Freie Universität Berlin
Chair of Statistics
Location: Berlin
Summer Term 2017
Lecture: Einführung in die Bayes-Statistik
Examiner: Dr. Florian Meinfelder

Program leave-one-out posterior predictive checking in R

Johannes Brinkmann (4659632), jojo-brinkmann@gmx.de
Carlo Michaelis (5043128), carlo.michaelis@gmail.com
Max Reinhardt (579174), max_reinhardt@me.com
Adrian Rolf (5071860), adrian.rolf@gmx.de

Master Statistics
August 29, 2017

Contents

List of Figures

List of Tables

1 Introduction

Leave-on-out cross validation is one of many methods which is used after fitting a bayesian model to measure its predictive accuracy and also to choose the best model (Vehtari, Gelman, and Gabry 2017). In practice it was not often used in the past due to the fact that it involves additional computation power and time as different models must be fitted in order to find the best model. But there was a change in the last 10 to 15 years as computational power got cheaper and was therefore more available for the public (Hoeting and Ibrahim 1998).
In the following paper model evaluation for several Bayesian linear models is done by calculating the mean squared error of the model as well as the log pointwise predictive density. The best model is then selected by looking for the smallest values of those calculated measures. Further, the selected model will be calculated using the frequentist approach. Finally the Bayesian model and the frequentist model will be compared using Cook's Distance. In both steps Leave-on-out cross validation is required to derive these measures.

2 Theory

2.1 Bayesian Linear Model

With linear models we want to express the conditional value of a variable y given X as a linear function of X. The classical linear regression model is defined as:

$$y = X\beta + \epsilon, \tag{2.1}$$

where $\epsilon_i \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma^2)$, i.e. the errors are independently distributed with homogeneous variance. To obtain an estimate for β the frequentist approach is to use the *ordinary least squares* (OLS) method which yields $\hat{\beta} = (X^\top X)^{-1} X^\top y$. σ^2 is estimated by the sample variance $s^2 = \frac{1}{n-p}(y - X\hat{\beta})^\top (y - X\hat{\beta})$. However, in a Bayesian approach we are not so much concerned with just finding specific values for β and σ^2, but rather being able to comment on the probability distributions of β and σ^2. Then, point estimates like the expected value can easily be derived.

In a Bayesian context, to cover all information in the model, we should also consider the distribution of X, i.e. $p(X|\psi)$ and the prior of ψ. But following (Gelman et al. 2014), focusing only on the parameters (β, σ) while assuming prior independence of (β, σ) and ψ – that is, we ignore the information that may be given by the distribution of X –, and using non-informative priors ends up in (approximately) the same results we will get from the frequentist approach described above. We will make use of these assumptions since we aim for a comparison of both approaches later on. Only dealing with β and σ^2 and further assume prior independence for these parameters we get the full conditionals:

$$\beta | \sigma^2, y, X \sim \mathcal{N}\left(\hat{\beta}, \sigma^2\right), \tag{2.2}$$

$$\frac{1}{\sigma^2} | \beta, y, X \sim \Gamma\left(\frac{n-p}{2}, \frac{(y - X\beta)^\top (y - X\beta)}{2}\right). \tag{2.3}$$

Instead of modeling σ^2 as *inverse-*χ^2 distributed as proposed by (Gelman et al. 2014), we will follow (Hoff 2009) and model $\frac{1}{\sigma^2}$ as Γ distributed. Having the full conditionals we can apply the Gibbs sampling method to draw observations from the conditional posteriors and to derive the expected values for β and σ^2 as Bayesian estimates.

2.2 Gibbs sampling

Gibbs sampling is a convenient method to simulate drawing from a joint posterior distribution when it is given in terms of its full conditionals. The procedure is as follows: Given some

initial values $\theta_1^{(0)}, ..., \theta_p^{(0)}$ for all parameters included in the conditional posteriors, draw $\theta_1^{(1)}$ from $p(\theta_1^{(1)}|\theta_2^{(0)}, \theta_3^{(0)}, ..., \theta_p^{(0)})$, then draw $\theta_2^{(1)}$ from $p(\theta_2^{(1)}|\theta_1^{(1)}, \theta_3^{(0)}, ..., \theta_p^{(0)})$ and so on. After drawing from each of the p conditional posteriors, take the draws as new start values and repeat the previous steps.

Repeating this procedure yields in drawing from the joint posterior distribution, since we are handling a Markov chain here that is converging against its stationary distribution (Wasserman 2013). Convergence may take a while until it is close enough to the stationary distribution, so that one can regard a draw as a draw from the posterior distribution. For this reason the draws from the first b iterations are usually discarded – thus, often called *burn-in*. Convergence behavior can be tracked by trace plots (see figure 2).

