Data Science per psicologi

Corrado Caudek

2021-09-13

# Indice

**Indice** **1**

**1 Modello Beta-Binomiale** **3**
1.1 Una proporzione . . . . . 3
1.2 Due proporzioni . . . . . 12
Considerazioni conclusive . . . . . 14

**2 Programmare in Stan** **15**
2.1 Che cos'è Stan? . . . . . 15
2.2 CmdStan . . . . . 16
2.3 Codice Stan . . . . . 17
2.4 Workflow . . . . . 20

**Bibliografia** **23**

*Capitolo 1*

# Modello Beta-Binomiale

## 1.1 Una proporzione

Si considerino $n$ variabili casuali Bernoulliane i.i.d.:

$$y = (y_1, \dots, y_n) \overset{iid}{\sim} \mathcal{B}(\theta).$$

Vogliamo stimare $\theta$ avendo osservato $y$. Essendo i.i.d., i dati possono essere riassunti dal numero totale di successi nelle $n$ prove, denotato da $y$. Il modello binomiale è

$$p(y \mid \theta) = \text{Bin}(y \mid n, \theta) = \binom{n}{y} \theta^y (1-\theta)^{n-y}, \tag{1.1}$$

dove nel termine di sinistra dell'equazione abbiamo ignorato $n$ in quanto viene considerato fisso per disegno.

L'inferenza sul modello binomiale richiede di assegnare una distribuzione a priori su $\theta$ che dipende dall'informazione disponibile a priori. Se sceglamo, ad esempio, una $\text{B}(2, 2)$ quale distribuzione a priori, il modello diventa:

$$\begin{aligned} y &\sim \text{Bin}(n, \theta) \\ \theta &\sim \text{B}(2, 2), \end{aligned} \tag{1.2}$$

dove la prima riga definisce la funzione di verosimiglianza e la seconda riga definisce la distribuzione a priori. Sulla base di ciò che è stato detto nel Capitolo **??**, sappiamo che le equazioni (1.2) definiscono il caso Beta-Binomiale.

### 1.1.1 Il presidente Trump e l'idrossiclorochina

Per fare un esempio concreto, consideriamo un set di dati reali. Cito dal *Washington Post* del 7 aprile 2020:

> One of the most bizarre and disturbing aspects of President Trump's nightly press briefings on the coronavirus pandemic is when he turns into a drug salesman. Like a cable TV pitchman hawking 'male enhancement' pills, Trump regularly extols the virtues of taking hydroxychloroquine, a drug used to treat malaria and lupus, as a potential 'game changer' that just might cure Covid-19.

Tralasciamo qui il fatto che il presidente Trump non è un esperto in questo campo. Esaminiamo invece le evidenze iniziali a supporto dell'ipotesi che l'idrossiclorochina possa essere utile per la cura del Covid-19, ovvero le evidenze che erano disponibili nel momento in cui il presidente Trump ha fatto le affermazioni riportate sopra (in seguito, quest'idea è stata screditata). Tali evidenze sono state fornite da uno studio di Gautret et al. (2020). Il disegno sperimentale di Gautret et al. (2020) comprende, tra le altre cose, il confronto tra una condizione sperimentale e una condizione di controllo. Il confronto importante è tra la proporzione di paziente positivi al virus SARS-CoV-2 nel gruppo sperimentale (a cui è stata somministrata l'idrossiclorochina; 6 su 14) e la proporzione di paziente positivi nel gruppo di controllo (a cui non è stata somministrata l'idrossiclorochina; ovvero 14 su 16). Obiettivo di questo Capitolo è mostrare come si possa fare inferenza sul modello (1.2) usando il linguaggio Stan.

### 1.1.2 Interfaccia `cmdstanr`

Nella seguente discussione verrà ottenuta una stima bayesiana del parametro $\theta$ usando l'interfaccia `cmdstanr` di CmdStan.[1]. Considereremo qui solo il gruppo di controllo. Iniziamo a caricare i pacchetti necessari:

```
library("cmdstanr")
set_cmdstan_path("/Users/corrado/.cmdstanr/cmdstan-2.27.0")
library("posterior")
rstan_options(auto_write = TRUE) # avoid recompilation of models
options(mc.cores = parallel::detectCores()) # parallelize across all CPUs
Sys.setenv(LOCAL_CPPFLAGS = '-march=native') # improve execution time
SEED <- 374237 # set random seed for reproducibility
```

Ci sono due passaggi essenziali per le analisi svolte mediante `cmdstanr`:

1. definire la struttura del modello bayesiano nella notazione Stan;
2. eseguire il campionamento della distribuzione a posteriori.

Esaminiamo questi due passaggi nel contesto del modello Beta-Binomiale definito dalla (1.2).

[1] I modelli discussi in questo capitolo sono discussi da Gelman et al. (1995) mentre il codice è stato ricavato dalla seguente pagina web.

### 1.1.3 Fase 1

È necessario definire i dati, i parametri e il modello. I *dati* del gruppo di controllo, che verrà qui esaminato, devono essere contenuti in un oggetto di classe `list`:

```
data1_list <- list(
  N = 16,
  y = c(rep(1, 14), rep(0, 2))
)
```

Il modello dipende dal *parametro* `theta`. In Stan, dobbiamo specificare che `theta` può essere un qualsiasi numero reale compreso tra 0 e 1.

