Introduction to Probabilistic Machine Learning with Stan

Daniel Emaasit

2017-04-27

Contents

Introduction

Let's load the required libraries.

```
library(rstan)
library(tidyverse)
library(bayesplot)
library(MASS)
library(gridExtra)
library(hrbrthemes)
rstan_options(auto_write = TRUE) #To write the stan object to the hard disk using saveRDS
options(mc.cores = parallel::detectCores())
```

The raw data

Let's use the Auto MPG Data Set from the UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/datasets/Auto+MPG

```
mpg_data <- read_csv("data/auto_mpg_data.csv") %>% na.omit()
head(mpg_data)
```

Parametric Bayesian Methods

Bayesian linear model

The linear model is given by

$$P(y_i|\mathbf{x}_i) = \mathcal{N}(\mu, \sigma^2)$$

$$\mu = \mathbb{E}(y|\mathbf{x}) = \beta_0 + \mathbf{x}\beta$$

Step 1: Prepare the input data

```
library(tidyverse)

mpg_data <- read_csv("data/auto_mpg_data.csv") %>% na.omit()
mpg_data$id <- 1:nrow(mpg_data)
# mpg_data <- sample_frac(mpg_data, 0.09) # small sample
train <- sample_frac(mpg_data, 0.7)
test  <- anti_join(mpg_data, train, by = 'id')

y <- train$mpg
x <- as.matrix(train[, 3])
N <- nrow(x)
D <- ncol(x)
x_pred <- as.matrix(test[, 3])
N_pred <- nrow(x_pred)

my_data <- list(y = y, x = x, N = N, D = D, x_pred = x_pred, N_pred = N_pred)
```

Step 2: Build the model

The model is prepared in a separate stan file named "linear_bayes.stan"

```
linear_bayes <- "
data {
  int<lower = 1> D;      // dim of dataset
  int<lower=1> N;       // sample size of training set
  matrix[N, D] x;      // obs of training set
  vector[N] y;         // response of training set

  int<lower=1> N_pred;       // sample size of test set
  matrix[N_pred, D] x_pred;     // obs of test set
}
```

```
parameters {
  real intercept;
  vector[D] slope;
  real<lower=0> sigma;
}
model {
  intercept ~ normal(0, 10);     // prior on intercept
  slope ~ normal(0, 10);     // prior
  sigma ~ cauchy(0, 10);   // prior

  y ~ normal(x * slope + intercept, sigma);
}
generated quantities {
  vector[N_pred] y_pred;
  for (n in 1:N_pred)
    y_pred[n] = normal_rng(x_pred[n] * slope + intercept, sigma);
}
"
```

Step 3: Translate and compile the model

Translate the Stan program to C++ code and compile the C++ code to create a dynamic shared object (DSO) that can be loaded by R.

```
model_compiled_lm <- stan_model(file = "linear_bayes.stan",
                                     model_name = "linear_model")
```

Step 4: Sample from the posterior

```
fit_lm <- sampling(model_compiled_lm, data = my_data,
                         iter = 2000, chains = 4, cores = getOption("mc.cores", 1L),
                      control = list(adapt_delta = 0.999, stepsize = 0.001, max_treedepth = 15))
#saveRDS(fit_lm, file = "fit_lm.RDS")
```

Step 5: Evaluate & criticize the results

```
fit_lm <-readRDS("fit_lm.RDS")
list_of_draws <- rstan::extract(fit_lm)
print(names(list_of_draws))
print(fit_lm)
```

Step 5.1: Predictive Accuracy

```
y_pred <- colMeans(list_of_draws$y_pred) %>% as.data.frame()
y_actual <- test$mpg
y_combined <- data.frame(y_pred = y_pred, y_actual = y_actual)

# Function that returns Root Mean Squared Error
rmse <- function(y_observed, y_predicted) {
  error <- y_observed - y_predicted
```

```r
  sqrt(mean(error^2))
}

# Function that returns Mean Absolute Error
mae <- function(y_observed, y_predicted) {
  error_abs <- abs(y_observed - y_predicted)
  colnames(error_abs) <- "absolute_error"
  mean(error_abs$absolute_error)
}

error1 <- rmse(y_observed = y_actual, y_predicted = y_pred)
error2 <- mae(y_observed = y_actual, y_predicted = y_pred)
error1
error2
```