2.3 Leave-one-out cross-validation

Leave-one-out cross-validation, or short *loo-cv* is a method for checking the predictive out-of-sample accuracy of a model. The idea is to omit the i-th observation and to fit the model for the remaining observations and then to predict y_i as an out of sample point. This prediction is denoted as $\hat{y}_i^{(i)}$. Thus, we can overcome – at least to some extent – the problem of overfitting. The loo-cv method can be used in several ways for model evaluation and selection. We will use it in two ways. First, we will use it to calculate the *log pointwise predictive density* using the loo-cv, which is defined as:

$$\text{lppd}_{\text{loo-cv}} = \sum_{i=1}^{n} \log p_{\text{dens}(-i)}(y_i). \tag{2.4}$$

That is, we calculate the densities of all y_i using the parameters we obtain from the n bayesian models, applying the loo-cv method. Then taking the logarithm and summing up. In the case at hand we deal with the normal density, since $y_i \sim \mathcal{N}\left(X_{(-i)}\beta_{(-i)}, \sigma^2_{(-i)}\right)$. Furthermore, multiplying with -2, i.e. $-2\ \text{lppd}_{\text{loo-cv}}$, gives us a measure that is comparable with other measures of predictive accuracy (e.g. AIC or WAIC) (Gelman et al. 2014). Within the Bayesian framework, we will use $-2\ \text{lppd}_{\text{loo-cv}}$ besides the MSE for model selection.

Second, we will need it in the context of comparing the predictive out-of-sample accuracy of the Bayesian and the frequentist model using Cook's distance.

2.4 Cook's Distance

Cooks's Distance, or short *Cook's D*, is a measure to detect influential points in a linear regression. It was proposed by (Cook 1977) and is defined as:

$$D_i = \frac{\sum_{j=1}^{n}\left(\hat{y}_j - \hat{y}_j^{(i)}\right)^2}{p\ \text{MSE}} \overset{\text{(OLS)}}{=} \frac{e_i^2}{p\ \text{MSE}}\left(\frac{h_{ii}}{(1-h_{ii})^2}\right), \tag{2.5}$$

where

$$\text{MSE} = \frac{1}{n-p}\sum_{i=1}^{n}(y_i - \hat{y})^2 \tag{2.6}$$

is the *Mean squared error*, $e_i^2 = (y_i - \hat{y})^2$ is the error for each observation i and $h_{ii} = x_i(X^\top X)^{-1}x_i^\top$ is the i-th diagonal element of the projection matrix of an OLS regression model, also called the *leverage* of an observation, which indicates the influence of an single observation on the model fit.

As above, the $\hat{y}^{(i)}$ is the predicted value for y_i if we exclude the i-th observation and fit the model for the remaining observations. Note that the right hand side term of (2.5) is an analytical expression. In this case there is no need to fit additional n models to get the $\hat{y}^{(i)}$ predictions. This yields a great advantage in performance, especially for large numbers of observations. But the analytical

expression is restricted to OLS models – indicated by the *OLS* over the equation sign – since the projection matrix is used. For this reason we will use the left hand expression to calculate Cook's D in the Bayesian case using the loo-cv method described above and the right hand expression for the frequentist OLS regression model.

3 Code

For both applications of the cross validation, the model evaluation and the calculation of Cook's distance, we need the Gibbs sampler and the leave-one-out cross validation (loo-cv) algorithm. These two functions are explained in the first part. The model evaluation is presented in the second part and the third part compares the loo-cv Cook's distance with the analytic solution, which are defined in equation (2.5).

Note that we only displayed the relevant parts of the code in the `pdf` file. Some sections regarding the plotting are hidden and can be found in the `rmd` file.

3.1 Basics

The first function we define is the *Gibbs sampler*. The Gibbs sampler draws the parameters $\beta_0, ..., \beta_p, \sigma^2$ from their conditional posterior distributions. These distributions (see equation (2.2) and (2.3)) need the design matrix `X` and labels `Y`, which are given as arguments. Furthermore the Gibbs sampler does not just draw one time, it draws B times, where $B = R + b$. b is the number of burn-in samples and R the number of used samples. If $b = 0$, we use all generated samples.

In figure 2 the traces of the Gibbs sampler are plotted for all parameters. It can be seen that the values drawn by the sampler differ in some range. Moreover we can see that the initial values of σ^2 and β are corrected very fast to a specific range. Since the initial value of β is chosen as $\hat{\beta}$, the following values are close to the inital value.

Furthermore in figure 3 the histogram and the extimated density (using kernel method) is plotted for every parameter. The red line marks the posterior mean point estimator for any parameters.