Il *modello* è $\text{Bin}(n, \theta)$ e, nel linguaggio Stan, può essere scritto come

```
for (i in 1:N) {
  y[i] ~ bernoulli(theta);
}
```

ovvero come

```
y ~ bernoulli(theta);
```

La struttura del modello Beta-Binomiale viene tradotta nella sintassi Stan[2] e viene poi memorizzata come stringa di caratteri del file `oneprop1.stan`:

```
modelString = "
data {
  int<lower=0> N;
  int<lower=0, upper=1> y[N];
}
parameters {
  real<lower=0, upper=1> theta;
}
model {
  theta ~ beta(2, 2);
  y ~ bernoulli(theta);
  // the notation using ~ is syntactic sugar for
  //  target += beta_lpdf(theta | 1, 1);   // lpdf for continuous theta
  //  target += bernoulli_lpmf(y | theta); // lpmf for discrete y
  // target is the log density to be sampled
  //
  // y is an array of integers and
  //  y ~ bernoulli(theta);
```

[2] Si veda l'Appendice 2

```
  // is equivalent to
  //  for (i in 1:N) {
  //    y[i] ~ bernoulli(theta);
  //  }
  // which is equivalent to
  //  for (i in 1:N) {
  //    target += bernoulli_lpmf(y[i] | theta);
  //  }
}
generated quantities {
  int y_rep[N];
  real log_lik[N];
  for (n in 1:N) {
    y_rep[n] = bernoulli_rng(theta);
    log_lik[n] = bernoulli_lpmf(y[n] | theta);
  }
}
"
writeLines(modelString, con = "code/oneprop1.stan")
```

### 1.1.4 Fase 2

Leggiamo l'indirizzo del file che contiene il codice Stan:

```
file <- file.path("code", "oneprop1.stan")
```

Compiliamo il codice:

```
mod <- cmdstan_model(file)
```

Il campionamento MCMC si realizza con la chiamata:

```
fit1 <- mod$sample(
  data = data1_list,
  iter_sampling = 4000L,
  iter_warmup = 2000L,
  seed = SEED,
  chains = 4L,
  parallel_chains = 4L,
  refresh = 0,
  thin = 1
)
```

Avendo assunto una distribuzione a priori per il parametro $\theta$, l'algoritmo procede in maniera ciclica, correggendo la distribuzione a priori di $\theta$ condizionandola ai valori già generati. Dopo un certo numero di cicli, necessari per

portare l'algoritmo a convergenza, i valori estratti possono essere assunti come campionati dalla distribuzione a posteriori di $\theta$.

Si noti che `$sample()` richiede due tipi di informazioni. Innanzitutto, dobbiamo specificare le informazioni sul modello in base a:

- `mod` = la stringa di caratteri che definisce il modello (qui `oneprop1.stan`),
- `data` = i dati in formato lista (`data1_list`).

Dobbiamo inoltre specificare le informazioni sul campionamento MCMC utilizzando tre argomenti aggiuntivi:

- L'argomento `chains` specifica quante catene di Markov parallele eseguire. Eseguiamo qui quattro catene, quindi otteniamo quattro campioni distinti di valori $\pi$.
- L'argomento `iter` specifica il numero desiderato di iterazioni o la lunghezza di ciascuna catena di Markov. Per impostazione predefinita, la prima metà di queste iterazioni è costituita da campioni "burn-in" o "warm-up" che verranno ignorati. La seconda metà è conservata e costituisce un campione della distribuzione a posteriori.
- L'argomento `seed` per impostare il numero casuale che genera il seme per una simulazione `cmdstanr`.

### 1.1.5 Burn-in

Al crescere del numero di passi della catena, la distribuzione di target viene sempre meglio approssimata. All'inizio del campionamento, però, la distribuzione può essere significativamente lontana da quella stazionaria, e ci vuole un certo tempo prima di raggiungere la distribuzione stazionaria di equilibrio, detto, appunto, periodo di *burn-in*. I campioni provenienti da tale parte iniziale della catena vanno tipicamente scartati perché possono non rappresentare accuratamente la distribuzione a posteriori

### 1.1.6 Inferenza

Un sommario della distribuzione a posteriori si ottiene con:

```
fit1$summary(c("theta"))
#> # A tibble: 1 x 10
#>   variable  mean median     sd    mad    q5   q95  rhat
#>   <chr>    <dbl>  <dbl>  <dbl>  <dbl> <dbl> <dbl> <dbl>
#> 1 theta    0.802  0.813 0.0868 0.0867 0.644 0.928  1.00
#> # ... with 2 more variables: ess_bulk <dbl>, ess_tail <dbl>
```

Creiamo un oggetto di classe `stanfit`

```
stanfit1 <- rstan::read_stan_csv(fit1$output_files())
```

di dimensioni

```
dim(as.matrix(stanfit1, pars = "theta"))
#> [1] 16000     1
```

I primi 10 valori sono presentati qui di seguito

```
as.matrix(stanfit1, pars = "theta") %>%
  head(10)
#>           parameters
#> iterations    theta
#>       [1,] 0.757239
#>       [2,] 0.703550
#>       [3,] 0.772169
#>       [4,] 0.747881
#>       [5,] 0.765079
#>       [6,] 0.793709
#>       [7,] 0.857447
#>       [8,] 0.845012
#>       [9,] 0.824972
#>      [10,] 0.881219
```

La matrice precedente include i valori assunti dalla catena di Markov, ovvero un insieme di valori plausibili $\theta$ estratti dalla distribuzione a posteriori.

