Lab 4

Alex Ziyu Jiang

(The code materials are kindly provided by Professor Carlos Cinelli.)

Bayesian multiple linear regression: Saratoga Housing example

Today we look at an example of Bayesian multiple linear regression and show how to implement it using sampling softwares such as jags and rstan. Moreover, under a specific setting we show the relationship between Bayesian multiple linear regression and ridge regression, a frequentist regularization method commonly used in machine learning.

model prerequisites

As usual, we load the package we need (remember to install them first using `install.packages()`) if you haven't done so:

```
rm(list = ls())
library(rjags)
```

```
## Loading required package: coda
```

```
## Linked to JAGS 4.3.0
```

```
## Loaded modules: basemod,bugs
```

```
library(rstan)
```

```
## Loading required package: StanHeaders
```

```
## Loading required package: ggplot2
```

```
## rstan (Version 2.21.3, GitRev: 2e1f913d3ca3)
```

```
## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)
```

```
##
## Attaching package: 'rstan'
```

```
## The following object is masked from 'package:coda':
##
##     traceplot
```

```
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 4.1.2
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-4
```

```r
library(rethinking)
```

```
## Loading required package: cmdstanr

## This is cmdstanr version 0.5.1.9000

## - CmdStanR documentation and vignettes: mc-stan.org/cmdstanr

## - CmdStan path: /Users/alexziyujiang/.cmdstan/cmdstan-2.29.2

## - CmdStan version: 2.29.2

## Loading required package: parallel

## rethinking (Version 2.21)

##
## Attaching package: 'rethinking'

## The following object is masked from 'package:rstan':
##
##     stan

## The following object is masked from 'package:stats':
##
##     rstudent
```

```r
options(scipen = 99)
```

data preprocessing

Then we load in the dataset and do some preprocessing to make the data suitable for our analysis. Just as a recap: for Bayesian regression models, standardizing (centering and rescaling by standard deviation) the predictors will lead to better MCMC sampling efficiency(Markov chains converge to equilibrium quicker) and easier prior choices, so here we standardize the columns in the data matrix.

```r
# Load data ---------------------------------------------------------------
df <- read.csv("SaratogaHouses.csv")

# create matrix
x <- model.matrix(price ~ ., data = df)

# scale variables (except constant)
x.scale <- x
x.scale[,-1] <- apply(x.scale[,-1], 2, function(z) (z - mean(z))/sd(z))
y.scale <- df$price/sd(df$price)

# save sd's to rescale back
sdx <- apply(x[,-1], 2, sd)
sdy <- sd(df$price)

# function to rescale coefficients
rescale <- function(beta){
  beta <- beta*sdy/sdx
  names(beta) <- colnames(x)[-1]
  beta
}
```

model form

Recall that here we are building a multivariate linear regression model. If the model matrix is $\mathbf{X}$, the vector of regression coefficients are $\boldsymbol{\beta}$, the model has the following form:

$$
\begin{aligned}
y_i &\sim \text{Normal}\left(\mu_i, \sigma^2\right), i = 1, ..., n \\
\boldsymbol{\mu} &= \mathbf{X}\boldsymbol{\beta} \\
\beta_0 &\sim \text{Normal}(\mu_0, s_0^2) \\
\beta_j &\sim \text{Normal}(\mu_\beta, s_\beta^2), j = 1, ..., p \\
\sigma &\sim \text{Exponential}(\lambda)
\end{aligned}
$$

Here μ_β and s_β^2 represents the mean and variance for the regression coefficients β_j, and λ is the rate coefficient of the exponential prior for σ. Note that μ_β, s_β^2 and λ are 'parameters' of the prior distributions, so instead of placing prior on them we feed them actual values – these parameters are called **hyperparameters** in Bayesian statistics. We choose $\lambda = 1$, a relatively 'flat' prior for the standard deviation. For the regression coefficients, we choose a bunch of normal priors with mean zero, because we don't really have prior knowledge about how each effect will look like before fitting the data. For the intercept model, we place a flat normal prior with standard deviation $s_0 = 1,000$. For the other variables, we place a 'tighter' regularization prior on all of these variables with standard deviation $s_\beta = 0.02$.