Both figures are based on data for the whole dataset of the optimal model (see below).

```
gibbsSampler <- function(X, Y, R, b = 0, initBetas = 1) {
  # Sample parameters from posterior distribution
  # Prepared for baysean linear regression with ordinary least square approach
  #
  # Args:
  #   X: Design matrix
  #   Y: Labels
  #   R: Number of draws (without burn-in), R is a scalar in this funcrion
  #   b: Number of burn-in draws
  #   initBetas: Initial value of betas (default 1)
  #
  # Returns:
  #   List containing sampled betas (number depends on design matrix) and sigmas

  # Get number of overall draws (including burn-in)
  B <- R + b

  # Size of design matrix
  n <- nrow(X) # Number of data points
  p <- ncol(X) # Number of parameters

  # Variables to store the samples in
  betas <- matrix(nrow = B, ncol = p)
  sigma <- rep(NA, B)
```

```
  # OLS of beta
  V <- solve(t(X)%*%X)      # (X^T X)^-1
  beta_hat <- V%*%t(X)%*%Y # (X^T X)^-1 X^T Y

  # Initalize betas and sigma
  betas[1,] <- beta_hat
  sigma[1] <- 1

  # Sampling
  for(i in 1:(B-1)){
    # Sample sigma from the full conditional
    counter <- t(Y-X%*%betas[i,])%*%(Y-X%*%betas[i,])
    sigma[i+1] <- 1/rgamma(1, (n-p)/2, counter/2)

    # Sample beta from the full conditional
    if(i < B) {
      betas[i+1,] <- rmvnorm(1, beta_hat, sigma[i+1]*V)
    }
  }

  # Remove burn-in, if there is some
  if(b != 0) {
    betas <- betas[-(1:b),]
    sigma <- sigma[-(1:b)]
  }

  return(list(betas = betas, sigma = sigma))
}
```

Next, we define a function for the leave-one-out cross validation. It deletes one data point and samples the parameters from the posterior distribution (using the `gibbsSampler` function). This is repeated n times, where n is the size (number of rows) of the data matrix, so there is a sampled posterior distribution for n combinations of the data. Additionally the function estimates the posterior mean (point estimation) of the parameters and predicts the labels, using the data matrix X and the point estimators of the parameters.

Note that the argument R can be a scalar or a vector. If R is a scalar, the crossvalidation only runs once. If R is an ordered vector the Gibbs sampler draws as often as the length of vector R. The length of vector R is saved in the variable steps. Afterwards the calculation of the posterior mean and the prediction is done for every step in the vector R. In the first step, just the first R_1 number of samples are used to estimate the parameters. In the second step, $R_2 > R_1$ number of samples are used, and so on.

```
crossValidation <- function(X, Y, R, b) {
  # Implementation of leave-one-out cross validation (LOO-CV)
  # Cross validate bayesian linear regression model with OLS
  #
  # Args:
  #   X: Design matrix
  #   Y: Labels
  #   R: Number of draws (without burn-in), R can be a
  #      (ordered) vector in this function
  #   b: Number of burn-in draws
  #
  # Returns:
  #   List containing estimated parameters and
  #   label predictions for every LOO-CV step
```

```
  # Size of design matrix
  n <- nrow(X) # Number of data points
  p <- ncol(X) # Number of parameters

  # Get size of R vector (steps = 1, if R is scalar)
  steps <- length(R)

  # Run gibbs sampler to get sampled parameters
  samples <- lapply(1:n, function(i) gibbsSampler(X[-i,], Y[-i], R[steps], b))

  # Initalize lists to store results of estimation and prediction
  Sigma <- list()
  Betas <- list()
  Yhati <- list()

  # Calculate sigma, betas and Yhati for every R step (do it once if R is scalar)
  for(k in 1:steps) {
    Sigma[[k]] <- sapply(samples, function(sample) mean(sample$sigma[1:R[k]]))
    Betas[[k]] <- sapply(samples, function(sample) colMeans(sample$betas[1:R[k],]))
    Yhati[[k]] <- sapply(1:n, function(i) X[i,]%*%Betas[[k]][,i])
  }

  return(list(Sigma = Sigma, Betas = Betas, Yhati = Yhati))
}
```

3.2 Model evaluation

The cross validation algorithm can be used to evaluate the best model. The best mode is found by looking for the smallest values compared to other models using the mean squared error, defined in equation (2.6) or the *lppd* , defined in equation (2.4). Both measures are obtained by loo-cv.

The `bayesModelEvaluation` function calls the cross validation function for every model. Then for every model the *mse* and *lppd* is calculated and returned.

```
bayesModelEvaluation <- function(models, Y, R, b) {
  # Model evaluation based on mean squared error (MSE) and log pointwise
  # predictive density (lppd) using leave-one-out cross validation (LOO-CV)
  #
  # Args:
  #   models: A list of design matrices with different size and types of parameters
  #   Y: Labels
  #   R: Number of draws (without burn-in), R can be a vector in this function
  #   b: Number of burn-in draws
  #
  # Returns:
  #   List containing MSEs and LPPDs for every model

  # Evaluate multiple models and return results from all models
  n <- length(Y)
  k <- length(models)

  # Cross validate every model
  results <- lapply(1:k, function(i) crossValidation(models[[i]], Y, R, b))

  # Calculate Mean Squared Errors
  MSEs <- sapply(results, function(el) {
    return((1/(n-nrow(el$Betas[[1]])))*sum((Y-el$Yhati[[1]])^2))
  })
```

```
  # Calculate log pointwise predictive density (log likelihood)
  # y ~ N(XB, s^2(X^T X)^-1) = N(Yhat, s^2(X^T X)^-1)
  LPPDs <- sapply(results, function(eva) {
      return(sum(log(dnorm(Y, eva$Yhati[[1]], eva$Sigma[[1]]))))
    })

  return(list(MSEs = MSEs, LPPDs = LPPDs))
}
```