Un tracciato della catena di Markov illustra questa esplorazione rappresentando il valore $\theta$ sulle ordinate e l'indice progressivo di in ogni iterazione sull'ascissa. Usiamo la funzione `mcmc_trace()` del pacchetto `bayesplot` (Gabry et al. 2019) per costruire il grafico che include tutte e quattro le catene di Markov:

```
stanfit1 %>%
  mcmc_trace(pars = c("theta"), size = 0.1)
```

La figura 1.1 mostra che le catene esplorano uno spazio compreso approssimativamenre tra 0.7 e 0.9; tale figura descrive il comportamento *longitudinale* delle catene di Markov.

Possiamo anche esaminare la distribuzione degli stati della catena di Markov, ovvero, dei valori che queste catene visitano lungo il loro percorso, ignorando l'ordine di queste visite. L'istogramma della figura 1.2 fornisce una rappresentazione grafica di questa distribuzione per i 16000 valori complessivi delle quattro catene, ovvero per 4000 valori provienienti da ciascuna catena.

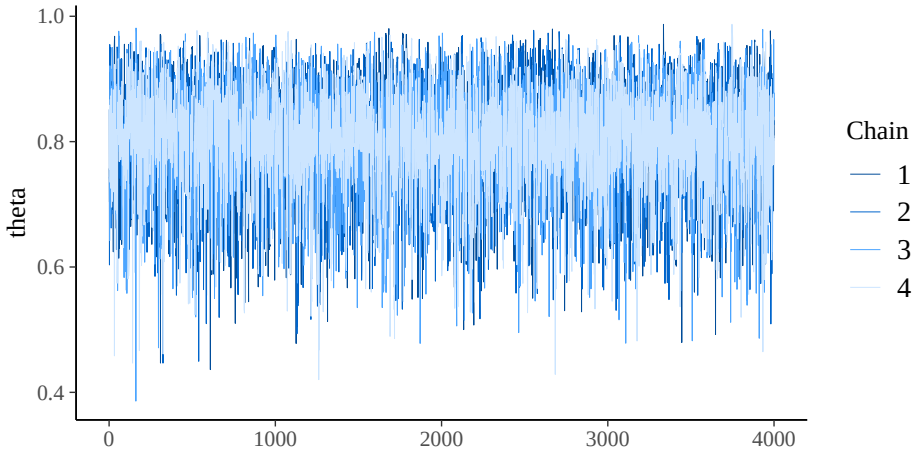


***Figura 1.1:*** *Trace-plot per il parametro $\theta$ nel modello Beta-Binomiale.*

```
mcmc_hist(stanfit1, pars = "theta") +
  yaxis_text(TRUE) +
  ylab("count")
```

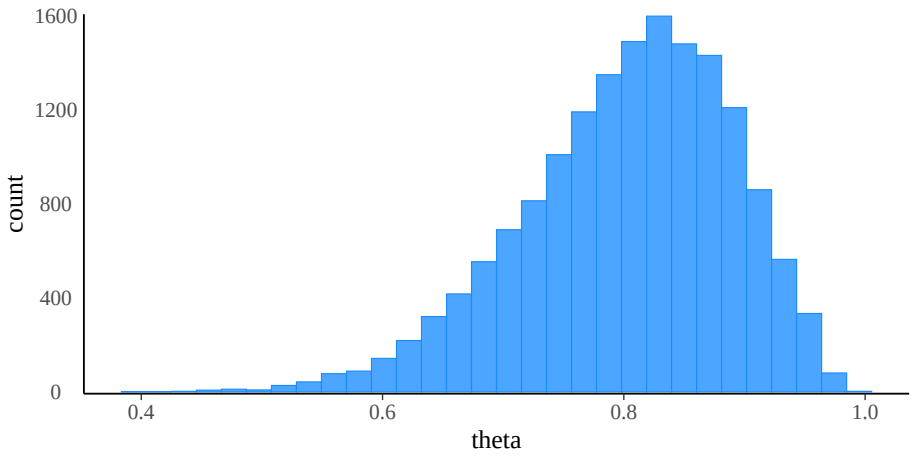


***Figura 1.2:*** *Istogramma che illustra l'approssimazione della distribuzione a posteriori per il parametro $\theta$ nel modello Beta-Binomiale.*

