

# Bayes R2 for LM

Silvano

April 18, 2018

## 1 18.04.18

The following  $R^2$  may be useful:

$$+ R^2_{Gelman} = \frac{V(\sum_{n=1}^N \hat{y}_n^s)}{V(\sum_{n=1}^N \hat{y}_n^s + V(\sum_{n=1}^N e_n^s)}, \text{ where } \theta^s, s = 1, \dots, S \text{ are draws from the posterior parameter distribution. For each } \theta^s \text{ we can compute the vector of predicted values } \hat{y}_n^s = E(y | X_n, \theta^s) \text{ and the vector of errors } e_n^s = y_n - \hat{y}_n^s.$$

This  $R^2$  can be interpreted as a data-based estimate of the proportion of variance explained for new data (Gelman). They note that their proposed  $R^2$  overestimates the proportion of explained variance for new data. They argue, a new issue then arises, when fitting a set of models to a single dataset. Now that the denominator of  $R^2$  is no longer fixed, we can no longer interpret an increase in  $R^2$  as a improved fit to a fixed target.

Instead of the above error definition we may as well use samples from the posterior  $\sigma_s$  in the denominator for the linear model. I think this way we would have a better possibility to include prior information in the sigma parameter. In the uninformative prior case this should not make a difference.

- + In the same paper they note: An alternative is to summarize residual error on the log-probability scale, in which case we recommend using approximate leave-one-out cross-validation (Vehtari, 2017), which can be done using the loo function within rstanarm. If desired, changes in expected log-probability scores can be converted back to an  $R^2$  scale as discussed by Nagelkerke (1991).

instead of leave-one out crossvalidation we may just use the whole data:

$$R^2_{LRT} = 1 - \left( \frac{L_{intercept}}{L_{full}} \right)^{\frac{2}{n}}, \text{ where } L = \text{Likelihood or in loglikelihood terms } 1 - \exp(-LRT/n)$$

For the  $L_{full}$  term we can use the vector of predicted values obtained

from the posterior sample. For the  $L_{intercept}$  term we can use the maximum likelihood estimate, the mean, of the data at hand. This  $R^2$  is then lower as if we would use the maximum likelihood for the full model. It then includes the uncertainty of the parameter estimates (the higher the sample size, the more certain we can be).

Even without leave-one out crossvalidation this measure is then substantially lower than the  $R^2_{Gelman}$  and can be negative. I think that it can be negative is not a big problem in this case. It measures geometric mean improvement per observation for new data. It predicts the mean for new data and can therefore be lower than if we would observe it. Using leave one out crossvalidation with maximum likelihood the  $R^2$  can also be negative if the correct  $R^2$  formula is used (Beware of  $R^2$ : simple, unambiguous assessment of the prediction accuracy of QSAR and QSPR models, 2015). A Problem with the LMG formula may arise. However, the non decreasing property when adding predictors is lost in all the bayes formulas described, so this problem is there anyway. A benefit of using the comparison to the fixed mean would be that we are comparing all models to a fixed target.

+ Another option would be to use the classical definition:

$$R^2_{KL} = 1 - \frac{SSE(X)}{SST} \quad (1)$$

$$SSE(X_s) = \sum_{n=1}^n (y_i - \hat{y}(x))^2 \quad (2)$$

$$SST_s = \sum_{n=1}^n (y_i - \bar{y})^2 \quad (3)$$

$SSE(X_s)$  uses the vector of predicted values from the posterior sample,  $SST_s$  uses its sample from the posterior intercept.

Note this  $R^2$  definition is also the Kullback-Leibler divergence  $R^2$  definition for the normal distribution.

This  $R^2$  can be negative and the target is again not fixed.

Some R code giving an example of each case

```
library(INLA); library(brinla); library(xtable); library(mvtnorm)

## Loading required package: sp
## Loading required package: Matrix
## This is INLA_17.06.20 built 2017-06-20 03:44:36 UTC.
## See www.r-inla.org/contact-us for how to get help.
```

```

#example data from https://github.com/julianfaraway/brinla

data(usair, package = "brinla")

center_scale <- function(x) {
  scale(x, scale = FALSE)
}
# demean predictors first, need for our  $R^2_{KL}$ 

usair2<-center_scale(usair[2:7])
S02<-usair[,1]
usair<-data.frame(cbind(S02,usair2))

usair.formula1 <- S02 ~ negtemp + manuf + wind + precip + days
usair.lm1 <- lm(usair.formula1, data = usair)
round(summary(usair.lm1)$r.squared,4)