To run the model, we first used the `swiss` data set. We defined a list of 11 design matrices, containing different models. Afterwards the `bayesModelEvaluation` is called to run the cross validation and the calculation of *mse* and *lppd* for every model.

```
# Swiss data
dat <- swiss

# Response variable
Y <- dat$Fertility
n <- nrow(dat)

# Design matrices
models <- list(
  matrix(c(rep(1,n), dat$Education), nrow=n),
  matrix(c(rep(1,n), dat$Agriculture), nrow=n),
  matrix(c(rep(1,n), dat$Examination), nrow=n),
  matrix(c(rep(1,n), dat$Catholic), nrow=n),
  matrix(c(rep(1,n), dat$Education, dat$Agriculture), nrow=n),
  matrix(c(rep(1,n), dat$Education, dat$Examination), nrow=n),
  matrix(c(rep(1,n), dat$Education, dat$Catholic), nrow=n),
  matrix(c(rep(1,n), dat$Education, dat$Agriculture, dat$Examination), nrow=n),
  matrix(c(rep(1,n), dat$Education, dat$Agriculture, dat$Catholic), nrow=n),
  matrix(c(rep(1,n), dat$Education, dat$Examination, dat$Catholic), nrow=n),
  matrix(c(rep(1,n), dat$Education, dat$Agriculture, dat$Examination ,dat$Catholic),
         nrow=n)
)

# Run model evaluation
criteria <- bayesModelEvaluation(models, Y, R = 500, b = 100)
```

The results in figure 1 show that model 9 seems to be the best model with the lowest *mse* and the lowest *lppd*, compared to the other 10 models. To check this quantitatively we get the minimum of the *mse* values and the minimum of the *lppd* values.

```
print(which.min(criteria$MSEs))
```

```
## [1] 9
```

```
print(which.min(-2*criteria$LPPDs))
```

```
## [1] 9
```

Indeed, both indicators favor model 9. We know that according to (Gelman et al. 2014), *mse* is just appropriate if the model is more or less normally distributed. Furthermore if the models are normal and have constant variances, *mse* and *lppd* should have the same proportion.

The proportions are calculated in table 1. It can be seen, that the proportion is in the same scale in all cases, but there are deviations. For exmaple in model 2 and 9 the proportions are very different. This is probably due to the fact, that all the models are not well normally distributed.

Finally we store the index of the optimal model in a variable. In the next step we want to calculate Cook's distance just for the best mode.

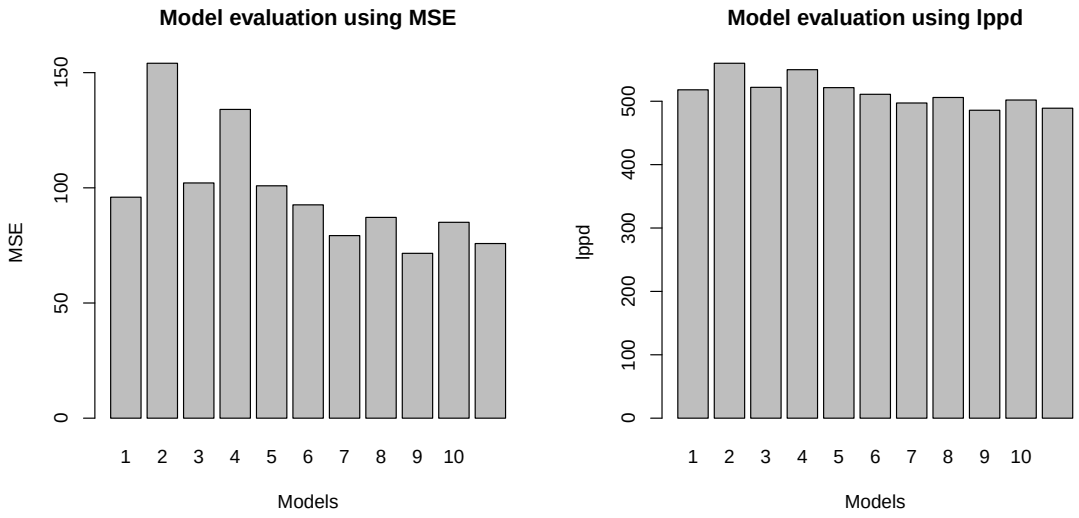

Figure 1: Model evaluation with leave-one-out cross validation (LOO-CV) using mean squared errors (MSE) and log pointwise predictive density (lppd)to obtain the best model.

