Comparing multiple models is one of the core but also one of the trickiest element of data analysis. Under a Bayesian framework the [loo](https://cran.r-project.org/web/packages/loo/index.html) package in R allows you to derive (among other things) leave-one-out cross-validation metrics to compare the predictive abilities of different models.

Cross-validation is basically: (i) separating the data into chunks, (ii) fitting the model while holding out one chunk at a time, (iii) evaluating the probability density of the held-out chunk of data based on the parameter estimates, (iv) derive some metrics from the likelihood of the held-out data. The basic idea is that if your model does a good job, then it can predict the value of held-out data pretty well.

We can separate the data into a different number of chunks, or folds, from 2 up to N, the number of data points. Holding out one data point at a time is called leave-one-out cross-validation and can be computationally costly (you need to fit the model N times), fortunately [Vehtari and colleagues](https://arxiv.org/pdf/1507.04544.pdf) have been developing methods to estimate leave-one-out cross validation without having to do expensive computation. Now, as with every estimation, it sometime cannot give reliable results and one could/should then fall back to standard K-fold cross-validation by fitting the model multiple time.

The aim of this post is to show one simple example of K-fold cross-validation in Stan via R, so that when loo cannot give you reliable estimates, you may still derive metrics to compare models.

**The Stan code**

Below is the Stan code for a simple linear normal regression allowing K-fold cross-validation

/\*

Standard normal regression for any number of predictor variables

with weakly informative priors on the betas and on the standard deviation

\*/

data{

int N; //number of observations

int K; //number of predictor variables

matrix[N,K] X; //the model matrix including intercept

vector[N] y; //the response variable

int holdout[N];

//index whether the observation should be held out (1) or used (0)

}

parameters{

vector[K] beta; //the regression parameters

real sigma;

}

model{

vector[N] mu; //the linear predictor

mu = X \* beta; //the regression

//priors

beta[1] ~ normal(0,10);

beta[2:K] ~ normal(0,5);

sigma ~ normal(0,10);

//likelihood holding out some data

for(n in 1:N){

if(holdout[n] == 0){

target += normal\_lpdf(y[n] | mu[n], sigma);

}

}

}

generated quantities{

vector[N] log\_lik;

for (n in 1:N){

log\_lik[n] = normal\_lpdf(y[n] | X[n,] \* beta, sigma);

}

}

The key element here is to pass as data a vector of 0s and 1s as long as the observed data indicating whether each point should be held-out during model estimation or not. Estimating the likelihood is done by going through each data point, asking if it should be held-out, if not then this data point is used to increment the likelihood based on the model parameter.

Below in the generated quantities segment, we are retrieving the probability density (the height of the probability distribution) for each data, also the one held-out.

The best way to go on if you want to try out the code is to copy/paste the model code into a .stan file.

**Simulating some data and fitting the models**

Let's start by setting some basic quantities and creating the held-out indeces.

#first load the libraries

library(rstan)

rstan\_options(auto\_write = TRUE)

library(pbmcapply)

library(loo)

N <- 100 #sample size

K <- 2 #number of predictors

n\_fold <- 10 #number of folds

#create 10 folds of data

hh <- sample(1:N,size = N,replace = FALSE)

holdout\_10 <- matrix(0,nrow=N,ncol=n\_fold)

for(i in 1:n\_fold){

id <- seq(1,100,by=10)

holdout\_10[hh[id[i]:(id[i] + 9)],i] <- 1

}

#some sanity checks

#apply(holdout\_10,1,sum)

#apply(holdout\_10,2,sum)

#turn into a list

holdout\_10 <- split(holdout\_10,rep(1:ncol(holdout\_10),each=nrow(holdout\_10)))

Randomly assigning each data point to a different fold is the trickiest part of the data preparation in K-fold cross-validation. What I basically did is randomly sample N times with no replacement from the data point index (the object hh), and put the first 10 index in the first fold, the subsequent 10 in the second fold and so on.