Step 5.2: Visualize results

Visualize the model itself ontop of the test data

```r
p1 <- ggplot(test, aes(test$displacement, test$mpg)) +
  geom_point(aes(colour = 'Test data')) +
  geom_abline(aes(intercept = 34.48, slope = -0.06, colour = 'Posterior linear function')) +
  theme_bw() + theme(legend.position = "bottom") +
  xlab("x = displacement") +
  ylab("y = mpg") +
  scale_color_manual(name = '', values = c('Test data'='black',
                                           'Posterior linear function'= 'blue')) +
  ggtitle("Bayesian Linear Regression",
          subtitle = "The estimated parameters are used to fit a linear model the test data.\n RMSE = 4
p1
```

Visualise distributions of MCMC draws

```r
library("bayesplot")
color_scheme_set("brightblue")
array_of_draws <- as.array(fit_lm)
mcmc_intervals(array_of_draws, pars = c("intercept", "slope[1]", "sigma")) +
  ggtitle("Intervals of parameters from Bayesian Linear Regression",
          subtitle = "")
```

```r
mcmc_areas(
  array_of_draws,
  pars = c("intercept", "slope[1]", "sigma"),
  prob = 0.8, # 80% intervals
  prob_outer = 0.99, # 99%
  point_est = "mean"
) +
  ggtitle("Distribution of parameters from Bayesian Linear Regression",
          subtitle = "")
```

```r
mcmc_dens(array_of_draws,
          pars = c("intercept", "slope[1]", "sigma"),
          facet_args = list(labeller = ggplot2::label_parsed)) +
  ggtitle("Density plots of parameters from Bayesian Linear Regression",
          subtitle = "")
```

Step 5.3: Diagnose MCMC draws

```
color_scheme_set("mix-blue-red")
mcmc_trace(array_of_draws, pars = c("intercept", "slope[1]", "sigma"),
           facet_args = list(ncol = 1, strip.position = "left"))
```

```
n_ratios <- neff_ratio(fit_lm)
#print(n_ratios)
mcmc_neff(n_ratios)
```

```
mcmc_neff_hist(n_ratios)
```

Nonparametric Bayesian Methods

Gaussian Process Joint Hyperparameter Fitting and Predictive Inference

Step 1: Prepare the input data

```
library(tidyverse)

mpg_data <- read_csv("data/auto_mpg_data.csv") %>% na.omit()
mpg_data$id <- 1:nrow(mpg_data)
mpg_data <- sample_frac(mpg_data, 0.5) # small sample
train <- sample_frac(mpg_data, 0.7)
test  <- anti_join(mpg_data, train, by = 'id')

y <- train$mpg
x <- as.matrix(train[, 3])
N <- nrow(x)
D <- ncol(x)
x_pred <- as.matrix(test[, 3])
N_pred <- nrow(x_pred)

my_data <- list(y = y, x = x, N = N, D = D, x_pred = x_pred, N_pred = N_pred)
```