	1	2	3	4	5	6	7	8	9	10	11
proportion	0.19	0.28	0.20	0.24	0.19	0.18	0.16	0.17	0.15	0.17	0.16

Table 1: Proportion of mse and lppd.

```
# Choose optimal modal
optIdx <- which.min(-2*criteria$LPPDs)
```

3.3 Cook's Distance

The second application of the cross validation is the approximate calculation of Cook's distance, which ist mathematically defined in equation (2.5). The function `cooksDistance` calculates the approximate solution, using loo-cv, and the analytic solution, based on frequentist linear regression with ordinary least squares.

To calculate the approximation of Cook's distance with cross validation we make use of the implemented `crossValidation` function above. We choose `R` as a vector, such that the cross validation function calculates multiple estimates for many different sample sizes. Finally we just apply the formula for Cook's distance.

For the analytic solution, we calculate a frequentist linear regression, where $\hat{\beta}$ was solved by ordinary least square approach. After we obtain the point estimatie, we can calculate Cook's distance.

In the end a list is returned, containing the analytic solution as a vector and the loo-cv solutions as a matrix, where the number of columns depend on the size of vector `R`.

```
cooksDistance <- function(X, Y, R, b) {
  # Calculate cooks distances for optimal model from model evaluaion
  # Two methods are used:
  #   (1) Use LOO-CV approach for bayesian linear regression model with OLS
  #   (2) Use analytic solution for frequentist linear regression model with OLS
  #
  # Args:
  #   X: Design matrix
  #   Y: Labels
  #   R: Number of draws (without burn-in), R can be a vector in this function
```

```
  #   b: Number of burn-in draws
  #
  # Returns:
  #   cooksBayesCV:   Cook's distances calculated by LOO-CV (bayes model),
  #                   where cook's distances are calculated for different
  #                   sample sizes
  #   cooksAnalytic:  Cook's distances calculated by analytic solution
  #   sample:         The gibbs sample from the bayes linear model
  #                   (for plotting purpose)

  # R is usually a vector (but don't has to)
  B <- R + b
  steps <- length(R)

  # Prepare projection matrix and number of parameters
  H <- X%*%solve(t(X)%*%X)%*%t(X)
  p <- ncol(X)

  # Run cross validation with vector R
  cv <- crossValidation(X, Y, R, b)

  # Sample whole model (add B = R + b samples, remove b later)
  sample <- gibbsSampler(X, Y, R[steps] + b)

  # Cook's distance: Frequentist appraoch with analytic solution
  betaHat <- solve(t(X)%*%X)%*%t(X)%*%Y
  YhatLinReg <- X%*%betaHat
  E <- Y-YhatLinReg
  cooksAnalytic <- (E^2/((1/(n-p))*sum(E^2)*p))*(diag(H)/(1-diag(H))^2)

  # Cook's distance: Bayesian appraoch with LOO-VC solution
  cooksBayesCV <- matrix(nrow = n, ncol = steps)
  for(k in 1:steps) {
    # Estimate posterior mean from betas
    betas <- colMeans(sample$betas[b:B[k],])

    # Predict values, using posterior mean
    Yhat <- X%*%betas

    # Calculate cook's distance
    dists <- apply(cv$Betas[[k]], 2, function(betas) {
      return((Yhat - X%*%betas)^2)
    })
    mse <- (1/(n-p))*sum((Y-Yhat)^2)
    cooksBayesCV[,k] <- colSums(dists)/(p*mse)
  }

  return(list(cooksBayesCV = cooksBayesCV,
              cooksAnalytic = cooksAnalytic,
              sample = sample))
}
```

We apply the function above to the optimal model, obtained by the model evaluation. We choose a specific sequence of sample sizes `R` and a burn-in size.

```
# Number of samples
b <- 100
R <- seq(100,5000,50)
```