```

```
[1] 0.604
```

```
round(summary(usair.lm1)$adj.r.squared,4)
```

```
[1] 0.5475
```

```

#R2 from Gelman implemented in rstanarm package using MCMC

library(rstanarm)

## Loading required package: Rcpp
## rstanarm (Version 2.17.2, packaged: )
## - Do not expect the default priors to remain the same in
  future rstanarm versions.
## Thus, R scripts should specify priors explicitly, even if
  they are just the defaults.
## - For execution on a local, multicore CPU with excess RAM
  we recommend calling
## options(mc.cores = parallel::detectCores())
## - Plotting theme set to bayesplot::theme_default().

M1 <- stan_glm(S02 ~ negtemp + manuf + wind + precip, data=usair )

```

SAMPLING FOR MODEL 'continuous' NOW (CHAIN 1).

Gradient evaluation took 0.000116 seconds 1000 transitions using 10 leapfrog steps per transition would take 1.16 seconds. Adjust your expectations accordingly!

Iteration: 1 / 2000 [ 0Iteration: 200 / 2000 [ 10Iteration: 400 / 2000 [ 20Iteration: 600 / 2000 [ 30Iteration: 800 / 2000 [ 40Iteration: 1000 / 2000 [ 50Iteration: 1001 / 2000 [ 50Iteration: 1200 / 2000 [ 60Iteration: 1400 / 2000 [ 70Iteration: 1600 / 2000 [ 80Iteration: 1800 / 2000 [ 90Iteration: 2000 / 2000 [100

Elapsed Time: 0.097546 seconds (Warm-up) 0.07666 seconds (Sampling) 0.174206 seconds (Total)

SAMPLING FOR MODEL 'continuous' NOW (CHAIN 2).

Gradient evaluation took 1.3e-05 seconds 1000 transitions using 10 leapfrog steps per transition would take 0.13 seconds. Adjust your expectations accordingly!

Iteration: 1 / 2000 [ 0Iteration: 200 / 2000 [ 10Iteration: 400 / 2000 [ 20Iteration: 600 / 2000 [ 30Iteration: 800 / 2000 [ 40Iteration: 1000 / 2000 [ 50Iteration: 1001 / 2000 [ 50Iteration: 1200 / 2000 [ 60Iteration: 1400 / 2000 [ 70Iteration: 1600 / 2000 [ 80Iteration: 1800 / 2000 [ 90Iteration: 2000 / 2000 [100

Elapsed Time: 0.093212 seconds (Warm-up) 0.087976 seconds (Sampling) 0.181188 seconds (Total)

SAMPLING FOR MODEL 'continuous' NOW (CHAIN 3).

Gradient evaluation took 1.5e-05 seconds 1000 transitions using 10 leapfrog steps per transition would take 0.15 seconds. Adjust your expectations accordingly!

Iteration: 1 / 2000 [ 0Iteration: 200 / 2000 [ 10Iteration: 400 / 2000 [ 20Iteration: 600 / 2000 [ 30Iteration: 800 / 2000 [ 40Iteration: 1000 / 2000 [ 50Iteration: 1001 / 2000 [ 50Iteration: 1200 / 2000 [ 60Iteration: 1400 / 2000 [ 70Iteration: 1600 / 2000 [ 80Iteration: 1800 / 2000 [ 90Iteration: 2000 / 2000 [100

Elapsed Time: 0.124865 seconds (Warm-up) 0.061417 seconds (Sampling) 0.186282 seconds (Total)

SAMPLING FOR MODEL 'continuous' NOW (CHAIN 4).

Gradient evaluation took 1.2e-05 seconds 1000 transitions using 10 leapfrog steps per transition would take 0.12 seconds. Adjust your expectations accordingly!

Iteration: 1 / 2000 [ 0Iteration: 200 / 2000 [ 10Iteration: 400 / 2000 [ 20Iteration: 600 / 2000 [ 30Iteration: 800 / 2000 [ 40Iteration: 1000 / 2000 [ 50Iteration: 1001 / 2000 [ 50Iteration: 1200 / 2000 [ 60Iteration:

1400 / 2000 [ 70Iteration: 1600 / 2000 [ 80Iteration: 1800 / 2000 [ 90Iteration: 2000 / 2000 [100

Elapsed Time: 0.089422 seconds (Warm-up) 0.069009 seconds (Sampling) 0.158431 seconds (Total)

```
print(median(bayes_R2(M1)))
```