Nel modello Beta-Binomiale in cui la verosimiglianza è binomiale con 14 successi su 16 prove e in cui assumiamo una distribuzione a priori di tipo Beta$(2, 2)$ sul parametro $\theta$, la distribuzione a posteriori è ancora una distribuzione Beta di parametri $\alpha = 2 + 14$ e $\beta = 2 + 16$ - 14. La figura 1.3 riporta un kernel density plot per i valori delle quattro catene di Markov con sovrapposta in nero la densità Beta$(16, 4)$. Il punto importante è che la distribuzione dei valori del-

le catene di Markov produce un'eccellente approssimazione alla distribuzione bersaglio.[3]

```
mcmc_dens(stanfit1, pars = "theta") +
  yaxis_text(TRUE) +
  ylab("density") +
  stat_function(fun = dbeta, args = list(shape1 = 16, shape2=4))
```

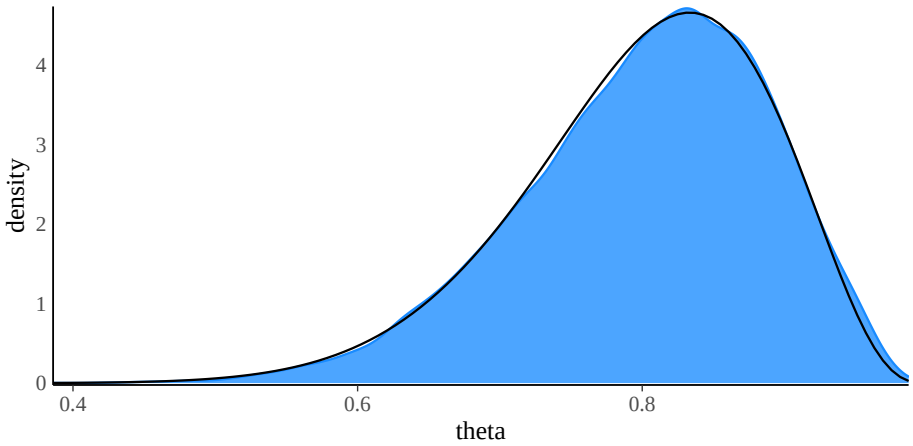


***Figura 1.3:*** *Istogramma che illustra l'approssimazione della distribuzione a posteriori per il parametro $\theta$ nel modello Beta-Binomiale. La curva nera rappresenta la corretta distribuzione a posteriori Beta(16, 4).*

Un intervallo di credibilità al 95% per $\theta$ si ottiene con la seguente chiamata:

```
posterior1 <- extract(stanfit1)
rstantools::posterior_interval(as.matrix(stanfit1), prob = 0.95)
#>                    2.5%       97.5%
#> theta         0.6107860   0.94402990
#> y_rep[1]      0.0000000   1.00000000
#> y_rep[2]      0.0000000   1.00000000
#> y_rep[3]      0.0000000   1.00000000
#> y_rep[4]      0.0000000   1.00000000
#> y_rep[5]      0.0000000   1.00000000
#> y_rep[6]      0.0000000   1.00000000
#> y_rep[7]      0.0000000   1.00000000
```

[3]Nel caso presente, il risultato è poco utile dato che è disponibile una soluzione analitica. Tuttavia, questo esercizio mette in evidenza il fatto cruciale che, nei casi in cui possiamo verificarne la soluzione, il campionamento Monte Carlo a catena di Markov è in grado di trovare la risposta corretta. Di conseguenza, possiamo anche essere sicuri che fornirà un'approssimazione alla distribuzione a posteriori anche in quei casi in cui una soluzione analitica non è disponibile.

```
#> y_rep[8]      0.0000000   1.00000000
#> y_rep[9]      0.0000000   1.00000000
#> y_rep[10]     0.0000000   1.00000000
#> y_rep[11]     0.0000000   1.00000000
#> y_rep[12]     0.0000000   1.00000000
#> y_rep[13]     0.0000000   1.00000000
#> y_rep[14]     0.0000000   1.00000000
#> y_rep[15]     0.0000000   1.00000000
#> y_rep[16]     0.0000000   1.00000000
#> log_lik[1]   -0.4930093  -0.05759735
#> log_lik[2]   -0.4930093  -0.05759735
#> log_lik[3]   -0.4930093  -0.05759735
#> log_lik[4]   -0.4930093  -0.05759735
#> log_lik[5]   -0.4930093  -0.05759735
#> log_lik[6]   -0.4930093  -0.05759735
#> log_lik[7]   -0.4930093  -0.05759735
#> log_lik[8]   -0.4930093  -0.05759735
#> log_lik[9]   -0.4930093  -0.05759735
#> log_lik[10]  -0.4930093  -0.05759735
#> log_lik[11]  -0.4930093  -0.05759735
#> log_lik[12]  -0.4930093  -0.05759735
#> log_lik[13]  -0.4930093  -0.05759735
#> log_lik[14]  -0.4930093  -0.05759735
#> log_lik[15]  -2.8829362  -0.94362543
#> log_lik[16]  -2.8829362  -0.94362543
#> lp__        -12.7342100 -10.00860000
```

Svolgendo un'analisi bayesiana simile a questa, Gautret et al. (2020) hanno trovato che gli intervalli di credibilità del gruppo di controllo e del gruppo sperimentale non si sovrappongono. Questo fatto viene interpretato dicendo che il parametro $\theta$ è diverso nei due gruppi. Sulla base di queste evidenza, Gautret et al. (2020) hanno concluso, con un grado di certezza soggettiva del 95%, che nel gruppo sperimentale vi è una probabilità più bassa di risultare positivi al SARS-CoV-2 rispetto al gruppo di controllo. In altri termini, questa analisi dei dati suggerisce che l'idrossiclorochina sia efficace come terapia per il Covid-19.