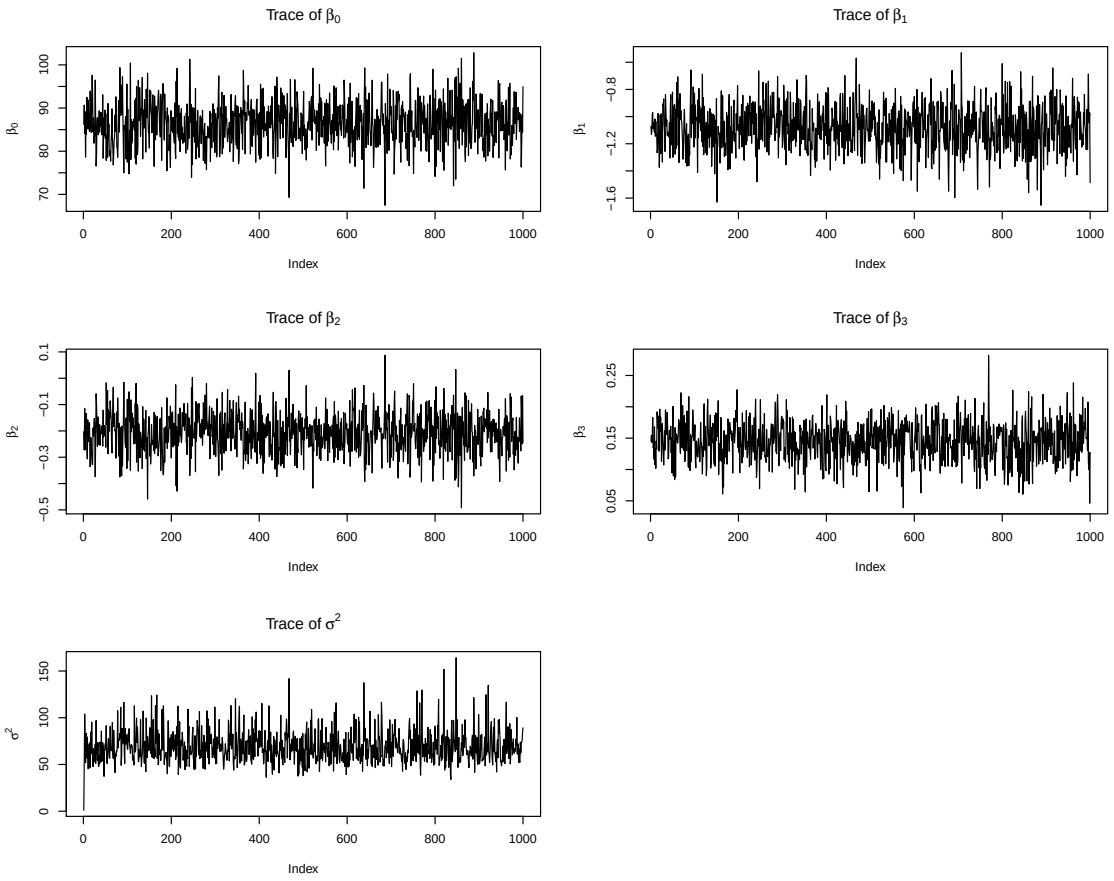

Figure 2: Traces of sampled parameters from posterior distributions regarding bayesian linear regression with ordinary least squares.

```
# Get optimal model and calculate projection matrix
cooks <- cooksDistance(models[[optIdx]], Y, R, b)
```

In the first step, we want to have a look at Cook's distances. In figure 4 the blue bars in all 3 plots are always the same analytic solution. They are compared to the loo-cv solution using the Bayes approach, which is shown in red. In the first plot 100 simulations were conducted, in the second plot 1050 and 5000 in the last plot. We can observe, that the more simulation are done the loo-cv solution is coming closer to the analytic solution.

To see the convergence behavior of the loo-cv solution to the analytic solution, we developed a measure d caluclating the distance between both solutions. The Cook's distance value for both solutions were subtracted, squared and summed (squared distance) as follows:

$$d = \sum_{i=1}^{n} (D_i^{\text{loo-cv}} - D_i^{\text{analytic}})^2, \tag{3.1}$$

where n is the number of data points, $D_i^{\text{loo-cv}}$ Cook's distance for the loo-cv solution and D_i^{analytic} Cook's distance for the analytic solution for point X_i.

```
# Initialize matrices
steps <- length(R)
matBayesD <- cooks$cooksBayesCV
matFreqD <- matrix(rep(cooks$cooksAnalytic, steps), ncol=steps)

# Calculate differences between Cook's distance methods
```

