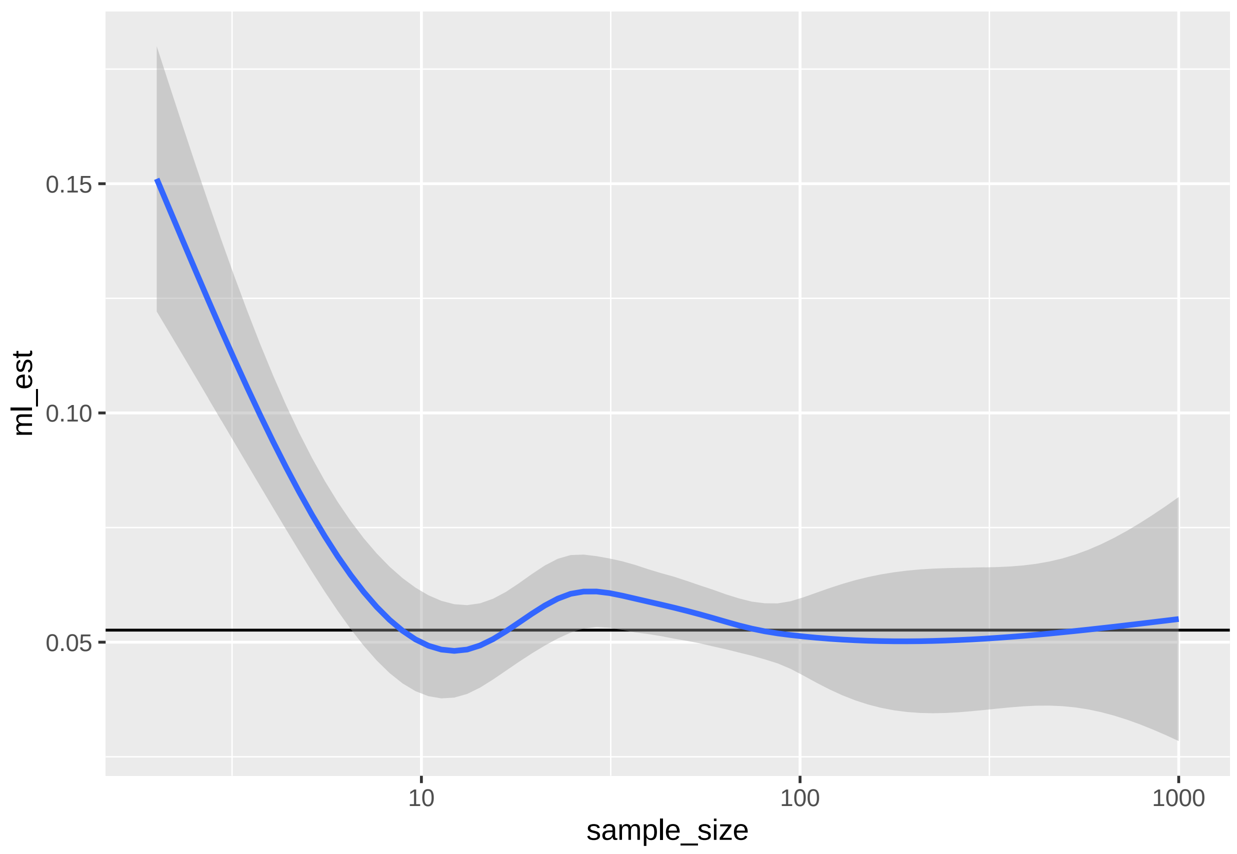
Example: Bernoulli Odds

There are two ways to see consistency for the Bernoulli. First, unless our sample size is a multiple of 20, it is impossible to obtain an estimated odds of 0.05/(1 - 0.05). Second, in small samples, the ML estimate of the odds is biased. As the sample size increases, the bias shrinks and the estimates collapse toward (and eventually onto) the true value.

```r
sample_sizes <- c(2:100, 250, 400, 500, 750, 1000)
n_mc_sims <- 10  # for each sample size
results_list <- list()  # grow with each iteration; slow, but easy
for (i in 1:length(sample_sizes)) {
  ml_est_i <- numeric(n_mc_sims)
  for (j in 1:n_mc_sims) {
    x <- rbinom(sample_sizes[i], 1, prob = 0.05)
    pi_hat <- mean(x)
    ml_est_i[j] <- pi_hat/(1 - pi_hat)
  }
  results_list[[i]] <- data.frame(sample_size = sample_sizes[i],
                                  ml_est = ml_est_i)
}
results <- dplyr::bind_rows(results_list)

ggplot(results, aes(x = sample_size, y = ml_est)) +
  geom_hline(yintercept = 0.05/(1 - 0.05)) +
  geom_jitter(alpha = 0.5, shape = 19) +
  scale_x_log10()
```



```
ggplot(results, aes(x = sample_size, y = ml_est)) +
  geom_hline(yintercept = 0.05/(1 - 0.05)) +
  scale_x_log10() +
  geom_smooth()
```



Predictive Distribution

In Bayesian statistics, a popular tool for model evaluation is the posterior predictive distribution. But we might use an analogous approach for models fit with maximum likelihood.