### 1.1.7 La critica di Hulme et al. (2020)

Un articolo pubblicato da Hulme et al. (2020) si è posto il problema di rianalizzare i dati di Gautret et al. (2020).[4] Tra gli autori di questo articolo figura anche Eric-Jan Wagenmakers, uno psicologo molto conosciuto per i suoi contributi metodologici. Hulme et al. (2020) hanno osservato che, nelle analisi statistiche

[4] Si veda https://osf.io/5dgmx/.

riportate, Gautret et al. (2020) hanno escluso alcuni dati. Nel gruppo sperimentale, infatti, vi erano alcuni pazienti i quali, anziché migliorare, sono in realtà peggiorati. L'analisi statistica di Gautret et al. (2020) ha escluso i dati di questi pazienti. Se consideriamo tutti i pazienti — non solo quelli selezionati da Gautret et al. (2020) — la situazione diventa la seguente:

- gruppo sperimentale: 10 positivi su 18;
- gruppo di controllo: 14 positivi su 16.

L'analisi dei dati proposta da Hulme et al. (2020) richiede l'uso di alcuni strumenti statistici che, in queste dispense, non verranno discussi. Ma possiamo giungere alle stesse conclusioni raggiunte da questi ricercatori anche usando le procedure statistiche descritte nel Paragrafo successivo.

## 1.2 Due proporzioni

Svolgiamo ora l'analisi considerando tutti i dati, come suggerito da Hulme et al. (2020). Per fare questo verrà creato un modello bayesiano per fare inferenza sulla differenza tra due proporzioni. Una volta generate le distribuzioni a posteriori per le proporzioni di "successi" nei due gruppi, verrà anche generata la quantità

$$\omega = \frac{\theta_2/(1-\theta_2)}{\theta_1/(1-\theta_1)},$$

ovvero il rapporto tra gli Odds di positività tra i pazienti del gruppo di controllo e gli Odds di positività tra i pazienti del gruppo sperimentale. Se il valore dell'OR è uguale a 1, significa che l'Odds di positività nel gruppo di controllo è uguale all'odds di positività nel gruppo sperimentale, cioè il fattore in esame (somministrazione dell'idrossiclorochina) è ininfluente sulla comparsa della malattia. L'inferenza statistica sull'efficacia dell'idrossiclorochina come terapia per il Covid-19 può dunque essere effettuata esaminando l'intervallo di credibilità al 95% per l'OR: se tale intervallo include il valore 1, allora non vi è evidenza che l'idrossiclorochina sia efficace come terapia per il Covid-19.

Nell'implementazione di questo modello, la quantità di interesse è dunque l'odds ratio; tale quantità viene calcolata nel blocco `generated quantities` del programma Stan. In questo esempio useremo delle distribuzioni a priori vagamente informative per i parametri $\theta_1$ e $\theta_1$.

```
data_list <- list(
  N1 = 18,
  y1 = 10,
  N2 = 16,
  y2 = 14
)
```

```
modelString = "
//  Comparison of two groups with Binomial
data {
  int<lower=0> N1;              // number of experiments in group 1
  int<lower=0> y1;              // number of deaths in group 1
  int<lower=0> N2;              // number of experiments in group 2
  int<lower=0> y2;              // number of deaths in group 2
}
parameters {
  real<lower=0,upper=1> theta1; // probability of death in group 1
  real<lower=0,upper=1> theta2; // probability of death in group 2
}
model {
  theta1 ~ beta(2, 2);          // prior
  theta2 ~ beta(2, 2);          // prior
  y1 ~ binomial(N1, theta1);    // observation model / likelihood
  y2 ~ binomial(N2, theta2);    // observation model / likelihood
}
generated quantities {
  // generated quantities are computed after sampling
  real oddsratio = (theta2/(1-theta2))/(theta1/(1-theta1));
}
"
writeLines(modelString, con = "code/twoprop1.stan")
```

```
file <- file.path("code", "twoprop1.stan")
```

```
mod <- cmdstan_model(file)
```

```
fit <- mod$sample(
  data = data_list,
  iter_sampling = 4000L,
  iter_warmup = 2000L,
  seed = SEED,
  chains = 4L,
  parallel_chains = 4L,
  refresh = 0,
  thin = 1
)
```

```
stanfit <- rstan::read_stan_csv(fit$output_files())
```

```
print(
  stanfit,
  pars = c("theta1", "theta2", "oddsratio"),
  digits_summary = 3L
)
#> Inference for Stan model: twoprop1-202109141358-1-94447a.
#> 4 chains, each with iter=6000; warmup=2000; thin=1;
#> post-warmup draws per chain=4000, total post-warmup draws=16000.
#>
#>            mean se_mean    sd  2.5%   25%   50%   75%
#> theta1    0.546   0.001 0.104 0.337 0.475 0.547 0.619
#> theta2    0.801   0.001 0.087 0.605 0.747 0.812 0.865
#> oddsratio 4.859   0.049 4.740 0.914 2.221 3.599 5.933
#>            97.5% n_eff  Rhat
#> theta1     0.743 11214 1.000
#> theta2     0.939 12359 1.000
#> oddsratio 16.251  9207 1.001
#>
#> Samples were drawn using NUTS(diag_e) at Mar Set 14 13:58:01 2021.
#> For each parameter, n_eff is a crude measure of effective sample size,
#> and Rhat is the potential scale reduction factor on split chains (at
#> convergence, Rhat=1).
```

