Lecture 2

Diagnostics and Model Evaluation

Colin Rundel

1/23/2017

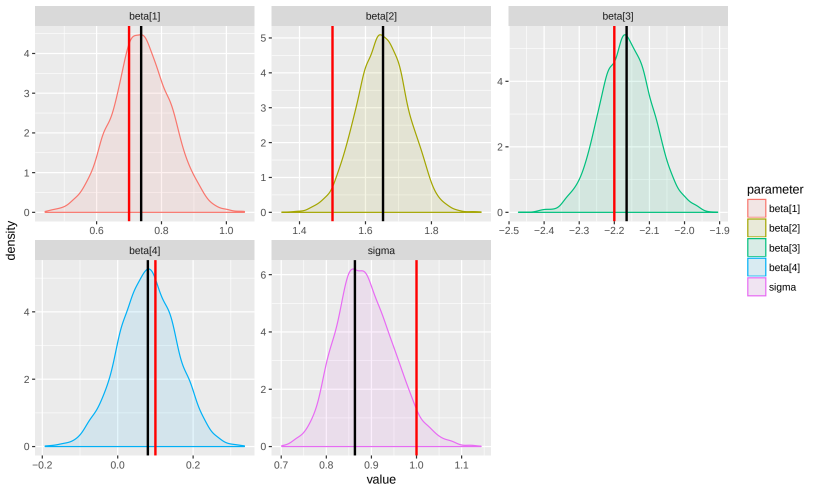
From last time

Linear model and data

```r
library(rjags)
library(dplyr)

set.seed(01172017)
n = 100
beta = c(0.7,1.5,-2.2,0.1)
eps = rnorm(n, mean=0, sd=1)

X0 = rep(1, n)
X1 = rt(n,df=5)
X2 = rt(n,df=5)
X3 = rt(n,df=5)

X = cbind(X0, X1, X2, X3)
Y = X %*% beta + eps
d = data.frame(Y,X[,-1])
```

Bayesian model

```
## model{
##   # Likelihood
##   for(i in 1:length(Y)){
##     Y[i]   ~ dnorm(mu[i],tau2)
##     mu[i] <- beta[1] + beta[2]*X1[i] + beta[3]*X2[i] + beta[4]*X3[i]
##   }
##
##   # Prior for beta
##   for(j in 1:4){
##     beta[j] ~ dnorm(0,1/100)
##   }
##
##   # Prior for the inverse variance
##   tau2   ~ dgamma(1, 1)
##   sigma <- 1/sqrt(tau2)
## }
```


beta[1]
beta[2]
beta[3]
beta[4]
sigma
density
value
0
1
2
3
4
5
6
0.6
0.8
1.0
1.4
1.6
1.8
−2.5
−2.4
−2.3
−2.2
−2.1
−2.0
−1.9
−0.2
0.0
0.2
0.7
0.8
0.9
1.0
1.1
parameter
beta[1]
beta[2]
beta[3]
beta[4]
sigma

Model Evaluation

Model assessment?

If we think back to our first regression class, one common option is R^2 which gives us the variability in Y explained by our model.

Quick review:

Model assessment?

If we think back to our first regression class, one common option is R^2 which gives us the variability in Y explained by our model.

Quick review:

$$\underbrace{\sum_{i}^{n} \left(Y_i - \overline{Y}\right)^2}_{\text{Total}} = \underbrace{\sum_{i=1}^{n} \left(\hat{Y}_i - \overline{Y}\right)^2}_{\text{Model}} + \underbrace{\sum_{i=1}^{n} \left(Y_i - \hat{Y}_i\right)^2}_{\text{Error}}$$

Model assessment?

If we think back to our first regression class, one common option is R^2 which gives us the variability in Y explained by our model.