The predictive distribution is just the distribution given the ML estimates. Using our notation above, the predictive distribution is $f(y; \hat{\theta})$.

When you perform a parametric bootstrap, you are resampling from this predictive distribution. Here, we're going to use it for a different purpose: to understand and evaluate our model.

In my view, the predictive distribution is the best way to (1) understand, (2) evaluate, and then (3) improve models.

You can use the predictive distribution as follows:

1. Fit your model with maximum likelihood.
2. Simulate a new outcome variable using the estimated model parameters (i.e., $f(y; the\hat{t}a)$). Perhaps simulate a handful for comparison.
3. Compare the simulated outcome variable(s) to the observed outcome variables.

Example: Poisson Distribution

Earlier, we fit a Poisson distribution to a sample of data from Hultman, Kathman, and Shannon (2013).

```
ml_est <- mean(civilian_casualties)
print(ml_est, digits = 3)
```

```
## [1] 630
```

```
n <- length(civilian_casualties)
y_pred <- rpois(n, lambda = ml_est)
print(y_pred[1:30])
```

```
##  [1] 649 597 672 646 589 602 656 652 621 664 590 622 622 678 639 615 662 613 603
## [20] 653 644 598 622 643 606 657 646 646 667 617
```

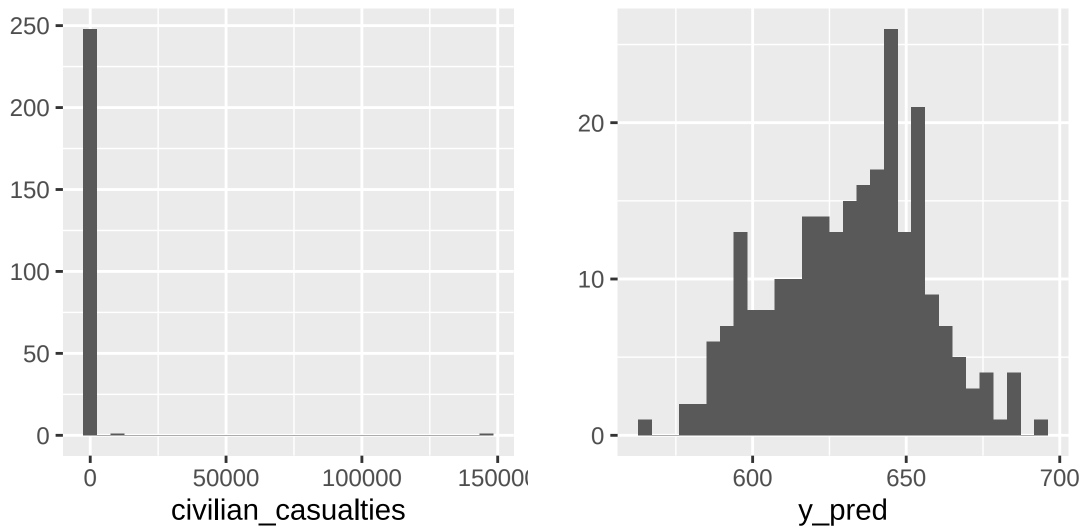
```
print(civilian_casualties[1:30])
```

```
##  [1]  0  0  0  0  0 13  0  0 61  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0
## [26]  0  0  0  0 19
```