L'intervallo di credibilità del 95% per l'OR include il valore di 1.0 (ovvero, il valore che indica che gli odds di positività sono uguali nei due gruppi). In base agli standard correnti, un risultato di questo tipo non viene considerato come evidenza sufficiente per potere concludere che il parametro $\theta$ assume un valore diverso nei due gruppi. In altri termini, se consideriamo tutti i dati, e non solo quelli selezionati dagli autori della ricerca originaria, non vi è evidenza alcuna che l'idrossiclorochina sia efficace come terapia per il Covid-19.

## Considerazioni conclusive

Concludiamo questa discussione dicendo che ciò che è stato presentato in questo capitolo è un esercizio didattico: la ricerca di Gautret et al. (2020) include tante altre informazioni che non sono state qui considerate. Tuttavia, notiamo che la semplice analisi statistica che abbiamo qui descritto è stata in grado di replicare le conclusioni a cui sono giunti (per altra via) Hulme et al. (2020).

*Capitolo 2*

# Programmare in Stan

## 2.1 Che cos'è Stan?

STAN è un linguaggio di programmazione probabilistico usato per l'inferenza bayesiana (Carpenter et al., 2017). Prende il nome da uno dei creatori del metodo Monte Carlo, Stanislaw Ulam (Eckhardt, 1987). Stan consente di generare campioni da distribuzioni di probabilità basati sulla costruzione di una catena di Markov avente come distribuzione di equilibrio (o stazionaria) la distribuzione desiderata.

Nelle parole degli autori:

> Stan is a state-of-the-art platform for statistical modeling and high-performance statistical computation. Thousands of users rely on Stan for statistical modeling, data analysis, and prediction in the social, biological, and physical sciences, engineering, and business. Users specify log density functions in Stan's probabilistic programming language and get: full Bayesian statistical inference with MCMC sampling (NUTS, HMC); approximate Bayesian inference with variational inference (ADVI): penalized maximum likelihood estimation with optimization (L-BFGS).

Esistono varie interfacce di STAN per diversi programmi, tra cui R, interfaccia a riga di comando (*command line*), Python, MATLAB, Julia, Stata e Mathematica.

### 2.1.1 Post-processing

Ci sono molti pacchetti R che facilitano l'uso di Stan, tra cui

- `rstan`: General R Interface to Stan
- `shinystan`: Interactive Visual and Numerical Diagnostics and Posterior Analysis for Bayesian Models

- `bayesplot`: Plotting functions for posterior analysis, model checking, and MCMC diagnostics.
- `brms`: Bayesian Regression Models using 'Stan', covering a growing number of model types
- `rstanarm`: Bayesian Applied Regression Modeling via Stan, with an emphasis on hierarchical/multilevel models
- `edstan`: Stan Models for Item Response Theory
- `rstantools`: Tools for Developing R Packages Interfacing with Stan

## 2.2 CmdStan

Negli esempi qui discusi verrà usata `CmdStanR`, un'interfaccia R per `CmdStan` che costituisce una recente alternativa alla tradizionale interfaccia `RStan`. `CmdStanR` non è ancora disponibile su CRAN, ma può essere installato come indicato su questo link.

Una volta installato `CmdStanR`, il pacchetto `cmdstanr` può essere caricato come un qualsiasi altro pacchetto R.

Si noti che `CmdStanR` richiede un'installazione funzionante di `CmdStan`, l'interfaccia shell per Stan. Se `CmdStan` non è installato, `CmdStanR` lo installerà automaticamente se il computer dispone di una *Toolchain* adatta. Stan richiede infatti che sul computer su cui viene installato siano presenti alcuni strumenti necessari per gestire i file C++. Tra le altre ragioni, questo è dovuto al fatto che il codice Stan viene tradotto in codice C++ e compilato. Il modo migliore per ottenere il software necessario per un computer Windows o Mac è quello di installare `RTools`. Per un computer Linux, è necessario installare `build-essential` e una versione recente dei compilatori g++ o clang++. I requisiti sono descritti nella Guida di CmdStan.