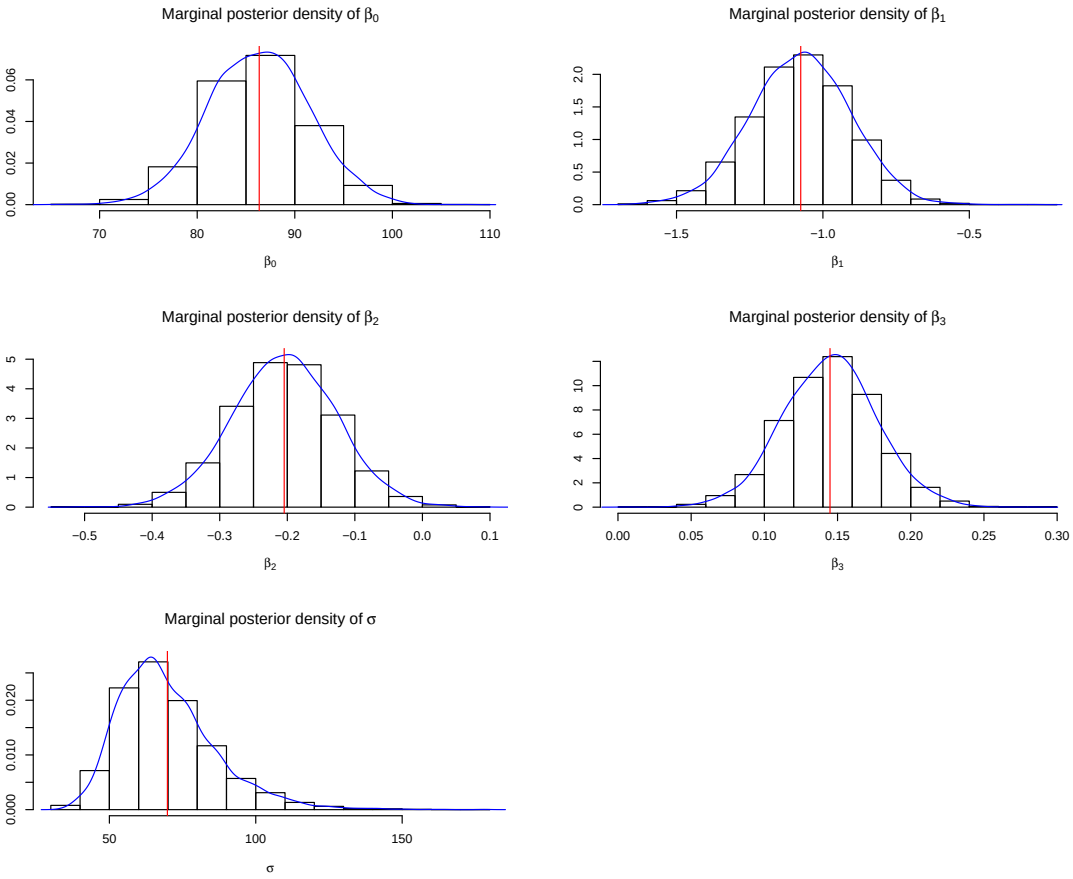

Figure 3: Posterior densities, histograms and posterior means (red vertical line) of sampled parameters regarding bayesian linear regression with ordinary least squares