Simply printing a few results, we can immediately see a problem with data, when compared with the raw data

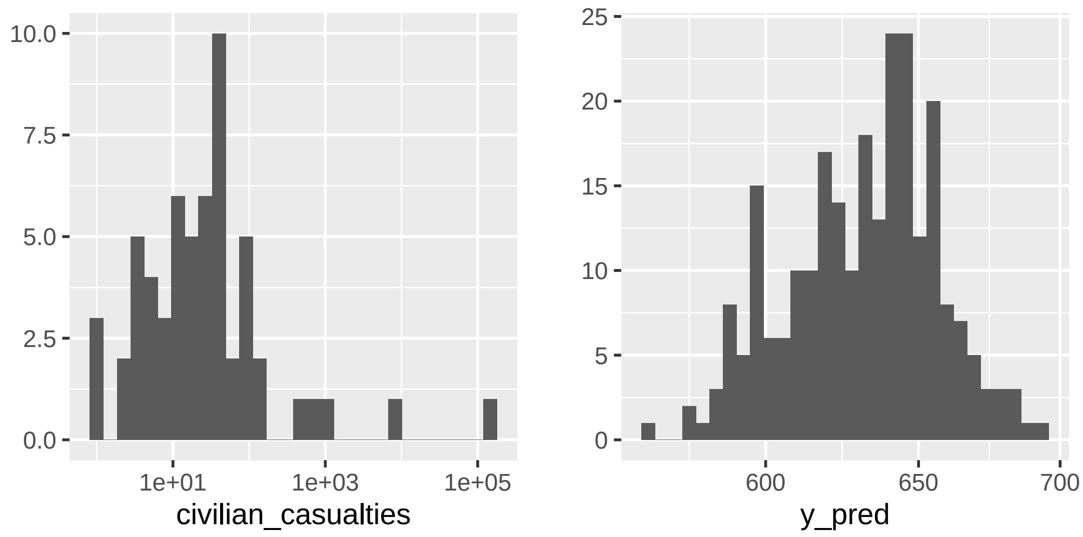
To see it even more clearly, we can create a histogram of the observed and simulated data.

```
library(patchwork)

p1 <- qplot(civilian_casualties)
p2 <- qplot(y_pred)

p1 + p2
```



These data sets are so different that the plots are difficult to read, so we might put the x-axes on the log scale. Note, though, that the two plots have very different ranges on the axes.

```
p1 <- qplot(civilian_casualties) + scale_x_log10()
p2 <- qplot(y_pred) + scale_x_log10()

p1 + p2
```