model implementation

JAGS

Finally, we compile the JAGS and STAN code for model implementation (note the notational difference regarding the normal variance/precision):

```
# JAGS -------------------------------------------------------------------

# generic model code
linear_model_code <- "
  data{
    D <- dim(x)
    n <- D[1]
    p <- D[2]
  }
  model{
   for(i in 1:n){
      # likelihood
      y[i] ~ dnorm(mu[i], tau)
      # # posterior predictive
      # ynew[i] ~ dnorm(mu[i], tau)
   }
    # conditional mean using matrix algebra
    mu <- x %*% beta
    for(j in 1:p){
      beta[j] ~ dnorm(mb[j], pow(sb[j], -2))
    }
    sigma ~ dexp(lambda)
    tau <- pow(sigma, -2)
  }
"

# flat prior for constant
```

```r
# tight regularizing priors for all other parameters
model <- jags.model(file = textConnection(linear_model_code),
                    data = list(x = x.scale,
                                y = y.scale,
                                mb = rep(0, ncol(x)),
                                sb = c(1000, rep(.02, ncol(x)-1)),
                                lambda = 1))
```

```
## Compiling data graph
##    Resolving undeclared variables
##    Allocating nodes
##    Initializing
##    Reading data back into data table
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 1728
##    Unobserved stochastic nodes: 20
##    Total graph size: 36359
##
## Initializing model
```

```r
nsim <- 5e3
# burn in
update(model, n.iter = nsim)

# samples
samps <- coda.samples(model = model, n.iter = nsim,
                      variable.names = c("beta", "sigma"))
# check trace plots
# plot(samps)

# transform back to original scale
samps.df <- as.data.frame(samps[[1]])
post.means <- apply(samps.df, 2, mean)[-c(1,20)]
post.means <- rescale(post.means)
post.means
```

```
##                lotSize                    age              landValue
##           7432.0780941           -113.1577644              0.6412473
##             livingArea             pctCollege               bedrooms
##             35.5364074            144.2256107           3464.7609200
##             fireplaces              bathrooms                  rooms
##           8884.6695971          22264.5847407           4423.9892157
##         heatinghot air heatinghot water/steam                fuelgas
##           6488.3455481          -1972.1805342           5584.9854936
##                fueloil sewerpublic/commercial            sewerseptic
##           -100.4664115            184.6637635           -436.5662278
##          waterfrontYes     newConstructionYes          centralAirYes
##          94370.7441953         -10778.6115139          12958.2920648
```

We run the model and generate 5,000 posterior samples for $\boldsymbol{\beta}$ and σ. We can use the posterior samples for $\boldsymbol{\beta}$ to calculate its posterior mean. Finally we transform it back to the original scale for clearer interpretation (in the sense that 'the rate of change' is associated with unit change in the actual variables).