Quick review:

$$\underset{\text{Total}}{\sum_{i}^{n} \left(Y_i - \overline{Y}\right)^2} = \underset{\text{Model}}{\sum_{i=1}^{n} \left(\hat{Y}_i - \overline{Y}\right)^2} + \underset{\text{Error}}{\sum_{i=1}^{n} \left(Y_i - \hat{Y}_i\right)^2}$$

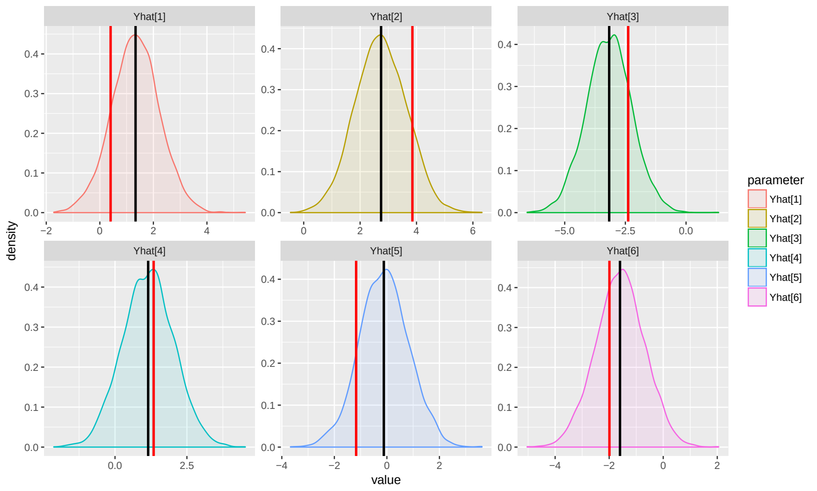
$$R^2 = \text{Corr}\left(Y, \hat{Y}\right)^2 = \frac{\sum_{i=1}^{n} \left(\hat{Y}_i - \overline{Y}\right)^2}{\sum_{i=1}^{n} \left(Y_i - \overline{Y}\right)^2} = 1 - \frac{\sum_{i=1}^{n} \left(\hat{Y}_i - \hat{Y}_i\right)^2}{\sum_{i=1}^{n} \left(Y_i - \overline{Y}\right)^2}$$

Bayesian R^2

While we can collapse our posterior parameter distributions to a single value (using mean, median, etc.) before calculating $\hat{Y}$ and then R^2, we don't need to.

```
n_sim = 5000; n = 100
Y_hat = matrix(NA, nrow=n_sim, ncol=n, dimnames=list(NULL, paste0(”Yhat[”,1:

for(i in 1:n_sim)
{
  beta_post = samp[[1]][i,1:4]
  sigma_post = samp[[1]][i,5]

  Y_hat[i,] = beta_post %*% t(X) + rnorm(n, sd = sigma_post)
}

Y_hat[1:5, 1:5]
##         Yhat[1]  Yhat[2]   Yhat[3]    Yhat[4]    Yhat[5]
## [1,] -1.7193735 2.712832 -2.935192 -0.3472465 -0.5000324
## [2,]  0.4478495 3.035438 -4.419457  2.2174576  0.3661503
## [3,]  3.2258841 2.608588 -2.472159  1.1553329  1.7027229
## [4,]  1.8120911 2.612417 -3.349495  2.3028439  1.0398770
## [5,]  1.2504531 1.996477 -2.424596  0.5437237  0.7191012
```