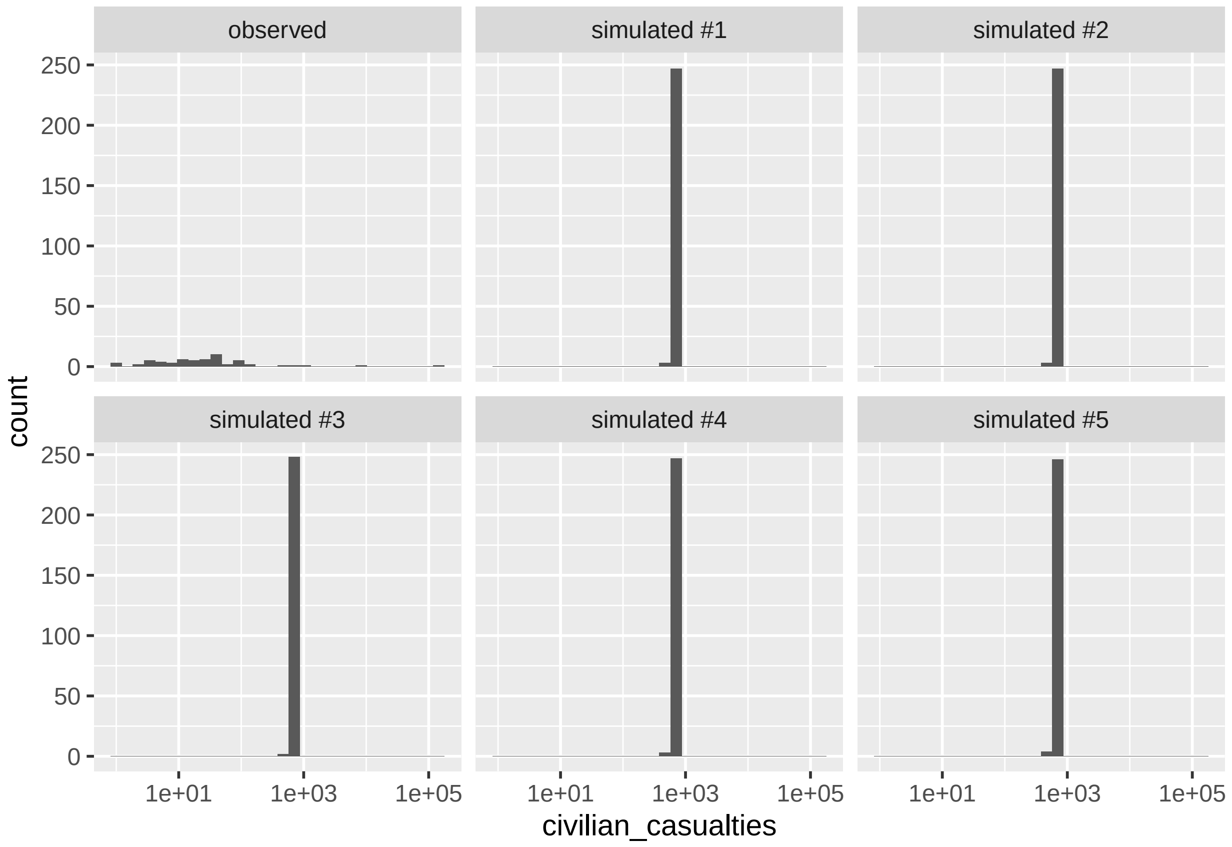


For a more accurate and complete comparison, let's simulate five fake data sets and use common axes

```
observed_data <- tibble(civilian_casualties, type = "observed")

sim_list <- list()
for (i in 1:5) {
  y_pred <- rpois(n, lambda = ml_est)
  sim_list[[i]] <- tibble(civilian_casualties = y_pred,
                          type = paste0("simulated #", i))
}
gg_data <- bind_rows(sim_list) %>%
  bind_rows(observed_data) %>%
  glimpse()
```

```
## Rows: 1,500
## Columns: 2
## $ civilian_casualties <dbl> 616, 666, 631, 595, 678, 664, 674, 642, 621, 633, ~
## $ type                <chr> "simulated #1", "simulated #1", "simulated #1", "s~
```

```
ggplot(gg_data, aes(x = civilian_casualties)) +
  geom_histogram() +
  facet_wrap(vars(type)) +
  scale_x_log10()
```



The fit of this model is almost absurd.