STAN

Similarly, we could do the stan version:

```
## Running MCMC with 1 chain...
##
## Chain 1 Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 1 Iteration:  100 / 5000 [  2%]  (Warmup)

## Chain 1 Informational Message: The current Metropolis proposal is about to be rejected because of the

## Chain 1 Exception: normal_lpdf: Scale parameter is 0, but must be positive! (in '/var/folders/42/k2fw

## Chain 1 If this warning occurs sporadically, such as for highly constrained variable types like covar

## Chain 1 but if this warning occurs often then your model may be either severely ill-conditioned or mi

## Chain 1

## Chain 1 Iteration:  200 / 5000 [  4%]  (Warmup)
## Chain 1 Iteration:  300 / 5000 [  6%]  (Warmup)
## Chain 1 Iteration:  400 / 5000 [  8%]  (Warmup)
## Chain 1 Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 1 Iteration:  600 / 5000 [ 12%]  (Warmup)
## Chain 1 Iteration:  700 / 5000 [ 14%]  (Warmup)
## Chain 1 Iteration:  800 / 5000 [ 16%]  (Warmup)
## Chain 1 Iteration:  900 / 5000 [ 18%]  (Warmup)
## Chain 1 Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 1 Iteration: 1100 / 5000 [ 22%]  (Warmup)
## Chain 1 Iteration: 1200 / 5000 [ 24%]  (Warmup)
## Chain 1 Iteration: 1300 / 5000 [ 26%]  (Warmup)
## Chain 1 Iteration: 1400 / 5000 [ 28%]  (Warmup)
## Chain 1 Iteration: 1500 / 5000 [ 30%]  (Warmup)
## Chain 1 Iteration: 1600 / 5000 [ 32%]  (Warmup)
## Chain 1 Iteration: 1700 / 5000 [ 34%]  (Warmup)
## Chain 1 Iteration: 1800 / 5000 [ 36%]  (Warmup)
## Chain 1 Iteration: 1900 / 5000 [ 38%]  (Warmup)
## Chain 1 Iteration: 2000 / 5000 [ 40%]  (Warmup)
## Chain 1 Iteration: 2100 / 5000 [ 42%]  (Warmup)
## Chain 1 Iteration: 2200 / 5000 [ 44%]  (Warmup)
## Chain 1 Iteration: 2300 / 5000 [ 46%]  (Warmup)
## Chain 1 Iteration: 2400 / 5000 [ 48%]  (Warmup)
## Chain 1 Iteration: 2500 / 5000 [ 50%]  (Warmup)
## Chain 1 Iteration: 2501 / 5000 [ 50%]  (Sampling)
## Chain 1 Iteration: 2600 / 5000 [ 52%]  (Sampling)
## Chain 1 Iteration: 2700 / 5000 [ 54%]  (Sampling)
## Chain 1 Iteration: 2800 / 5000 [ 56%]  (Sampling)
## Chain 1 Iteration: 2900 / 5000 [ 58%]  (Sampling)
## Chain 1 Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 1 Iteration: 3100 / 5000 [ 62%]  (Sampling)
## Chain 1 Iteration: 3200 / 5000 [ 64%]  (Sampling)
## Chain 1 Iteration: 3300 / 5000 [ 66%]  (Sampling)
## Chain 1 Iteration: 3400 / 5000 [ 68%]  (Sampling)
## Chain 1 Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 1 Iteration: 3600 / 5000 [ 72%]  (Sampling)
## Chain 1 Iteration: 3700 / 5000 [ 74%]  (Sampling)
## Chain 1 Iteration: 3800 / 5000 [ 76%]  (Sampling)
## Chain 1 Iteration: 3900 / 5000 [ 78%]  (Sampling)
```

```
## Chain 1 Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 1 Iteration: 4100 / 5000 [ 82%]  (Sampling)
## Chain 1 Iteration: 4200 / 5000 [ 84%]  (Sampling)
## Chain 1 Iteration: 4300 / 5000 [ 86%]  (Sampling)
## Chain 1 Iteration: 4400 / 5000 [ 88%]  (Sampling)
## Chain 1 Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 1 Iteration: 4600 / 5000 [ 92%]  (Sampling)
## Chain 1 Iteration: 4700 / 5000 [ 94%]  (Sampling)
## Chain 1 Iteration: 4800 / 5000 [ 96%]  (Sampling)
## Chain 1 Iteration: 4900 / 5000 [ 98%]  (Sampling)
## Chain 1 Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 1 finished in 3.8 seconds.
```

```
stan.means <- apply(as.data.frame(m.stan), 2, mean)[-c(1,20,21)]
stan.means <- rescale(stan.means)
stan.means
```

```
##               lotSize                   age             landValue
##          7382.6122462          -111.6198372             0.6423868
##            livingArea            pctCollege              bedrooms
##            35.5053727           143.8605749          3559.4846901
##            fireplaces             bathrooms                 rooms
##          8906.9120395         22184.2658244          4409.4458256
##        heatinghot air heatinghot water/steam               fuelgas
##          6406.9875993         -2049.3089260          5656.3703135
##               fueloil sewerpublic/commercial           sewerseptic
##           -58.0251741            67.9907910          -487.1475680
##         waterfrontYes    newConstructionYes          centralAirYes
##         94308.0033961        -10502.0987866         12961.8292228
```