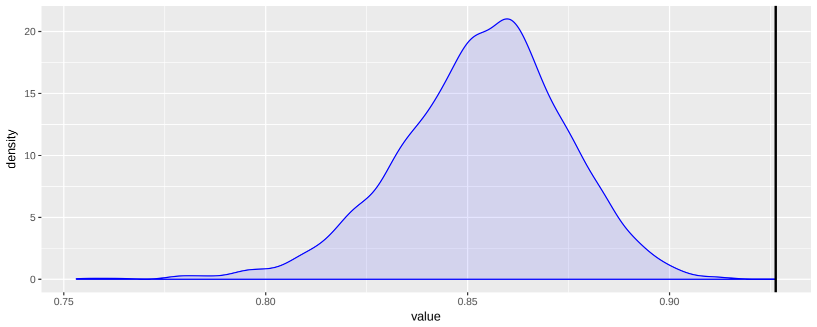
$\hat{Y}$ - lm vs Bayesian lm



Posterior R^2

For each posterior sample s we can calculate $R^2_s = \text{Corr}(Y, \hat{Y}_s)^2$,

```
R2_post = apply(Y_hat, 1, function(Y_hat_s) cor(Y, Y_hat_s)^2)

summary(c(R2_post)) %>% t()
##         Min. 1st Qu. Median   Mean 3rd Qu.   Max.
## [1,] 0.7531  0.8410 0.8549 0.8536  0.8672 0.9168

summary(lm(Y~., data=d))$r.squared
## [1] 0.9262839
```



What if we collapsed first?

```
Y_hat_post_mean = apply(Y_hat, 2, mean)
head(Y_hat_post_mean)
##    Yhat[1]    Yhat[2]    Yhat[3]    Yhat[4]    Yhat[5]    Yhat[6]
##  1.3324163  2.7363221 -3.1772690  1.1531411 -0.1029881 -1.5918980

Y_hat_post_med  = apply(Y_hat, 2, median)
head(Y_hat_post_med)
##    Yhat[1]    Yhat[2]    Yhat[3]    Yhat[4]    Yhat[5]    Yhat[6]
##  1.3336394  2.7270929 -3.1659989  1.1631532 -0.1003662 -1.5807078

cor(Y_hat_post_mean, Y)^2
##           [,1]
## [1,] 0.9264776

cor(Y_hat_post_med,  Y)^2
##           [,1]
## [1,] 0.9264527

summary(lm(Y~., data=d))$r.squared
## [1] 0.9262839
```

What went wrong?

If our criteria is to maximize R^2, then nothing. Why this result?

What went wrong?

If our criteria is to maximize R^2, then nothing. Why this result?

Remember that $\hat{\beta}_{MLE} = \hat{\beta}_{LS}$, the latter of which is achieved by

$$\arg\min_{\boldsymbol{\beta}} \sum_{i=1}^{n} \left(Y_i - \mathbf{X}_{i\cdot}\boldsymbol{\beta}\right)^2 = \arg\min_{\boldsymbol{\beta}} \sum_{i=1}^{n} \left(Y_i - \hat{Y}_i\right)^2$$

What went wrong?

If our criteria is to maximize R^2, then nothing. Why this result?

Remember that $\hat{\beta}_{MLE} = \hat{\beta}_{LS}$, the latter of which is achieved by

$$\arg\min_{\boldsymbol{\beta}} \sum_{i=1}^{n} (Y_i - \mathbf{X}_{i\cdot}\boldsymbol{\beta})^2 = \arg\min_{\boldsymbol{\beta}} \sum_{i=1}^{n} \left(Y_i - \hat{Y}_i\right)^2$$

So if we have $\boldsymbol{\beta}$ such that it minimizes the least squares criterion what does that tell us about

$$R^2 = \text{Corr}(\mathbf{Y}, \hat{\mathbf{Y}})^2 = \frac{\sum_{i=1}^{n} \left(\hat{Y}_i - \overline{Y}\right)^2}{\sum_{i=1}^{n} \left(Y_i - \overline{Y}\right)^2} = 1 - \frac{\sum_{i=1}^{n} \left(Y_i - \hat{Y}_i\right)^2}{\sum_{i=1}^{n} \left(Y_i - \overline{Y}\right)^2}$$

Some problems with R^2

- R^2 always increases (or stays the same) when a predictor is added
- R^2 is highly susecptible to over fitting
- R^2 is sensitive to outliers
- R^2 depends heavily on current values of Y
- R^2 can differ drastically for two equivalent models (i.e. nearly identical inferences about key parameters)

Other Metrics

Root Mean Square Error

The traditional definition of rmse is as follows

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} \left(Y_i - \hat{Y}_i\right)^2}$$

In the bayesian context where we have posterior samples from each parameter / prediction of interest we can express this equation as

$$\text{RMSE} = \sqrt{\frac{1}{n\, n_s} \sum_{s=1}^{n_s} \sum_{i=1}^{n} \left(Y_i - \hat{Y}_{i,s}\right)^2}$$

Note that as we just saw with R^2 using the first definition with $\hat{Y}_i = \sum_{s=1}^{n_s} \hat{Y}_{i,s}/n$ does not necessarily give the same result as the 2nd equation.