Per verificare che la Toolchain sia configurata correttamente è possibile utilizzare la funzione `check_cmdstan_toolchain()`:

```
check_cmdstan_toolchain()
```

Se la toolchain è configurata correttamente, `CmdStan` può essere installato mediante la funzione `install_cmdstan()`:

```
# do not run!
# install_cmdstan(cores = 2)
```

Prima di poter utilizzare `CmdStanR`, è necessario specificare dove si trova l'installazione di `CmdStan`.

```
# do not run!
# set_cmdstan_path(PATH_TO_CMDSTAN)
```

Sul mio computer `PATH_TO_CMDSTAN` è la seguente stringa (incluse le virgolette):

```
cmdstan_path()
#> [1] "/Users/corrado/.cmdstanr/cmdstan-2.27.0"
```

La versione installata di `CmdStan` si ottiene con:

```
cmdstan_version()
#> [1] "2.27.0"
```

## 2.3 Codice Stan

Qualunque sia l'interfaccia che viene usata, i modelli sottostanti sono sempre scritti nel linguaggio Stan, il che significa che lo stesso codice Stan è valido per tutte le interfacce possibili. Il codice Stan è costituito da una serie di blocchi che vengono usati per specificare un modello statistico. In ordine, questi blocchi sono: `data`, `transformed data`, `parameters`, `transformed parameters`, `model`, e `generated quantities`.

Un programma Stan contiene tre "blocchi" obbligatori: blocco `data`, blocco `parameters`, blocco `model`.

### 2.3.1 Blocco `data`

Qui vengono dichiarate le variabili che saranno passate a Stan. Devono essere elencati i nomi delle variabili che saranno utilizzate nel programma, il *tipo di dati* da registrare per ciascuna variabile, per esempio: - *int* = intero, - *real* = numeri reali (ovvero, numeri con cifre decimali), - *vector* = sequenze ordinate di numeri reali unidimensionali, - *matrix* = matrici bidimensionali di numeri reali, - *array* = sequenze ordinate di dati multidimensionali.

Devono anche essere dichiarate le dimensioni delle variabili e le eventuali restrizioni sulle variabili (es. `upper = 1 lower = 0`, che fungono da controlli per Stan). Tutti i nomi delle variabili assegnate qui saranno anche usati negli altri blocchi del programma.

Per esempio, l'istruzione seguente dichiaria la variabile `Y` – la quale rappresenta, ad esempio, l'altezza di 10 persone – come una variabile di tipo `real[10]`. Ciò significa che specifichiamo un array di lunghezza 10, i cui elementi sono variabili continue definite sull'intervallo dei numeri reali $[-\infty, +\infty]$.

```
data {
  real Y[10]; // heights for 10 people
}
```

Invece, con l'istruzione

```
data {
  int Y[10]; // qi for 10 people
}
```

dichiariamo la variabile `Y` – la quale rappresenta, ad esempio, il QI di 10 persone – come una variabile di tipo `int[10]`, ovvero un array di lunghezza 10, i cui elementi sono numeri naturali, cioè numeri interi non negativi $\{0, +1, +2, +3, +4, \dots\}$.

Un altro esempio è

```
data {
  real<lower=0, upper=1> Y[10]; // 10 proportions
}
```

nel quale viene specificato un array di lunghezza 10, i cui elementi sono delle variabili continue definite sull'intervallo dei numeri reali $[0, 1]$ — per esempio, delle proporzioni.

Si noti che i tipi `vector` e `matrix` contengono solo elementi di tipo `real`, ovvero variabili continue, mentre gli `array` possono contenere dati di qualsiasi tipo. I dati passati a Stan devono essere contenuti in un oggetto del tipo `list`.

### 2.3.2 Blocco `parameters`

I parametri che vengono stimati sono dichiarati nel blocco `parameters`. Per esempio, l'istruzione

```
parameters {
  real mu; // mean height in population
  real<lower=0> sigma; // sd of height distribution
}
```

dichiaria la variabile `mu` che codifica l'altezza media nella popolazione, che è una variabile continua in un intervallo illimitato di valori, e la deviazione standard `sigma`, che è una variabile continua non negativa. Avremmo anche potuto specificare un limite inferiore di zero su `mu` perché deve essere non negativo.

Per una regressione lineare semplice, ad esempio, devono essere dichiarate le variabili corrispondenti all'intercetta (`alpha`), alla pendenza (`beta`) e alla deviazione standard degli errori attorno alla linea di regressione (`sigma`). In altri termini, nel blocco `parameters` devono essere elencati tutti i parametri che dovranno essere stimati dal modello. Si noti che parametri discreti non sono possibili. Infatti, Stan attualmente non supporta i parametri con valori interi, almeno non direttamente.

### 2.3.3 Blocco `model`

Nel blocco `model` vengono elencate le dichiarazioni relative alla verosimiglianza dei dati e alle distribuzioni a priori dei parametri, come ad esempio, nelle istruzioni seguenti.

```
model {
  for(i in 1:10) {
    Y[i] ~ normal(mu, sigma);
  }
  mu ~ normal(170, 15); // prior for mu
  sigma ~ cauchy(0, 20); // prior for sigma
}
```

Mediante l'istruzione all'interno del ciclo `for`, ciascun valore dell'altezza viene concepito come una variable casuale proveniente da una distribuzione Normale di parametri $\mu$ e $\sigma$ (i parametri di interesse nell'inferenza). Il ciclo `for` viene ripetuto 10 volte perché i dati sono costituiti da un array di 10 elementi (ovvero, il campione è costituito da 10 osservazioni).

Le due righe che seguno il ciclo `for` specificano le distribuzioni a priori dei parametri su cui vogliamo effettuare l'inferenza. Per $\mu$ assumiamo una distribuzione a priori Normale di parametri $\mu = 170$ e $\sigma = 15$; per $\sigma$ assumiamo una distribuzione a priori Cauchy(0, 20).