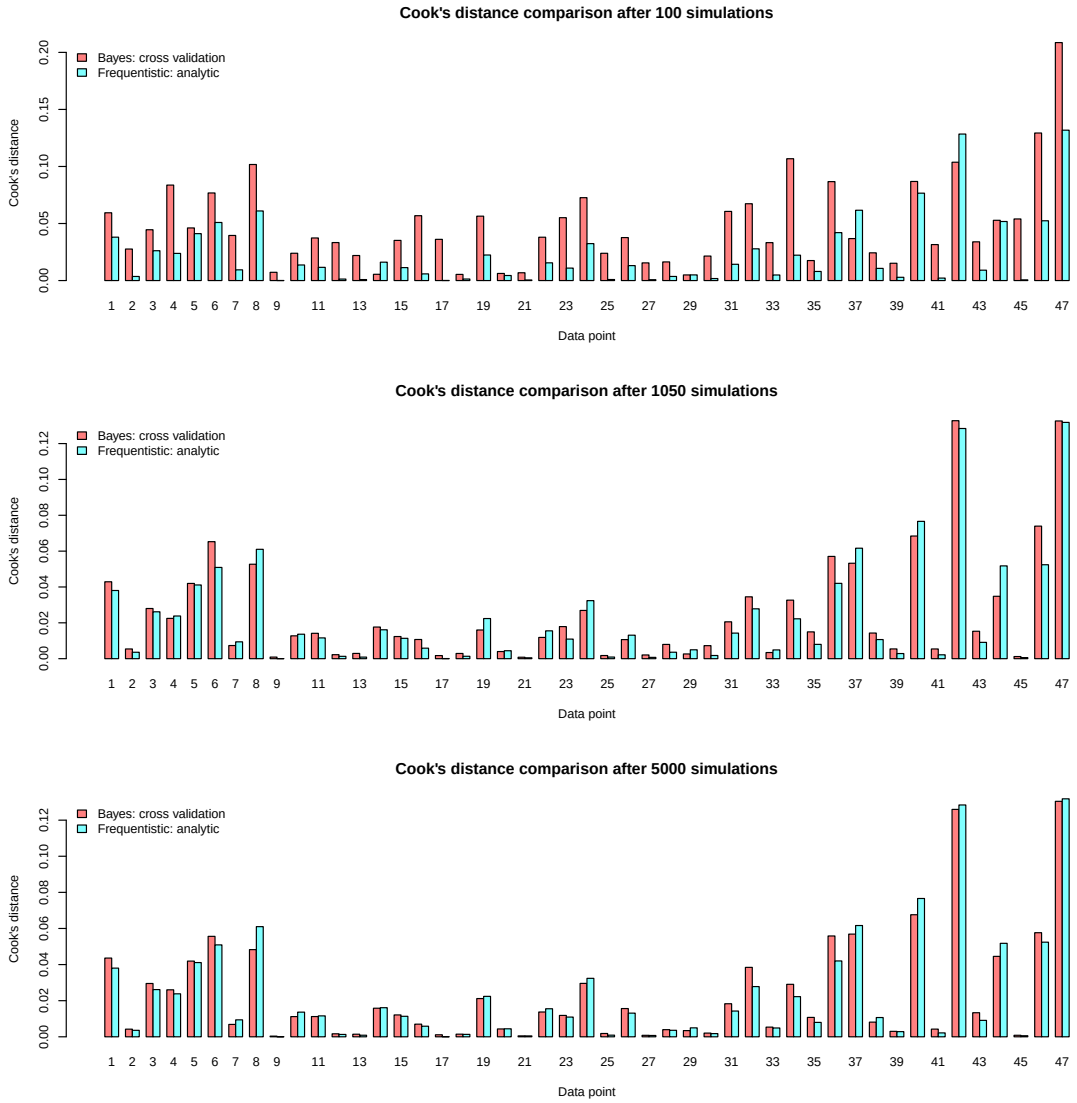

Figure 4: Pointwise cook's distance, calculated with three different sample sizes, compared to analytic cook's distance from frequentist linear regression model using least squares.

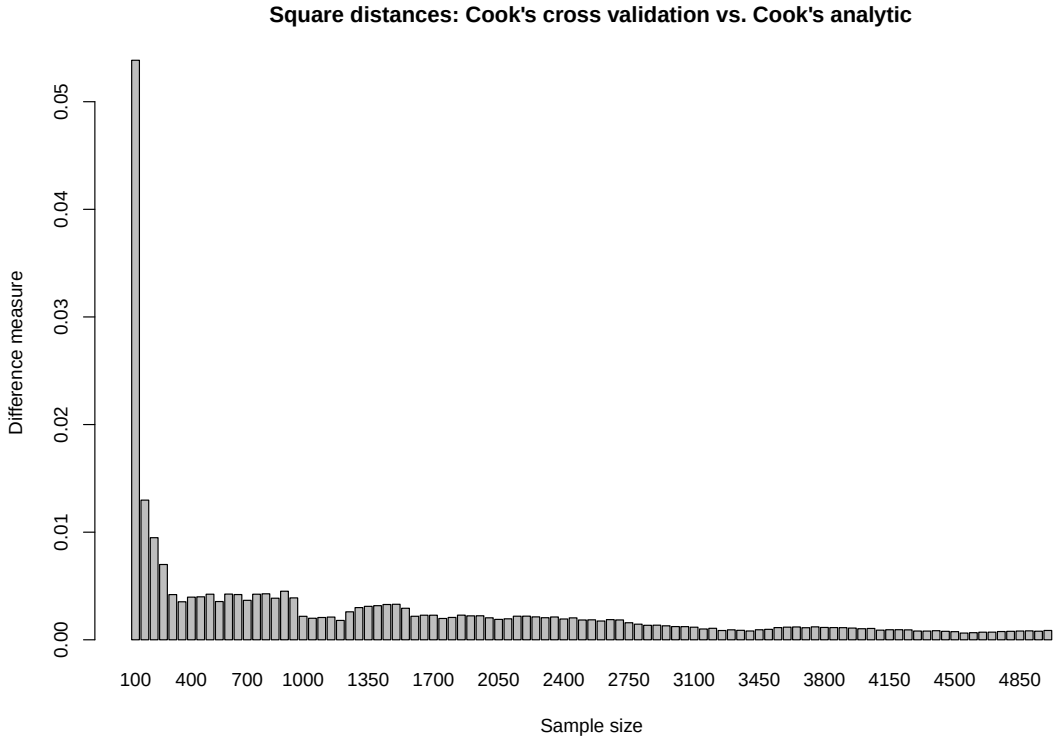

Figure 5: Cook's distance measure to quantify approximation from cross validated cook's distances of analytic cook's distances. The approximate solution with LOO-CV comes closer to the analytic solution, the more samples are drawn.

```
cooksMeasure <- colSums((matBayesD-matFreqD)^2)
```

In figure 5 we can see that the defined distance d becomes lower. In other words: both solutions are coming closer together the more simulations are done. Therefore a convergence behavior can be seen.

References

Cook, R Dennis. 1977. "Detection of Influential Observation in Linear Regression." *Technometrics* 19 (1). Taylor & Francis: 15–18.

Gelman, Andrew, John B Carlin, Hal S Stern, David B Dunson, Aki Vehtari, and Donald B Rubin. 2014. *Bayesian Data Analysis*. Vol. 3. CRC press Boca Raton, FL.

Hoeting, Jennifer A, and Joseph G Ibrahim. 1998. "Bayesian Predictive Simultaneous Variable and Transformation Selection in the Linear Model." *Computational Statistics & Data Analysis* 28 (1). Elsevier: 87–103.

Hoff, Peter D. 2009. *A First Course in Bayesian Statistical Methods*. Springer Science & Business Media.

Vehtari, Aki, Andrew Gelman, and Jonah Gabry. 2017. "Practical Bayesian Model Evaluation Using Leave-One-Out Cross-Validation and Waic." *Statistics and Computing* 27 (5). Springer: 1413–32.

Wasserman, Larry. 2013. *All of Statistics: A Concise Course in Statistical Inference*. Springer Science & Business Media.