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

Master Statistics
August 23, 2017

Contents

List of Figures

List of Tables

1 Introduction

2 Theory

2.1 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 the 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 prediction density* using the loo-cv, which is defined as

$$\text{lppd}_{\text{loo-cv}} = \sum_{i=1}^{n} \log p_{\text{dens}(-i)}(y_i). \tag{2.1}$$

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, 2013). Besides the MSE we will use $-2\ \text{lppd}_{\text{loo-cv}}$ for model selection within the Bayesian framework.

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.

2.2 Cooks's Distance

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

$$D_i = \frac{\sum_{j=1}^{n}\left(\hat{y}_j - \hat{y}_j^{(i)}\right)^2}{p\ \text{MSE}} \overset{\text{(OLS)}}{=} \frac{e_i^2}{p\ \text{MSE}}\left(\frac{h_{ii}}{(1-h_{ii})^2}\right), \tag{2.2}$$

where $\text{MSE} = \frac{1}{n-p}\sum_{i=1}^{n}(y_i - \hat{y})^2$ is the *mean squared error*, $e_i^2 = (y_i - \hat{y})^2$ is the error for each observation i and $h_{ii} = x_i(X^TX)^{-1}x_i^T$ 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 the model 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 to a great advantage in performance, especially for large numbers of observations. But the analytical expression is constraint 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. Those two functions are explained in the first part. The second part is about the model evaluation and the third part compares the loo-cv Cook's distance with the analytic solution, which are defined in equation (2.2).

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

3.1 Basics

The first function we define ist the *Gibbs sampler*. The Gibbs sampler draw the parameters $\beta_0, ..., \beta_p, \sigma^2$ from their posterior distributions. Those distributions (see equation (??) and (??)) need the design matrix `X` and labels `Y`, which are given as arguments. Furthermore the Gibbs sampler is not just drawing 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.

```r
gibbsSampler <- function(X, Y, R, b = 0, initSigma = 1) {
  # Sample parameters from posterior distribution
  # Prepared for baysean linear regression with ordinary least square approach (OLS)
  #
  # 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
  #   initSigma: Initial value of sigma (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 (initalize sigma with initSigma)
  betas <- matrix(nrow = B, ncol = p)
  sigma <- c(initSigma, rep(NA, B-1))

  # Sampling
  for(i in 1: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

    # OLS of sigma
    sigma_hat <- t(Y-X%*%beta_hat)%*%(Y-X%*%beta_hat)/(n-p)

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

    # Sample sigma from the full conditional
    if(i < B) {
      sigma[i+1] <- 1/rgamma(1, (n-p)/2, (n-p)*sigma_hat/2)
    }
  }

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

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

As next, we define a function doing the leave-one-out cross validation. It delets one data point and samples the parameters from the posterior distribution (using the `gibbsSampler` function). It is done 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 predict the labels, using the data matrix X and the point esimtators of the parameters.

Note that the argument R can be a scalar or a vector. If R is a scalar, the cross validation just runs usual. If R is a ordered vector the Gibbs sampler draws as often as the last value of the R vector. The size of the vector is indicated by 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.

```r
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 vector (ordered) 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 evaluaion

```r
bayesModelEvaluation <- function(models, Y, R, b) {
  # Model evaluation based on mean squared error (MSE) and log posterior
  # 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 posterior 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) sum(log(dnorm(Y, eva$Yhati[[1]], eva$Sigma[[1]]))))

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

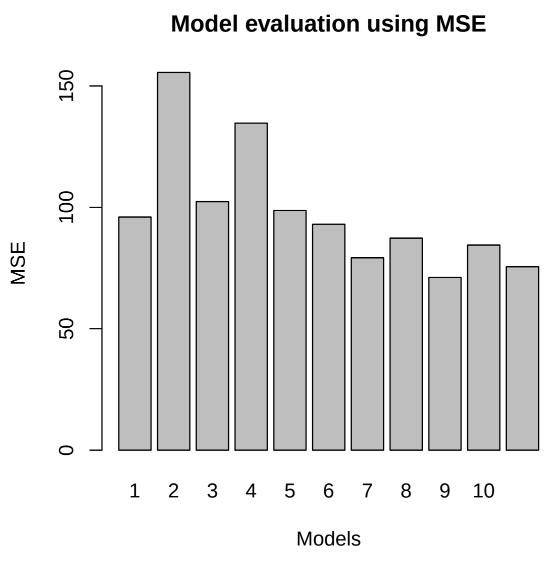
Run model evaluation with data