[1] 0.5947026

*#For the other two R<sup>2</sup> INLA is used.*

```
usair.inla1 <- inla(usair.formula1, data = usair, control.compute = list(dic =
inla.obj<-usair.inla1
samples<-inla.posterior.sample(n=1000, inla.obj)
beta <- matrix(0, nrow = 1000, ncol = 6)
for(i in 1:1000){
  beta[i,] <- samples[[i]]$latent[42:47]
}
```

```
yhat <- inla.obj$model.matrix %*% t(beta)
```

*#R2\_LRT*

```
M<-matrix(0, 1000,2)
```

```
for ( i in 1:1000){
```

```
  v<-1/samples[[i]]$hyperpar
```

```
  M[i,2]=v
```

```
  logLik1 <- dmvnorm(usair$S02, mean=yhat[,i], sigma=diag(41)*v, log=T)
```

```
  #logLik0 <- dmvnorm(usair$S02, mean=rep(beta[i,1],41), sigma=diag(41)*v, log=
```

```
  fit0<-lm(S02~1, data=usair)
```

```
  logLik0 <- logLik(fit0)
```

```
  LRT=-2*(logLik0-logLik1)
```

```
  M[i,1]=1-exp(-LRT/41)
```

```
}
```

```
median(M[,1]) # R2_LRT substantially lower
```

[1] 0.5328855

```
# third option classical definition R_KL

M=matrix(0, 1000, 1)
for (i in 1:1000){
M[i,1]=1-var(usair$S02-yhat[,i])/var((usair$S02-yhat[1,i]))
}

median(M)
```

```
[1] 0.5555212
```

```
#4th option like Gelman but error also sampled from posterior

M=matrix(0, 1000, 1)
for (i in 1:1000){
  M[i,1]=var(yhat[,i])/(var(yhat[,i])+(1/samples[[i]]$hyperpar))
}

median(M) # does not make a big difference
```

```
[1] 0.5974581
```

```
# when prior is uninformativ compared to R^2_Gelman
```

Some Problems with predictors that have no explanatory power

```
set.seed(3)

usair$x1<-scale(rnorm(41,0,1))
usair$x2<-scale(rnorm(41,0,1))
usair$x3<-scale(rnorm(41,0,1))

usair.formula1 <- S02 ~ x1 + x2 + x3
usair.lm1 <- lm(usair.formula1, data = usair)
round(summary(usair.lm1)$r.squared,4)
```

```
[1] 0.0493
```

```
round(summary(usair.lm1)$adj.r.squared,4)
```

```
[1] -0.0278
```

```
M1 <- stan_glm(S02 ~ x1 + x2 + x3, data=usair )
```

SAMPLING FOR MODEL 'continuous' NOW (CHAIN 1).

Gradient evaluation took 2.4e-05 seconds 1000 transitions using 10 leapfrog steps per transition would take 0.24 seconds. Adjust your expectations accordingly!

Iteration: 1 / 2000 [ 0Iteration: 200 / 2000 [ 10Iteration: 400 / 2000 [ 20Iteration: 600 / 2000 [ 30Iteration: 800 / 2000 [ 40Iteration: 1000 / 2000 [ 50Iteration: 1001 / 2000 [ 50Iteration: 1200 / 2000 [ 60Iteration: 1400 / 2000 [ 70Iteration: 1600 / 2000 [ 80Iteration: 1800 / 2000 [ 90Iteration: 2000 / 2000 [100

Elapsed Time: 0.088092 seconds (Warm-up) 0.061695 seconds (Sampling) 0.149787 seconds (Total)

SAMPLING FOR MODEL 'continuous' NOW (CHAIN 2).

Gradient evaluation took 1.3e-05 seconds 1000 transitions using 10 leapfrog steps per transition would take 0.13 seconds. Adjust your expectations accordingly!

Iteration: 1 / 2000 [ 0Iteration: 200 / 2000 [ 10Iteration: 400 / 2000 [ 20Iteration: 600 / 2000 [ 30Iteration: 800 / 2000 [ 40Iteration: 1000 / 2000 [ 50Iteration: 1001 / 2000 [ 50Iteration: 1200 / 2000 [ 60Iteration: 1400 / 2000 [ 70Iteration: 1600 / 2000 [ 80Iteration: 1800 / 2000 [ 90Iteration: 2000 / 2000 [100

Elapsed Time: 0.098843 seconds (Warm-up) 0.088581 seconds (Sampling) 0.187424 seconds (Total)

SAMPLING FOR MODEL 'continuous' NOW (CHAIN 3).

Gradient evaluation took 3.3e-05 seconds 1000 transitions using 10 leapfrog steps per transition would take 0.33 seconds. Adjust your expectations accordingly!