Step 2: Build the model

The model is prepared as a "Stan program" in a separate stan file named "gp_model.stan". The specified Stan program encodes a joint hyperparameter fit and predictive inference model, by declaring the hyperparameters as additional parameters and giving them priors.

```
gp_model <- "
data {
  int<lower=1> N;
  int<lower=1> D;
  int<lower=1> N_pred;
  vector[N] y;
  vector[D] x[N];
  vector[D] x_pred[N_pred];
}
parameters {
  real<lower=1e-12> length_scale;
```

```
  real<lower=0> alpha;
  real<lower=1e-12> sigma;
  vector[N] eta;
}
transformed parameters {
  vector[N] f;
  {
     matrix[N, N] L_cov;
     matrix[N, N] cov;
     cov = cov_exp_quad(x, alpha, length_scale);
     for (n in 1:N)
       cov[n, n] = cov[n, n] + 1e-12;
     L_cov = cholesky_decompose(cov);
     f = L_cov * eta;
  }
}
model {
  length_scale ~ student_t(4,0,1); # (df, mean, sd)
  alpha ~ normal(0, 1);
  sigma ~ normal(0, 1);
  eta ~ normal(0, 1);
  y ~ normal(f, sigma);
}
"
```

Step 3: Translate and compile the model

Translate the Stan program to C++ code and compile the C++ code to create a dynamic shared object (DSO) that can be loaded by R.

```
model_compiled <- stan_model(file = "gp_model.stan",
                                     model_name = "gp_model")
```

Step 4: Sample from the posterior

Run the DSO to sample from the posterior distribution.

```
fit <- sampling(model_compiled, data = my_data,
                          iter = 200, chains = 4, cores = getOption("mc.cores", 1L),
                       control = list(adapt_delta = 0.999, stepsize = 0.001, max_treedepth = 15))
#saveRDS(fit, file = "fit.RDS")
```

Step 5: Evaluate & criticize the results

```
fit <- readRDS("fit.RDS")
list_of_draws <- rstan::extract(fit)
print(names(list_of_draws))
```

```
tidy_fit <- broom::tidy(fit, estimate.method = "mean", conf.int = TRUE, conf.level = 0.80, conf.method =
  rhat = TRUE, ess = TRUE)
saveRDS(tidy_fit, "tidy_fit.RDS")
print(fit)
```

Step 5.1: Predictive Accuracy

```
library(magrittr)
y_pred <- colMeans(list_of_draws$y_pred) %>% as.data.frame()
y_actual <- test$mpg
y_combined <- data.frame(y_pred = y_pred, y_actual = y_actual)

# Function that returns Root Mean Squared Error
rmse <- function(y_observed, y_predicted) {
  error <- y_observed - y_predicted
  sqrt(mean(error^2))
}

# Function that returns Mean Absolute Error
mae <- function(y_observed, y_predicted) {
  error_abs <- abs(y_observed - y_predicted)
  colnames(error_abs) <- "absolute_error"
  mean(error_abs$absolute_error)
}

error1 <- rmse(y_observed = y_actual, y_predicted = y_pred)
error2 <- mae(y_observed = y_actual, y_predicted = y_pred)
error1
error2
```

Step 5.2: Visualise Results

Posterior distribution of fitted parameters

Visualise distributions of MCMC draws

```
array_of_draws <- as.array(fit)
mcmc_dens(array_of_draws,
          pars = c("alpha", "sigma"),
          facet_args = list(labeller = as_labeller(c(
                    `alpha` = "Signal variance",
                    `sigma` = "Noise variance"
                    )))) +
  ggtitle("Posterior distribution of Hyperparameters",
         subtitle = "from Bayesian Nonparametric Regression") +
  theme_ipsum(grid="Y") +
  theme(legend.position="none",
        axis.text.x = element_text(size = 16, colour = "black"),
        axis.text.y = element_text(size = 16, colour = "black"),
        plot.title = element_text(size = 18, colour = "black", face = "bold"),
        plot.subtitle = element_text(size = 14, colour = "black"),
        axis.title.x = element_text(size = 16),
        strip.text.x = element_text(size = 16),
        axis.title.y = element_text(size = 16))
# + scale_x_continuous(breaks = c(0.3,0.4,0.5))
```

```
mcmc_dens(array_of_draws,
          pars = c("length_scale"),
          facet_args = list(labeller = as_labeller(c(
                    `length_scale` = "Length scale"
                    )))) +
  ggtitle("Posterior distribution of Hyperparameters",
          subtitle = "from Bayesian Nonparametric Regression") +
  theme_ipsum(grid="Y") +
  theme(legend.position="none",
        axis.text.x = element_text(size = 16, colour = "black"),
        axis.text.y = element_text(size = 16, colour = "black"),
        plot.title = element_text(size = 18, colour = "black", face = "bold"),
        plot.subtitle = element_text(size = 14, colour = "black"),
        axis.title.x = element_text(size = 16),
        strip.text.x = element_text(size = 16),
        axis.title.y = element_text(size = 16)) +
  xlab('Estimate') +
  ylab('Density')
```