Example: Beta Distribution

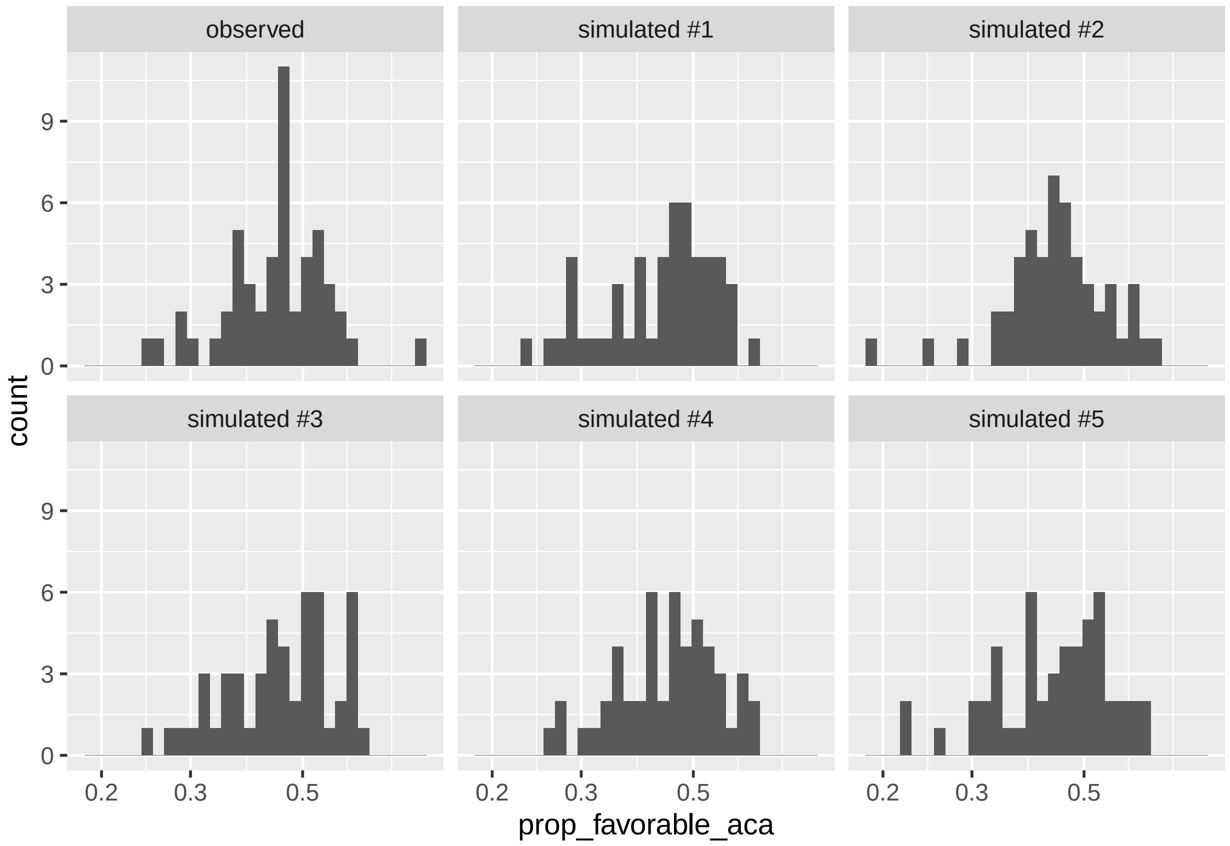
Now let's return to our beta model of states' opinions toward the ACA in the br data frame we loaded earlier.

```r
# obtain ml estimates
log_lik_fn <- function(par = c(2, 2), y) {
  a <- par[1]  # pulling these out makes the code a bit easier to follow
  b <- par[2]
  log_lik_i <- dbeta(y, shape1 = a, shape2 = b, log = TRUE)
  log_lik <- sum(log_lik_i)
  return(log_lik)
}
opt <- optim(par = c(2, 2), fn = log_lik_fn, y = br$prop_favorable_aca,
             control = list(fnscale = -1))
ml_est <- opt$par
```

Now let's simulate some fake data from the predictive distribution and compare that to the observed data

```r
observed_data <- br %>%
  mutate(type = "observed")

n <- nrow(br)
sim_list <- list()
for (i in 1:5) {
  y_pred <- rbeta(n, shape1 = ml_est[1], shape2 = ml_est[2])
  sim_list[[i]] <- tibble(prop_favorable_aca = y_pred,
                          type = paste0("simulated #", i))
}
```

```r
gg_data <- bind_rows(sim_list) %>%
  bind_rows(observed_data)

ggplot(gg_data, aes(x = prop_favorable_aca)) +
  geom_histogram() +
  facet_wrap(vars(type)) +
  scale_x_log10()
```



On the whole, we see hear a fairly close correspondence between the observed and simulated data. That suggests that our model is a good description of the data.