Iteration: 1 / 2000 [ 0Iteration: 200 / 2000 [ 10Iteration: 400 / 2000 [ 20Iteration: 600 / 2000 [ 30Iteration: 800 / 2000 [ 40Iteration: 1000 / 2000 [ 50Iteration: 1001 / 2000 [ 50Iteration: 1200 / 2000 [ 60Iteration: 1400 / 2000 [ 70Iteration: 1600 / 2000 [ 80Iteration: 1800 / 2000 [ 90Iteration: 2000 / 2000 [100

Elapsed Time: 0.084399 seconds (Warm-up) 0.064255 seconds (Sampling) 0.148654 seconds (Total)

SAMPLING FOR MODEL 'continuous' NOW (CHAIN 4).

Gradient evaluation took 1.3e-05 seconds 1000 transitions using 10 leapfrog steps per transition would take 0.13 seconds. Adjust your expectations accordingly!

```
Iteration: 1 / 2000 [ 0Iteration: 200 / 2000 [ 10Iteration: 400 / 2000 [
20Iteration: 600 / 2000 [ 30Iteration: 800 / 2000 [ 40Iteration: 1000 /
2000 [ 50Iteration: 1001 / 2000 [ 50Iteration: 1200 / 2000 [ 60Iteration:
1400 / 2000 [ 70Iteration: 1600 / 2000 [ 80Iteration: 1800 / 2000 [
90Iteration: 2000 / 2000 [100
```

Elapsed Time: 0.084435 seconds (Warm-up) 0.059713 seconds (Sampling) 0.144148 seconds (Total)

```
print(median(bayes_R2(M1))) # bigger than maximum likelihood R^2
```

```
[1] 0.09407575
```

```
usair.inla1 <- inla(usair.formula1, data = usair, control.compute = list(dic =
inla.obj<-usair.inla1
samples<-inla.posterior.sample(n=1000, inla.obj)
beta <- matrix(0, nrow = 1000, ncol = 4)
for(i in 1:1000){
  beta[i,] <- samples[[i]]$latent[42:45]
}
```

```
yhat <- inla.obj$model.matrix %*% t(beta)
```

```
#R2_LRT
```

```
M<-matrix(0, 1000,2)
for ( i in 1:1000){

  v<-1/samples[[i]]$hyperpar
  M[i,2]=v
  logLik1 <- dmvnorm(usair$S02, mean=yhat[,i], sigma=diag(41)*v, log=T)

  fit0<-lm(S02~1, data=usair)
  logLik0 <- logLik(fit0)
  LRT=-2*(logLik0-logLik1)
  M[i,1]=1-exp(-LRT/41)
}
median(M[,1]) # R2_LRT substantially lower
```

```
[1] -0.05644335
```

```
# third option classical definition R_KL
```

```
M=matrix(0, 1000, 1)
```



```
for (i in 1:1000){
M[i,1]=1-var(usair$S02-yhat[,i])/var((usair$S02-yhat[1,i]))
}

median(M)
```

[1] -0.009601396

```
#4th option like Gelman but error also sampled from posterior

M=matrix(0, 1000, 1)
for (i in 1:1000){
  M[i,1]=var(yhat[,i])/(var(yhat[,i])+(1/samples[[i]]$hyperpar))
}

median(M)
```

[1] 0.09215904

The  $R^2_{Gelman}$  is even bigger than the  $R^2$  from the lm object using maximum likelihood. Comparing Bayesian  $R^2$  :  $R^2_{Gelman} > R^2_{KL} > R^2_{LRT}$ , which was expected.

The in Gelman (2017) proposed  $R^2$  using crossvalidation can also easily be calculated in INLA and rstanarm. We have then just one value per model and may therefore be faster to calculate for all models.

$LPML = \sum_{n=1}^n \log(p(y_i|y_{-i}))$  is used to calculate the loglikelihood of the model. For the intercept only model i think the maximum likelihood should be used.

```
#R2 LOO crossvalidation with likelihood

library(rstanarm)

usair.formula1 <- S02 ~ negtemp + manuf + wind + precip + days
usair.inla1 <- inla(usair.formula1, data = usair, control.compute = list(dic =
LPMLfit<-sum(log(usair.inla1$cpo$cpo))
fit0 <- lm(S02 ~ 1, data = usair)
LRT<- -2*(logLik(fit0)-LPMLfit)
1-exp(-LRT/41) # much lower
```

'log Lik.' 0.3798198 (df=2)

The LOO crossvalidation could of course also be used for the other  $R^2$  definitions, but will then take longer to calculate.