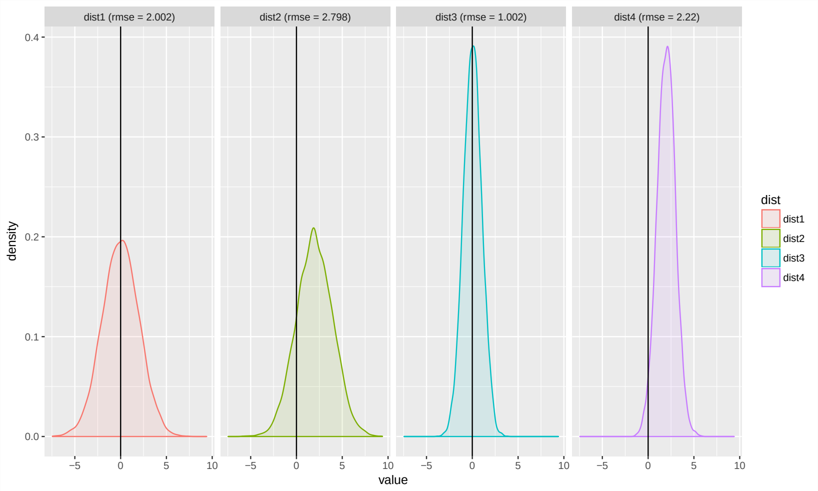
Continuous Rank Probability Score

RMSE (and related metrics like MAE) are not directly applicable to probabilistic predictions since they require fixed values of $\hat{Y}_i$. We can generalize to a fully continuous case where $\hat{Y}$ is given by a predictive distribution using a Continuous Rank Probability Score

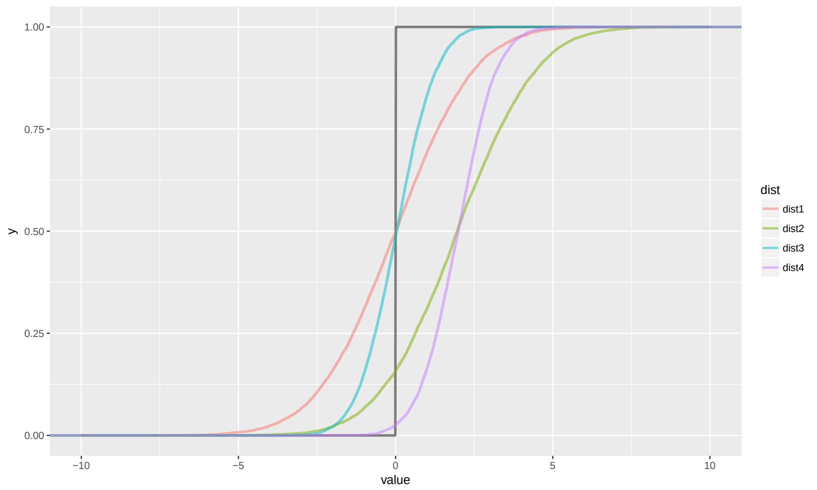
$$\text{CRPS} = \int_{-\infty}^{\infty} \left(F_{\hat{Y}}(z) - \mathbf{1}_{\{z \geq Y\}}\right)^2 dz$$

where $F_{\hat{Y}}$ is the empirical CDF of $\hat{Y}$ (the posterior predictive distribution for Y) and $\mathbf{1}_{z \geq Y}$ is the indicator function which equals 1 when $z \geq Y$, the true/observed value of Y.

