Cross-validation in R and Stan

Luis Usier

June 24, 2016

Goalkeepers

```
Source: local data frame [209 x 2]

     keeper  save
      <chr> <lgl>
1  Courtois FALSE
2    De Gea  TRUE
3    Lloris  TRUE
4      Hart  TRUE
5      Hart  TRUE
6      Hart  TRUE
7    De Gea  TRUE
8      Hart  TRUE
9  Courtois  TRUE
10     Cech  TRUE
..      ...   ...
```

Bayesian Data Analysis

1. Set up probability model:
   - denote saves by s and goals by g
   - $s_i \sim Bern(p_{k_i})$
2. Solve for posterior distribution:
   - analytically in this case
   - $p_{k_i} \sim Beta(\sum s_{k_i}, \sum g_{k_i})$

Bayesian Data Analysis

```
Source: local data frame [5 x 4]

    keeper alpha  beta  mean
     <chr> <int> <int> <dbl>
1     Cech    25    21  0.54
2 Courtois    24    13  0.65
3   De Gea    29     9  0.76
4     Hart    39     8  0.83
5   Lloris    33     8  0.80
```

Going Hierarchical

This model is too simple:

- Information about one keeper tells us about the other keepers
- What do we do with new keepers?
- Make the model *hierarchical*

Going Hierarchical

Keeper skills normally distributed:

$$\pi_k \sim N(\mu, 0.5)$$

Map skills from real numbers to $[0, 1]$:

$$p_k = \frac{1}{1 + e^{\pi_k}}$$

Saves are Bernoulli trials:

$$s_i \sim Bern(p_{k_i})$$

Going Hierarchical

No analytical solution; must fit in Stan.

```
data {
  int shots; int keepers;
  int save[shots]; int keeper[shots];
}
parameters {
  real mu;
  vector[keepers] pi;
}
transformed parameters{
  vector[keepers] p;

  for (i in 1:keepers)
    p[i] <- 1 / (1 + exp(-pi[i]));
}
model {
  pi ~ normal(mu, 1);

  for (i in 1:shots)
    save[i] ~ bernoulli(p[keeper[i]]);
}
```

Going Hierarchical

```
Source: local data frame [5 x 5]

    keeper  mean  2.5% 97.5%  true
     <chr> <dbl> <dbl> <dbl> <dbl>
1     Cech  0.56  0.43  0.70  0.53
2 Courtois  0.66  0.52  0.80  0.66
3   De Gea  0.76  0.61  0.87  0.72
4     Hart  0.82  0.71  0.91  0.76
5   Lloris  0.80  0.66  0.90  0.84
```

Scale Parameters

Back to the model:

$$\pi_k \sim N(\mu, \mathbf{0.5})$$

How did we choose 0.5???

Ideally, should be estimated from the data just like other parameters:

$$\pi_k \sim N(\mu, \sigma)$$

Scale Parameters

```
data {
  int shots; int keepers;
  int save[shots]; int keeper[shots];
}
parameters {
  real mu;
  real<lower=0> sigma;
  vector[keepers] pi;
}
transformed parameters{
  vector[keepers] p;

  for (i in 1:keepers)
    p[i] <- 1 / (1 + exp(-pi[i]));
}
model {
  pi ~ normal(mu, sigma);

  for (i in 1:shots)
    save[i] ~ bernoulli(p[keeper[i]]);
}
```

Scale Parameters

```
Warning: Deprecated, use tibble::rownames_to_column() instead.
```

```
Source: local data frame [8 x 5]

  parameter    mean    2.5%   97.5%  true
      <chr>   <dbl>   <dbl>   <dbl> <dbl>
1        mu    1.01    0.19    1.93  1.00
2     sigma    0.82    0.18    2.30  0.50
3      p[1]    0.59    0.43    0.72  0.53
4      p[2]    0.67    0.52    0.79  0.66
5      p[3]    0.75    0.63    0.86  0.72
6      p[4]    0.80    0.68    0.89  0.76
7      p[5]    0.78    0.67    0.89  0.84
8      lp__ -121.28 -126.72 -118.04    NA
```

Seems fine… **but** there are hidden problems.

Scale Parameters

Adding the ML estimates from Stan's built-in optimizer:

```
Source: local data frame [8 x 6]

  parameter    mean    2.5%   97.5%  true        MLE
      <chr>   <dbl>   <dbl>   <dbl> <dbl>      <dbl>
1        mu    1.01    0.19    1.93  1.00   1.423282
2     sigma    0.82    0.18    2.30  0.50   0.000001
3      p[1]    0.59    0.43    0.72  0.53   0.805852
4      p[2]    0.67    0.52    0.79  0.66   0.805853
5      p[3]    0.75    0.63    0.86  0.72   0.805853
6      p[4]    0.80    0.68    0.89  0.76   0.805853
7      p[5]    0.78    0.67    0.89  0.84   0.805853
8      lp__ -121.28 -126.72 -118.04    NA -60.850666
```

The posterior distribution is unbounded, and thus **improper**.

As a result, the assumptions that underlie MCMC break down and, in general, sampling will not be meaningful.