Note that if there is some kind of grouping structure in your data that you want to be taken care of while splitting it, it might get a bit more complex, there is certainly some clever functions out there that might help you out (such as in the [caret](https://cran.r-project.org/web/packages/caret/index.html) package).

Now we can simulate some data and create the data object to be passed to Stan, already note that we will need one data object per fold, so a list might be an easy way to combine these:

X <- cbind(rep(1,N),runif(N,-2,2))

y <- rnorm(N,X %\*% c(1, 0.5),1)

#the basic data object

data\_m <- list(N=N,K=K,X=X,y=y)

#create a list of data list

data\_l <- rep(list(data\_m),10)

#add the holdout index to it

for(i in 1:10) data\_l[[i]]$holdout <- holdout\_10[[i]]

We are now ready to fit the model to each fold, we could just loop through the folds but this is inefficient, rather I am going to use the functions available [in this link](https://github.com/stan-dev/stancon_talks/blob/master/2017/Contributed-Talks/07_nicenboim/kfold.Rmd) and pasted at the end of this post. Basically stan\_kfold output a list of stanfit objects (one for each fold), extract\_log\_lik\_K output a S x N matrix where S is the number of posterior draws where each element is the log-likelihood when the data point was held-out and kfold compute the expected log pointwise predictive density elpd. Basically, the elpd is the height (density) of the probability distribution, given the model parameters, at the data point (pointwise) that were held-out (predictive).

#run the functions

ss <- stan\_kfold(file="Documents/PostDoc\_Ghent/STAN\_stuff/Models/normal\_model\_basic\_cv.stan",data\_l,chains=4,cores=2)

ee <- extract\_log\_lik\_K(ss,holdout\_10)

kk <- kfold(ee)

#compare with official loo results

ll <- loo(ee)

The ee matrix can actually also be used as input to loo to get some more metrics.

All this is nice and fine but these metrics are only relative and only truly make sense when comparing between different models fitted tot he same data. So let's fit two additional models, one overly complex and one overly simple:

# fit a too complex and a too simple model

X\_comp <- cbind(X,runif(N,-2,2))

X\_simp <- X[,1,drop=FALSE]

# new data

data\_comp <- data\_l

for(i in 1:10){

data\_comp[[i]]$X <- X\_comp

data\_comp[[i]]$K <- 3

}

data\_simp <- data\_l

for(i in 1:10){

data\_simp[[i]]$X <- X\_simp

data\_simp[[i]]$K <- 1

}

#fit the new models

ss\_comp <- stan\_kfold(file="Documents/PostDoc\_Ghent/STAN\_stuff/Models/normal\_model\_basic\_cv.stan",data\_comp,chains=4,cores=2)

ss\_simp <- stan\_kfold(file="Documents/PostDoc\_Ghent/STAN\_stuff/Models/normal\_model\_basic\_cv.stan",data\_simp,chains=4,cores=2)

ee\_comp <- extract\_log\_lik\_K(ss\_comp,holdout\_10)

ee\_simp <- extract\_log\_lik\_K(ss\_simp,holdout\_10)

#compare the models

compare(loo(ee),loo(ee\_comp),loo(ee\_simp))

*#output:*

*# elpd\_diff elpd\_loo se\_elpd\_loo p\_loo se\_p\_loo looic se\_looic*

*#loo(ee) 0.0 -148.9 9.5 4.3 1.0 297.7 19.1*

*#loo(ee\_comp) -1.8 -150.7 9.6 5.5 1.3 301.4 19.2*

*#loo(ee\_simp) -11.2 -160.1 8.9 2.8 0.8 320.1 17.8*

The compare functions ouput quite some information, let's go through it.

The first column is the difference in the summed expected log pointwise predictive density, the difference between the second and the first model is -1.8, meaning that the first model as slightly higher predictive density. The second column is the summed expected log pointwise predictive density, the values in the first column are differences from this one. The third column is the standard error in the expected log poitwise predictive density. The fourth and fifth columns is the effective number of parameter in the model and its standard error. The sixth column is the -2 \* the second column, putting the elpd on the deviance scale so being similar to other information criteria metrics such as AIC or DIC. And the final column is the standard error of the information criteria.