Accuracy vs. Precision



CDF vs Indicator



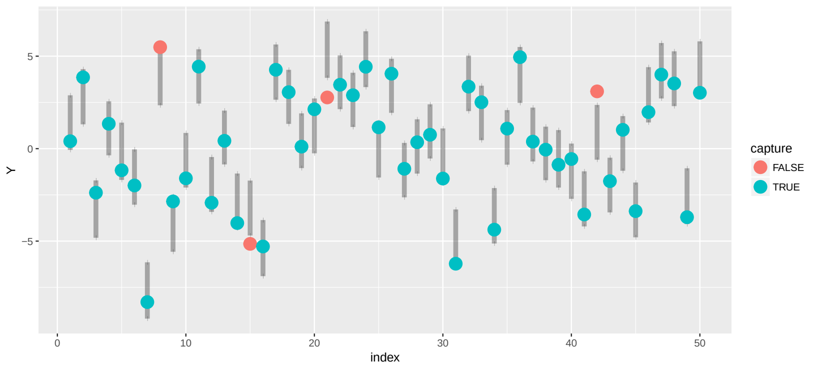
```
## dist1 dist2 dist3 dist4
## 0.468 1.188 0.235 1.432
```

Empirical Coverage

One final method of assessing model calibration is assessing how well credible intervals, derived from the posterior predictive distributions of the Ys, capture the true/observed values.

Empirical Coverage

One final method of assessing model calibration is assessing how well credible intervals, derived from the posterior predictive distributions of the *Y*s, capture the true/observed values.

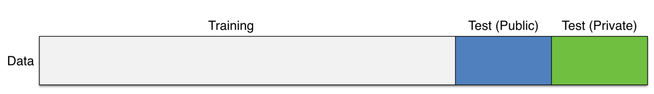


```
## 90% CI empirical coverage
##                         0.92
```

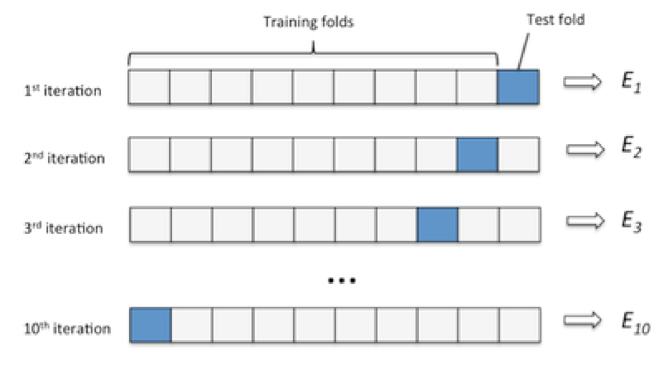
Cross-validation

Cross-validation styles

Kaggle style:



k-fold:



Cross-validation in R with modelr

```
library(modelr)

d_kaggle = resample_partition(d, c(train=0.70, test1=0.15, test2=0.15))
d_kaggle
## $train
## <resample [69 x 4]> 1, 2, 3, 4, 6, 7, 8, 9, 11, 13, ...
##
## $test1
## <resample [15 x 4]> 12, 19, 23, 30, 34, 35, 36, 48, 54, 58, ...
##
## $test2
## <resample [16 x 4]> 5, 10, 18, 21, 31, 41, 45, 46, 57, 68, ...

d_kfold = crossv_kfold(d, k=5)
d_kfold
## # A tibble: 5 × 3
##             train            test   .id
##            <list>          <list> <chr>
## 1 <S3: resample> <S3: resample>     1
## 2 <S3: resample> <S3: resample>     2
## 3 <S3: resample> <S3: resample>     3
## 4 <S3: resample> <S3: resample>     4
## 5 <S3: resample> <S3: resample>     5
```