```r
# 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)
)
```



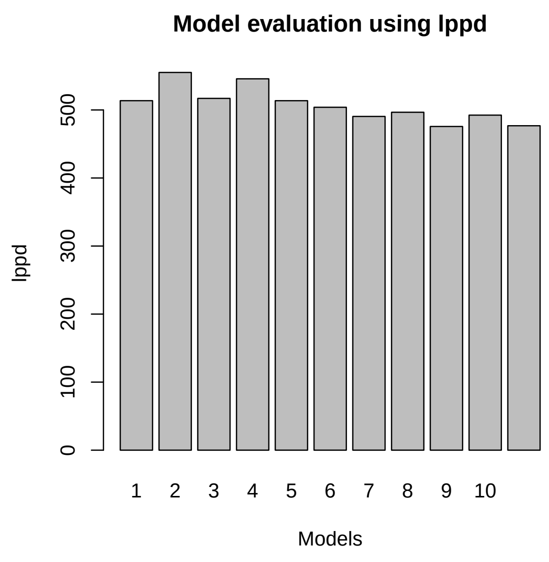


Figure 1: Model evaluation with leave-one-out cross validation (LOO-CV) using mean squared errors (MSE) and log posterior predictive density (lppd) to obtain best model.

```r
# Run model evaluation
criteria <- bayesModelEvaluation(models, Y, R = 50, b = 10) # R = 500, b = 100
```

```r
# Check proportion
print(criteria$MSEs/(-2*criteria$LPPDs))
```

```
##  [1] 0.1869484 0.2802171 0.1979397 0.2467844 0.1921273 0.1846834 0.1614863
##  [8] 0.1758955 0.1496473 0.1715877 0.1583839
```

```r
# TODO: Als Tabelle ausgeben? Explain why not fulfilled

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

# Check if MSE and LPPD would choose the same
print(paste(which.min(criteria$MSEs), "=", which.min(-2*criteria$LPPDs)))
```

```
## [1] "9 = 9"
```

```r
# TODO den Vergleich einfach nur im Text verwenden?
```

3.3 Cook's Distance

```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:
```

```r
  #   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))
}
```

Calculate cook's distances

```r
# Number of samples
b <- 100
R <- seq(100,500,50) # seq(100,5000,50)