Exercises

From this week, you should be able to…

1. Starting with a given distribution (pdf or pmf), find the log-likihood function.
2. Optimize the log-likelihood function analytically or numerically (i.e., using `optim()`).
3. Use the invariance property of ML estimator to transform parameter estimates into estimates of quantities of interest.
4. Use the parametric bootstrap to compute confidence intervals.
5. Describe a “sampling distribution” and illustrate it with a computer simulation.
6. Describe the concepts of “bias” and “consistency.” Are ML estimates unbaised? Consistent? Under what conditions?
7. Use the predictive distribution to evaluate models.

Questions About the Exponential Distribution

Suppose we collect N random samples $x = \{x_1, x_2, ..., x_N\}$ and model each draw as a random variable $X \sim$ exponential(λ) with pdf $f(x_n | \lambda) = \lambda e^{-\lambda x_n}$.

1. Find the maximum likelihood estimator of λ.
2. Perform a Monte Carlo simulation to assess the bias in the ML estimator of λ you found above. Use 100,000 Monte Carlo simulations. Estimate the Monte Carlo error as $\frac{\text{SD of estimates}}{\sqrt{\text{number of MC simulations}}}$. Try a small sample size (e.g., $N = 5$ and a large (e.g., $N = 1,000$). Demonstrate analytically that the estimate is biased.
3. Is the ML estimator of λ consistent? Why or why not?
4. We interpret the parameter λ as a "rate." Find the ML estimate of the mean, which is the reciprocal of the rate. Is this estimate unbiased? Consistent?

Solution

1. The math follows the Poisson example closely. However, the solution is the inverse–$\hat{\lambda} = \frac{N}{\sum_{n=1}^{N} x_n} = \frac{1}{\text{avg}(x)}$.

Suppose a data set `x <- c(0.306, 0.023, 0.0471, 0.042, 0.227)`. Model this data set using an exponential distribution and estimate the rate λ using maximum likelihood. Find the estimates in two ways. First, compute the ML estimates using the analytical solution you found above. Second, derive the log-likelihood function, plot it, and use `optim()` to find the maximum. Show that the two solutions agree.

Load the `cancer` data frame from the survival package in R using `data(cancer, package="survival")`. Estimate both the rate and the mean. Use the parametric bootstrap to obtain a 95% confidence interval for each. Use the predictive distribution to evaluate the fit of the model. Do the simulated data sets seems to match the observed data sets?

Obtain the data for Barrilleaux and Rainey (2014) from GitHub: https://github.com/carlislerainey/aca-opinion/blob/master/Data/mrp_est.csv. Find the file `mrp_est.csv`. The variable `percent_supporting_expansion` gives the the percent of each state that supports expanding Medicaid under the ACA.

1. Model the distribution of *proportions* (you'll need to transform the variable from a percent to a proportion) as a beta distribution. Estimate the two shape parameters using maximum likelihood. Transform these estimates into estimates of the mean and SD. Compute a 95% confidence interval for the mean and SD using the parametric bootstap.
2. Simulate fake data from the predictive distribution and graphically compare these draws to the observed values. How do the observed data deviate from the model? Compare the the observed and simulated distributions in several ways, such as histograms, ecdf plots, two-number summaries, five-number summaries, etc. Look for ways that the simulated data sets consistently deviate from the observed.

Suppose a discrete uniform distribution from 0 to K. The pdf is $f(x; K) = \frac{1}{K}$ for $x \in \{0, 1, ..., K\}$. Suppose I have three samples from the distribution: 676, 459, and 912. Find the ML estimate of K.

DeGroot and Schervish, q. 9, p. 425. Suppose a distribution $f(x; \theta) = \theta x^{\theta - 1}$ for $0 < x < 1$ and $\theta > 0$. Find the ML estimator of θ.

Remember that the estimate $\hat{\lambda} = \text{avg}(y)$ for the Poisson distribution is unbiased. Remember that $E(y) = \lambda$ and SD(y) is $\sqrt{\lambda}$. By the invariance properity, the ML estimator of $SD(y) = \sqrt{\hat{\lambda}}$. Use a Monte Carlo simulation to assess whether (and how much) this estimator of the SD is biased. Be sure to experiment with the sample size of y (e.g., N = 5, N = 200, etc.), but use a large number of Monte Carlo simulations (e.g., 100,000).