Posterior predictive distribution

```
library(reshape2)
post_pred <- data.frame(x = test$displacement, y_pred = colMeans(list_of_draws$y_pred),
                        y_actual = test$mpg)
post_mu_fs <- data.frame(x = test$displacement, y = t(list_of_draws$y_pred))
#post_mu_fs <- post_mu_fs[,1:6]
post_mu_fs_melt <- melt(post_mu_fs, id.vars = "x")

library(ggplot2)
p1 <- ggplot(data = post_pred, aes(x = x, y = y_actual)) +
  geom_line(data = post_mu_fs_melt, aes(x = x, y = value, group = variable,
                                        colour = 'Posterior functions'), alpha = 0.15) +
  theme_bw() + theme(legend.position="bottom") +
  geom_line(data = post_pred, aes(x = x, y = y_pred, colour = 'Posterior mean function')) +
  theme_bw() + theme(legend.position = "bottom") +
  geom_point(aes(colour = 'Realized data')) +
  scale_color_manual(name = '', values = c('Realized data'='black',
                                           'Posterior functions'= 'blue',
                                           'Posterior mean function'='red')) +
  xlab('x = displacement') +
  ylab('y = mpg') +
  ggtitle("Bayesian Nonparametric Regression",
          subtitle = paste0('Posterior predictive distribution with N = ',length(t(x_pred)),', length-s
p1
```

Step 5.3: Diagnose MCMC draws

```
color_scheme_set("mix-blue-red")
mcmc_trace(array_of_draws, pars = c("length_scale", "alpha", "sigma"),
           facet_args = list(ncol = 1, strip.position = "left"))
```

```
n_ratios <- neff_ratio(fit)
#print(n_ratios)
```

```
mcmc_neff(n_ratios)
```

```
mcmc_neff_hist(n_ratios)
```

Appendix

References

- *Website:* http://mc-stan.org/
- *Stan Manual(v2.14):* https://github.com/stan-dev/stan/releases/download/v2.14.0/stan-reference-2.14.0.pdf
- *RStan:* https://cran.r-project.org/web/packages/rstan/vignettes/rstan.html
- *STANCON 2017 Intro Course Materials:* https://t.co/6d3omvBkrd
- *Statistical Rethinking* by R. McElreath: http://xcelab.net/rm/statistical-rethinking/
- *Mailing list:* https://groups.google.com/forum/#!forum/stan-users
- Winn, J., Bishop, C. M., Diethe, T. (2015). Model-Based Machine Learning. Microsoft Research Cambridge. http://www.mbmlbook.com.

Compare with Frequentist linear model

```
library(caret)
##fit a linear model using "lm" method from caret package
lm_fit<-train(mpg ~ displacement, method="lm", data = train)
##then use the model to predict new values
lm_predict<-predict(lm_fit, newdata = test)
rmse(y_actual, y_pred = as.data.frame(lm_predict))
mae(y_actual, y_pred = as.data.frame(lm_predict))
```

```
postResample(pred = lm_predict, obs = test$mpg) ## my rmse is the same as from caret
```

```
library(ggplot2)
ggplot(data = train, aes(x = displacement, y = mpg)) +
  geom_point() +
  geom_smooth(aes(color = "Polynomial"), method = "lm",
              formula = (formula= (y ~ poly(x, 4))))
```