From this output one could argue that the model with two predictors is definitively superior to the intercept-only model, and his slightly better than the model with 3 parameters.

Voila, do remember that model selection is very tricky, I would not encourage using information criteria or cross-validation metrics to decide whether one should include that particular interaction or this extra covariate. To my mind, these type of comparison are relevant to compare different model structure such as simple regression vs hierarchical models vs spatial effects vs autoregressive structure. Or to compare models based on a different set of covariates such as to compare if plant growth can be better predicted by trait information vs environmental information.

Happy folding

**The function code**

Below is the code of the function used to fit the models and extract the information:

#functions slightly modified from: https://github.com/stan-dev/stancon\_talks/blob/master/2017/Contributed-Talks/07\_nicenboim/kfold.Rmd

#function to parrallelize all computations

#need at least two chains !!!

stan\_kfold <- function(file, list\_of\_datas, chains, cores,...){

library(pbmcapply)

badRhat <- 1.1 # don't know why we need this?

n\_fold <- length(list\_of\_datas)

model <- stan\_model(file=file)

# First parallelize all chains:

sflist <-

pbmclapply(1:(n\_fold\*chains), mc.cores = cores,

function(i){

# Fold number:

k <- ceiling(i / chains)

s <- sampling(model, data = list\_of\_datas[[k]],

chains = 1, chain\_id = i,...)

return(s)

})

# Then merge the K \* chains to create K stanfits:

stanfit <- list()

for(k in 1:n\_fold){

inchains <- (chains\*k - (chains - 1)):(chains\*k)

# Merge `chains` of each fold

stanfit[[k]] <- sflist2stanfit(sflist[inchains])

}

return(stanfit)

}

#extract log-likelihoods of held-out data

extract\_log\_lik\_K <- function(list\_of\_stanfits, list\_of\_holdout, ...){

require(loo)

K <- length(list\_of\_stanfits)

list\_of\_log\_liks <- plyr::llply(1:K, function(k){

extract\_log\_lik(list\_of\_stanfits[[k]],...)

})

# `log\_lik\_heldout` will include the loglike of all the held out data of all the folds.

# We define `log\_lik\_heldout` as a (samples x N\_obs) matrix

# (similar to each log\_lik matrix)

log\_lik\_heldout <- list\_of\_log\_liks[[1]] \* NA

for(k in 1:K){

log\_lik <- list\_of\_log\_liks[[k]]

samples <- dim(log\_lik)[1]

N\_obs <- dim(log\_lik)[2]

# This is a matrix with the same size as log\_lik\_heldout

# with 1 if the data was held out in the fold k

heldout <- matrix(rep(list\_of\_holdout[[k]], each = samples), nrow = samples)

# Sanity check that the previous log\_lik is not being overwritten:

if(any(!is.na(log\_lik\_heldout[heldout==1]))){

warning("Heldout log\_lik has been overwritten!!!!")

}

# We save here the log\_lik of the fold k in the matrix:

log\_lik\_heldout[heldout==1] <- log\_lik[heldout==1]

}

return(log\_lik\_heldout)

}

#compute ELPD

kfold <- function(log\_lik\_heldout) {

library(matrixStats)

logColMeansExp <- function(x) {

# should be more stable than log(colMeans(exp(x)))

S <- nrow(x)

colLogSumExps(x) - log(S)

}

# See equation (20) of @VehtariEtAl2016

pointwise <- matrix(logColMeansExp(log\_lik\_heldout), ncol= 1)

colnames(pointwise) <- "elpd"

# See equation (21) of @VehtariEtAl2016

elpd\_kfold <- sum(pointwise)

se\_elpd\_kfold <- sqrt(ncol(log\_lik\_heldout) \* var(pointwise))

out <- list(

pointwise = pointwise,

elpd\_kfold = elpd\_kfold,

se\_elpd\_kfold = se\_elpd\_kfold)

#structure(out, class = "loo")

return(out)

}