resample objects

The simple idea behind **resample** objects is that there is no need to create and hold on to these subsets / partitions of the original data frame - only need to track which rows belong to what subset and then handle the creation of the new data frame when necessary.

```
d_kaggle$test1
## <resample [15 x 4]> 12, 19, 23, 30, 34, 35, 36, 48, 54, 58, ...

str(d_kaggle$test1)
## List of 2
##  $ data:’data.frame’:    100 obs. of  4 variables:
##   ..$ Y : num [1:100] 0.402 3.855 -2.384 1.344 -1.171 ...
##   ..$ X1: num [1:100] 0.0513 0.0259 -1.2622 -0.443 -3.1655 ...
##   ..$ X2: num [1:100] -0.284 -0.928 0.948 -0.535 -1.961 ...
##   ..$ X3: num [1:100] -1.279 -0.571 2.938 -0.15 1.664 ...
##  $ idx : int [1:15] 12 19 23 30 34 35 36 48 54 58 ...
##  - attr(*, ”class”)= chr ”resample”

as.data.frame(d_kaggle$test1)
##             Y          X1         X2          X3
## 12 -2.9263300 -0.82679975  0.5995159 -1.64750639
## 19  0.1090239  1.34327879  1.1326438 -0.16019583
## 23  2.8883752  0.40828680 -0.5064750  1.58942370
```

Simple usage

```r
lm_train = lm(Y~., data=d_kaggle$train)

lm_train %>% summary() %$% r.squared
## [1] 0.9410859

rsquare(lm_train, d_kaggle$train)
## [1] 0.9410859


Y_hat_test1 = predict(lm_train, d_kaggle$test1)

(Y_hat_test1 - as.data.frame(d_kaggle$test1)$Y)^2 %>% mean() %>% sqrt()
## [1] 1.071201

rmse(lm_train, d_kaggle$test1)
## [1] 1.071201

rmse(lm_train, d_kaggle$test2)
## [1] 1.031323
```

Aside: `purrr`

`purrr` is a package by Hadley which improves functional programming in R by focusing on pure and type stable functions. It provides basic functions for looping over objects and returning a value (of a specific type) - think of it as a better version of `lapply`/`sapply`/`vapply`.

- `map()` - returns a list.
- `map_lgl()` - returns a logical vector.
- `map_int()` - returns a integer vector.
- `map_dbl()` - returns a double vector.
- `map_chr()` - returns a character vector.
- `map_df()` - returns a data frame.
- `map2_*` - variants for iterating over two vectors simultaneously.

Aside: Type Consistency

R is a weakly / dynamically typed language which means there is no way to define a function which enforces the argument or return types.

This flexibility can be useful at times, but often it makes it hard to reason about your code and requires more verbose code to handle edge cases.

```
library(purrr)
##
## Attaching package: ’purrr’
## The following objects are masked from ’package:dplyr’:
##
##     contains, order_by
## The following object is masked from ’package:magrittr’:
##
##     set_names

map_dbl(list(rnorm(1e3),rnorm(1e3),rnorm(1e3)), mean)
## [1] -0.02679155  0.02086224 -0.01229868
map_chr(list(rnorm(1e3),rnorm(1e3),rnorm(1e3)), mean)
## [1] ”0.021504”  ”0.025358”  ”-0.035972”
map_int(list(rnorm(1e3),rnorm(1e3),rnorm(1e3)), mean)
## Error: Can’t coerce element 1 from a double to a integer
```

Aside: Anonymous Functions shortcut

An anonymous function is one that is never given a name (i.e. assigned to a variable), using base R we would write something like the following,

```
sapply(1:10, function(x) x^(x+1))
##  [1]            1            8           81         1024        15625
##  [6]       279936      5764801    134217728   3486784401 100000000000
```

purrr lets us write anonymous functions using the traditional style, but also lets us use one sided formulas where the value being mapped is referenced by `.`

```
map_dbl(1:10, function(x) x^(x+1))
##  [1]            1            8           81         1024        15625
##  [6]       279936      5764801    134217728   3486784401 100000000000

map_dbl(1:10, ~ .^(.+1))
##  [1]            1            8           81         1024        15625
##  [6]       279936      5764801    134217728   3486784401 100000000000
```