The problem is worse for models with more levels and more variance parameters, and/or more groups.

Cross-validating parameters

If you're building deep hierarchical models in Stan, sooner or later you may run into this problem.

When that happens, cross-validation is an alternative.

Cross-validating parameters

In cross-validation, we:

1. Fit the model on a subset of the data
2. See how well the model predicts the outstanding data
3. Select amongst the models according to their predictive accuracy

Cross-validating parameters

Ideally, the subsets of the data would be of size n - 1

The problem: running one model takes long enough, running n would take forever

The package `loo` implements an algorithm that approximates leave-one-out cross-validation in RStan.

Cross-validating parameters

```
data {
  int shots; int keepers;
  int save[shots]; int keeper[shots];
  real<lower=0> sigma;
}
parameters {
  real mu; vector[keepers] pi;
}
transformed parameters{
  vector[keepers] p;
  for (i in 1:keepers)
    p[i] <- 1 / (1 + exp(-pi[i]));
}
model {
  pi ~ normal(mu, sigma);
  for (i in 1:shots)
    save[i] ~ bernoulli(p[keeper[i]]);
}
generated quantities {
  vector[shots] log_lik;
  for (i in 1:shots)
    log_lik[i] <- bernoulli_log(save[i], p[keeper[i]]);
}
```

Cross-validating parameters

```
list(fit25, fit50, fit100) %>%
  map(extract_log_lik) %>% map(loo) %>% walk(print)
```

```
Computed from 1000 by 209 log-likelihood matrix

         Estimate   SE
elpd_loo   -122.9  6.1
p_loo         2.2  0.1
looic       245.8 12.1

All Pareto k estimates OK (k < 0.5)
Computed from 1000 by 209 log-likelihood matrix

         Estimate   SE
elpd_loo   -122.4  6.5
p_loo         3.6  0.3
looic       244.7 13.0

All Pareto k estimates OK (k < 0.5)
Computed from 1000 by 209 log-likelihood matrix

         Estimate   SE
elpd_loo   -123.3  6.9
p_loo         5.0  0.4
looic       246.6 13.8
```

Cross-validating models

The same principle can be applied when comparing two different models altogether

Suppose that in our initial model we had used a different link function: probit instead of logistic

$$p_k = \Phi(\pi_k)$$

Cross-validating models

```
list(fit_log, fit_prob) %>%
  map(extract_log_lik) %>% map(loo) %>% walk(print)
```

```
Computed from 1000 by 209 log-likelihood matrix

         Estimate   SE
elpd_loo   -122.0  6.5
p_loo         3.2  0.2
looic       244.0 13.0

All Pareto k estimates OK (k < 0.5)
Computed from 1000 by 209 log-likelihood matrix

         Estimate   SE
elpd_loo   -122.7  7.3
p_loo         4.3  0.4
looic       245.4 14.6

All Pareto k estimates OK (k < 0.5)
```

Cross-validating time series

Take now a slightly more complicated time series model:

Underlying propensities are a stochastic function of time:

$$y_{t+1} \sim N(y_t e^{-\beta \Delta t}, \sigma \sqrt{1 - (e^{-\beta \Delta t})^2})$$

$$p_t = \frac{1}{1 + e^{-y_t}}$$

Binomial poll results:

$$rem_t \sim Binom(200, p_t)$$

Cross-validating time series

Parameters of interest: individual y_t, σ and β

Finding out just the individual y_t would be very easy, but also not very informative or predictive

However, σ and β cannot be easily fit because again, the likelihood is unbounded

Cross-validating time series

In Stan:

```
data {
  int n; int t[n]; int r[n];
}
parameters {
  real<lower=0> beta; real<lower=0> sigma;
  vector[n] y;
}
transformed parameters {
  vector[n] p;
  for (i in 1:n)
    p[i] <- 1 / (1 + exp(-y[i]));
}
model {
  y[1] ~ normal(0, sigma);
  for (i in 2:n)
    y[i] ~ normal(y[i-1] * exp(-beta * t[i]), sigma * sqrt(1 - exp(-beta * t[i]) ^ 2));
  r ~ binomial(200, p);
}
```

Cross-validating time series

```
Source: local data frame [62 x 3]

   parameter    estimate actual
       <chr>       <dbl>  <dbl>
1       beta  0.00000066  0.010
2      sigma  0.00360323  0.250
3       y[1] -0.01654461 -0.122
4       y[2] -0.01654462 -0.082
5       y[3] -0.01654464  0.049
6       y[4] -0.01654465  0.109
7       y[5] -0.01654468 -0.059
8       y[6] -0.01654472 -0.223
9       y[7] -0.01654474 -0.255
10      y[8] -0.01654477 -0.188
..       ...         ...    ...
```

Cross-validating time series

Again, cross validation is in due order

In order to perform leave-one-out, using the 30 datapoints, can use `loo`

However, there are theoretical reasons to prefer **sequential** cross-validation, using first just the first n polls, then polls 2 to n + 1, 3 to n + 2…

This is not implemented in `loo`; leave-one-out should suffice for most purposes