QUAP() from the textbook

Finally, the textbook has a fancy function called `quap()`, to approximate the posterior distributions under regression settings. We repeat the similar analysis:

```
# Quadratic Approximation -------------------------------------------------
# using quadratic approximation (your book)
model.quap <- quap(flist = alist(
  y ~ dnorm(mu, sigma),
  mu <- alpha + x %*% beta,
  alpha ~ dnorm(0, 1000),
  beta ~ dnorm(0, 0.02),
  sigma ~ dexp(1)),
  data = list(x = x.scale[,-1],
              y = y.scale),
  start = list(beta = rep(0,ncol(x)-1)))
# transform back
quap.coef <- coef(model.quap)[-c(19, 20)]
quap.coef <- rescale(quap.coef)
quap.coef
```

```
##               lotSize                   age             landValue
##          7402.5861991          -112.6974093             0.6430178
##            livingArea            pctCollege              bedrooms
##            35.6413884           144.3355290          3462.2684098
##            fireplaces             bathrooms                 rooms
```

```
##          8889.9119816          22212.2662579            4422.3961462
##         heatinghot air heatinghot water/steam                 fuelgas
##          6454.1508202          -2020.5862189            5663.4477399
##              fueloil sewerpublic/commercial             sewerseptic
##           -72.3232879            108.4333332            -484.0987777
##         waterfrontYes     newConstructionYes           centralAirYes
##         94543.4351029         -10759.8934385           12941.0703763
```

Ridge regression and Bayesian linear regression

Let's think about frequentist linear regression model for a moment. The ordinary least squares estimate of a linear model can be reframed as an optimization problem:

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} (y - X\beta)^T (y - X\beta).$$

As a remedy to model overfitting problems, ridge regression is a commonly used regularization method that tends to reduce the magnitude of each predictor variable in the model. To do ridge regression, we simply add an extra penalty term $\lambda \|\beta\|_2^2$ that penalizes the Euclidean norm of the vector of coefficient. Here λ is a hyperparameter (a different 'hyperparameter' than the one in Bayesian statistics as there is no prior here) that controls how much you want to penalize the vector of coefficients.

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} (y - X\beta)^T (y - X\beta) + \lambda \|\beta\|_2^2,$$

There is a Bayesian interpretation to the ridge regression framework: for a Bayesian linear regression model with fixed residual variance σ^2 and independent Gaussian prior $\mathcal{N}(0, \tau^2 \mathbf{I}_q)$ on the regression coefficients $\boldsymbol{\beta}$, the posterior mode for $\boldsymbol{\beta}$, $p(\boldsymbol{\beta} \mid \mathbf{X}, \sigma^2, \tau^2)$ corresponds to the ridge regression estimate with $\lambda = \frac{\sigma^2}{\tau^2}$. We won't get into the reasoning behind this, but you can refer to here for more detail: https://statisticaloddsandends.wordpress.com/2018/12/29/bayesian-interpretation-of-ridge-regression/.