Cross-validation in R with `modelr` + `purrr`

```
lm_models = map(d_kfold$train, ~ lm(Y~., data=.))
str(lm_models, max.level = 1)
## List of 5
##  $ 1:List of 12
##   ..- attr(*, ”class”)= chr ”lm”
##  $ 2:List of 12
##   ..- attr(*, ”class”)= chr ”lm”
##  $ 3:List of 12
##   ..- attr(*, ”class”)= chr ”lm”
##  $ 4:List of 12
##   ..- attr(*, ”class”)= chr ”lm”
##  $ 5:List of 12
##   ..- attr(*, ”class”)= chr ”lm”

map2_dbl(lm_models, d_kfold$train, rsquare)
##         1         2         3         4         5
## 0.9201087 0.9137336 0.9285830 0.9379184 0.9301658

map2_dbl(lm_models, d_kfold$test, rmse)
##         1         2         3         4         5
## 0.8957795 0.6809255 0.8808314 1.0825899 0.8538277
```

Getting `modelr` to play nice with `rjags`

We used the following code to fit out model previously, lets generalize / functionalize it so we can use it with modelr

```
m = jags.model(
        textConnection(model),
        data = list(Y=c(Y), X1=X1, X2=X2, X3=X3)
       )
update(m, n.iter=1000, progress.bar=”none”)
samp = coda.samples(
        m, variable.names=c(”beta”,”sigma”),
        n.iter=5000, progress.bar=”none”
       )
```

Fitting the model

```
fit_jags_lm = function(data, n_burnin=1000, n_samps=5000)
{
  data = as.data.frame(data, optional=TRUE)

  m = jags.model(textConnection(model), data = data, quiet=TRUE)
  update(m, n.iter=n_burnin, progress.bar=”none”)
  coda.samples(
    m, variable.names=c(”beta”,”sigma”),
    n.iter=n_samps, progress.bar=”none”
  )[[1]]
}
```

Predicting the model

```
predict_jags_lm = function(samp, newdata)
{
  data = as.data.frame(newdata, optional=TRUE) %>% tbl_df()

  n = nrow(newdata)
  beta0_post = samp[,1]; beta1_post = samp[,2]
  beta2_post = samp[,3]; beta3_post = samp[,4]
  sigma_post = samp[,5]

  data$post_pred = list(NA)
  for(i in 1:n)
  {
    mu = beta0_post * 1          + beta1_post * data$X1[i] +
         beta2_post * data$X2[i] + beta3_post * data$X3[i]
    error = rnorm(n_sim, sd = sigma_post)

    data$post_pred[[i]] = mu + error
  }

  data
}
```

Empirical Coverage

```r
empcov = function(pred, obs_col, width=0.9)
{
  cred_int = map(pred$post_pred, ~ HPDinterval(., width)) %>%
    do.call(rbind, .)

  observed = pred[[obs_col]]

  data = cbind(pred, cred_int) %>%
    tbl_df() %>%
    mutate(capture = lower <= observed & upper >= observed)

  cat(width*100,"% CI empirical coverage = ",
      round(sum(data$capture)/nrow(data),3), "\n", sep="")

  invisible(data)
}
```

Putting it together

```
model_fit = fit_jags_lm(d_kaggle$train)
train_pred = predict_jags_lm(model_fit, newdata = d_kaggle$train)
test1_pred = predict_jags_lm(model_fit, newdata = d_kaggle$test1)
test2_pred = predict_jags_lm(model_fit, newdata = d_kaggle$test2)

empcov(train_pred, obs_col=”Y”, width=0.9)
## 90% CI empirical coverage = 0.913

empcov(test1_pred, obs_col=”Y”, width=0.9)
## 90% CI empirical coverage = 0.733

empcov(test2_pred, obs_col=”Y”, width=0.9)
## 90% CI empirical coverage = 0.875
```