Se non viene definita alcuna distribuzione a priori, Stan utilizzerà la distribuzione a priori predefinita $Unif(-\infty, +\infty)$. Raccomandazioni sulle distribuzioni a priori sono fornite in questo link.

La precedente notazione di campionamento può anche essere espressa usando la seguente notazione alternativa:

```
  for(i in 1:10) {
    target += normal_lpdf(Y[i] | mu, sigma);
  }
```

Questa notazione rende trasparente il fatto che, in pratica, Stan esegue un campionamento nello spazio

$$\log p(\theta \mid y) \propto \log p(y \mid \theta) + \log p(\theta) = \sum_{i=1}^{n} \log p(y_i \mid \theta) + \log p(\theta).$$

Per ogni passo MCMC, viene ottenuto un nuovo valore di $\mu$ e $\sigma$ eviene valutata la log densità a posteriori non normalizzata. Ad ogni passo MCMC, Stan calcola un nuovo valore della densità a posteriori su scala logaritmica partendo da un valore di 0 e incrementandola ogni volta che incontra un'istruzione ~. Quindi, le istruzioni precedenti aumentano la log-densità di una quantità pari a $\log(p(Y[i])) \propto -\frac{1}{2}\log(\sigma^2) - (Y[i] - \mu)^2/2\sigma^2$ per le altezze si ciascuno degli $i = 1 \dots, 10$ individui – laddove la formula esprime, in termini logaritmici, la densità Normale da cui sono stati esclusi i termini costanti.

Oppure, in termini vettorializzati, il modello descritto sopra può essere espresso come

```
model {
  Y ~ normal(mu, sigma);
}
```

dove il termine a sinistra di $\sim$ è un array. Questa notazione più compatta è anche la più efficiente.

### 2.3.4 Blocchi opzionali

Ci sono inoltre tre blocchi opzionali:

- Il blocco `transformed data` consente il pre-processing dei dati. È possibile trasformare i parametri del modello; solitamente ciò viene fatto nel caso dei modelli più avanzati per consentire un campionamento MCMC più efficiente.
- Il blocco `transformed parameters` consente la manipolazione dei parametri prima del calcolo della distribuzione a posteriori.
- Il blocco `generated quantities` consente il post-processing riguardante qualsiasi quantità che non fa parte del modello ma può essere calcolata a partire dai parametri del modello, per ogni iterazione dell'algoritmo. Esempi includono la generazione dei campioni a posteriori e le dimensioni degli effetti.

### 2.3.5 Sintassi

Si noti che il codice Stan richiede i punti e virgola (;) alla fine di ogni istruzione di assegnazione. Questo accade per le dichiarazioni dei dati, per le dichiarazioni dei parametri e ovunque si acceda ad un elemento di un tipo `data` e lo si assegni a qualcos'altro. I punti e virgola non sono invece richiesti all'inizio di un ciclo o di un'istruzione condizionale, dove non viene assegnato nulla.

In STAN, qualsiasi stringa che segue `//` denota un commento e viene ignorata dal programma.

Stan è un linguaggio estremamente potente e consente di implementare quasi tutti i modelli statistici, ma al prezzo di un certo sforzo di programmazione. Anche l'adattamento di semplici modelli statistici mediante il linguaggio STAN a volte può essere laborioso. Per molti modelli comunemente usati, come i modelli di regressione e multilivello, tale processo può essere semplificato usando le funzioni del pacchetto `brms`. D'altra parte, per modelli veramente complessi, non ci sono molte alternative all'uso di STAN. Per chi è curioso, il manuale del linguaggio Stan è accessibile al seguente link.

## 2.4 Workflow

Se usiamo `CmdStan`, dobbiamo prima scrivere il codice con il modello, poi compilare il codice con la funzione `cmdstan_model()`, poi eseguire il campionamento

MCMC con il metodo `$sample()` e, infine, creare un sommario dei risultati usando, per esempio, usando il metodo `$summary()`.

# Bibliografia


Carpenter, B., Gelman, A., Hoffman, M. D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M., Guo, J., Li, P., and Riddell, A. (2017). Stan: A probabilistic programming language. *Journal of statistical software*, 76(1):1–32.

Eckhardt, R. (1987). Stan Ulam, John Von Neumann and the Monte Carlo Method. *Los Alamos Science Special Issue*.

Gautret, P., Lagier, J. C., Parola, P., Meddeb, L., Mailhe, M., Doudier, B., and ... Honoré, S. (2020). Hydroxychloroquine and azithromycin as a treatment of covid-19: Results of an open-label non-randomized clinical trial. *International Journal of Antimicrobial Agents*.

Gelman, A., Carlin, J. B., Stern, H. S., and Rubin, D. B. (1995). *Bayesian data analysis*. Chapman and Hall/CRC.

Hulme, O. J., Wagenmakers, E. J., Damkier, P., Madelung, C. F., Siebner, H. R., Helweg-Larsen, J., and ... Madsen, K. H. (2020). Reply to gautret et al. 2020: A bayesian reanalysis of the effects of hydroxychloroquine and azithromycin on viral carriage in patients with covid-19. *medRxiv*.