We will first fit the ridge regression estimates using functions in package `glmnet`. A couple of things to notice:

- 'alpha = 0' means we are doing the ridge regression (unrelated to today's material, but if you set alpha to be 1, we get lasso instead).
- Since the model we are considering is a little different from the setting above, we will not be getting exactly the same estimates(also, we are actually using posterior mean instead of posterior mode), but they should be similar.
- We will also be computing estimates without penalizing to show the difference between these estimates.

```r
# OLS and Ridge ----------------------------------------------------------


# fit Ridge for comparison
gl.out <- glmnet(x = x.scale[,-1], y = y.scale, standardize = F, intercept = T, alpha = 0, lambda = 0.45
gl.coef <- coef(gl.out)[-1]

# transform back to original scale
gl.coef <- rescale(gl.coef)
gl.coef
```

```
##               lotSize                    age               landValue
##          7599.0529985           -114.5099914               0.6813298
##            livingArea             pctCollege                bedrooms
##            38.0590035            121.5486238            2686.8866930
##            fireplaces              bathrooms                   rooms
```

```
##           8408.1669380          22905.8049761           4431.8754235
##          heatinghot air heatinghot water/steam                fuelgas
##           6430.8509086          -2289.0060878           5520.2448287
##                fueloil sewerpublic/commercial            sewerseptic
##             71.6557607             -9.8817011           -360.9500877
##          waterfrontYes     newConstructionYes          centralAirYes
##          99426.1087329         -13930.7062043          12915.6685082
```

```
# fit lm for comparison
lm.coef <- coef(lm(price ~ ., data = df))[-1]

# compare estimates
round(cbind(`lm (not regularized)` = lm.coef,
      jags = post.means,
      stan = stan.means,
      quap = quap.coef,
      glmnet = gl.coef),3)
```

```
##                          lm (not regularized)       jags       stan       quap
## lotSize                              7599.449   7432.078   7382.612   7402.586
## age                                  -130.446   -113.158   -111.620   -112.697
## landValue                               0.922      0.641      0.642      0.643
## livingArea                             69.960     35.536     35.505     35.641
## pctCollege                           -110.159    144.226    143.861    144.336
## bedrooms                            -7835.192   3464.761   3559.485   3462.268
## fireplaces                           1036.613   8884.670   8906.912   8889.912
## bathrooms                           23112.452  22264.585  22184.266  22212.266
## rooms                                3019.761   4423.989   4409.446   4422.396
## heatinghot air                         82.452   6488.346   6406.988   6454.151
## heatinghot water/steam             -10372.246  -1972.181  -2049.309  -2020.586
## fuelgas                             10931.274   5584.985   5656.370   5663.448
## fueloil                              6550.471   -100.466    -58.025    -72.323
## sewerpublic/commercial               3321.168    184.664     67.991    108.433
## sewerseptic                          4845.107   -436.566   -487.148   -484.099
## waterfrontYes                      120193.978  94370.744  94308.003  94543.435
## newConstructionYes                 -45443.421 -10778.612 -10502.099 -10759.893
## centralAirYes                        9953.091  12958.292  12961.829  12941.070
##                             glmnet
## lotSize                   7599.053
## age                       -114.510
## landValue                    0.681
## livingArea                  38.059
## pctCollege                 121.549
## bedrooms                  2686.887
## fireplaces                8408.167
## bathrooms                22905.805
## rooms                     4431.875
## heatinghot air            6430.851
## heatinghot water/steam   -2289.006
## fuelgas                   5520.245
## fueloil                     71.656
## sewerpublic/commercial      -9.882
## sewerseptic               -360.950
## waterfrontYes            99426.109
## newConstructionYes      -13930.706
```

```
## centralAirYes           12915.669
```

Conclusion

- Bayesian linear regression can be viewed as a regularization method
- Some concluding remarks on covariate standardizing: centering and rescaling (1) helps sampling and (2) helps choosing priors:
  - In ridge regression we standardize the covariates and give them the same 'penalty term', for the Bayesian equivalent, instead of doing the penalty term we place a tight prior with large precision around zero for all of the variables
  - For the 'common penalty' (in terms of the tight prior) to make sense, we need to do the standardization