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

Traces from bayes regression sampling

Posterior densities from bayes regression sampling

Apply cook's distance measure

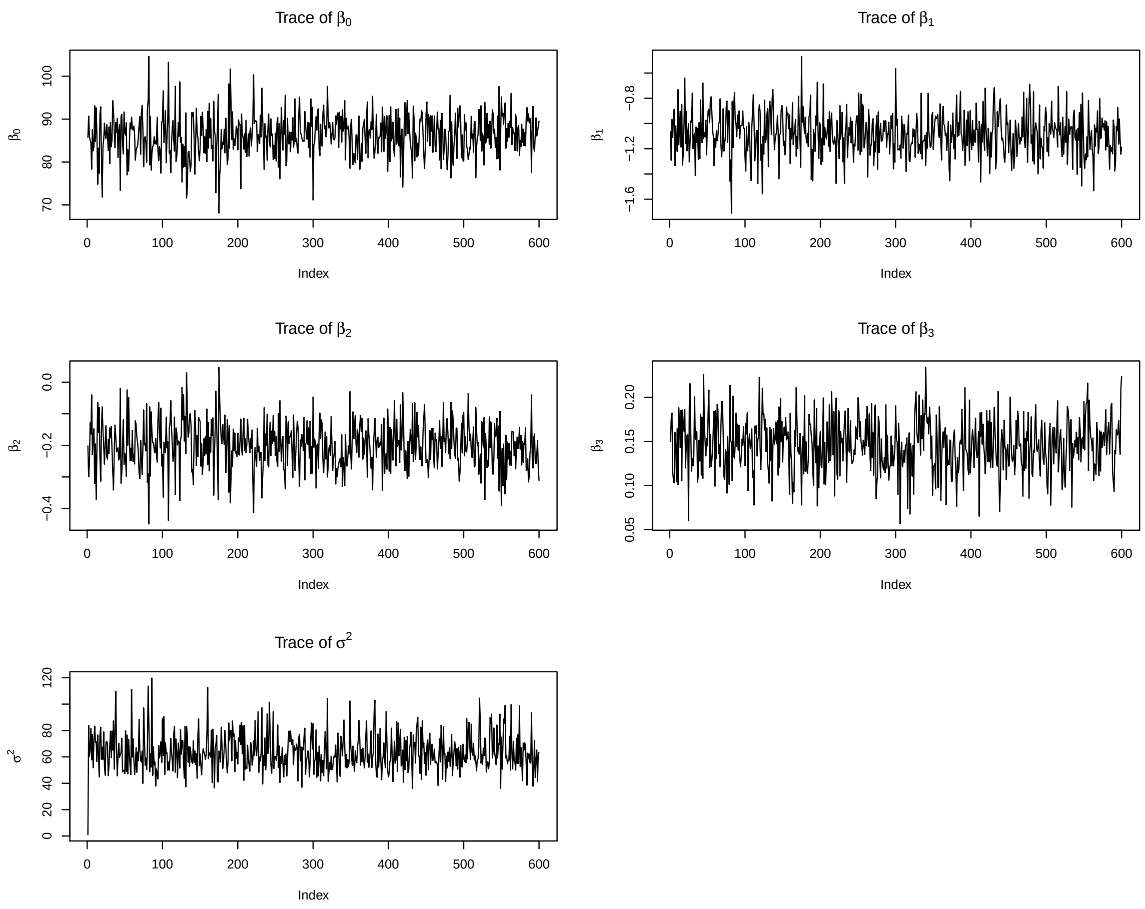


Figure 2: Traces of sampled parameters from posterior distributions regarding bayesian linear regression with ordinary least squares.

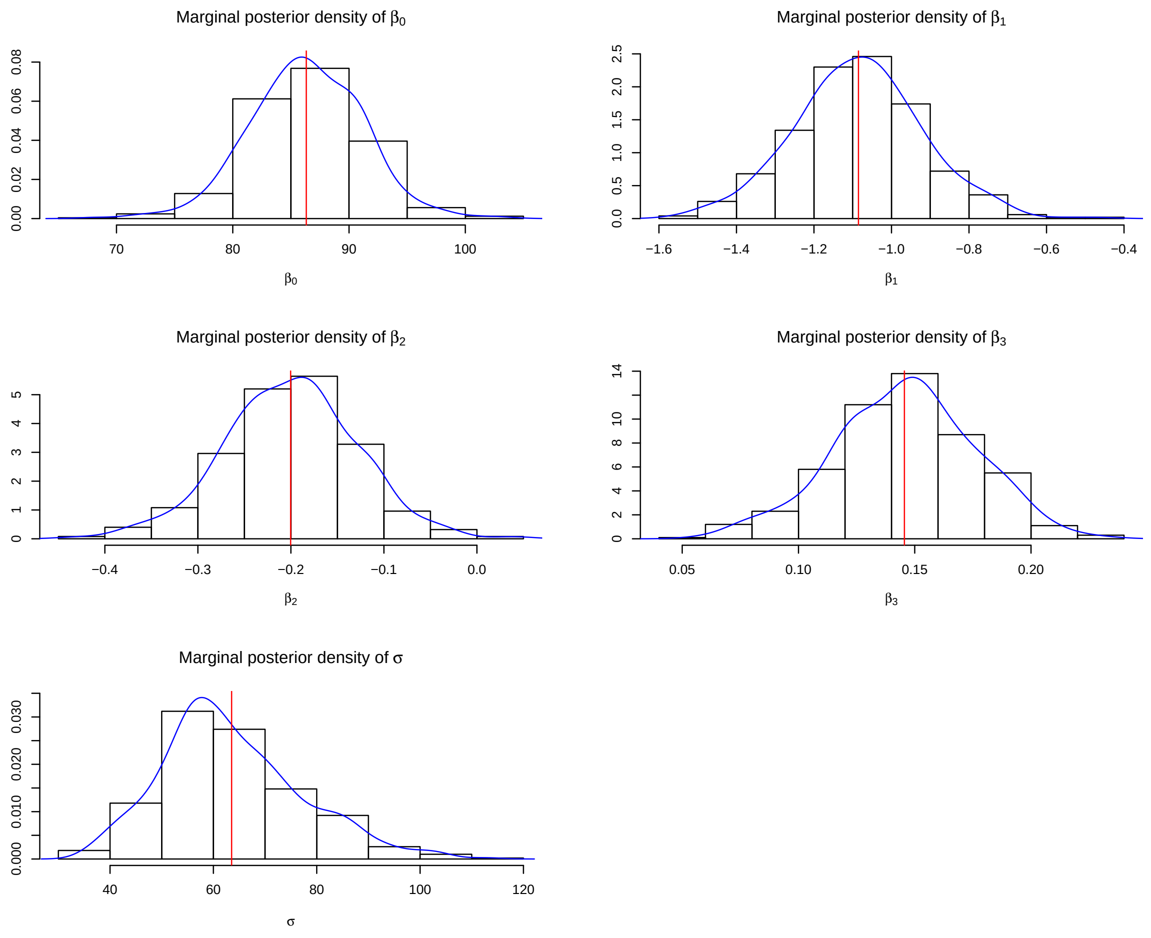


Figure 3: Posterior densities, histograms and posterior means (red vertical line) of sampled parameters regarding bayesian linear regression with ordinary least squares

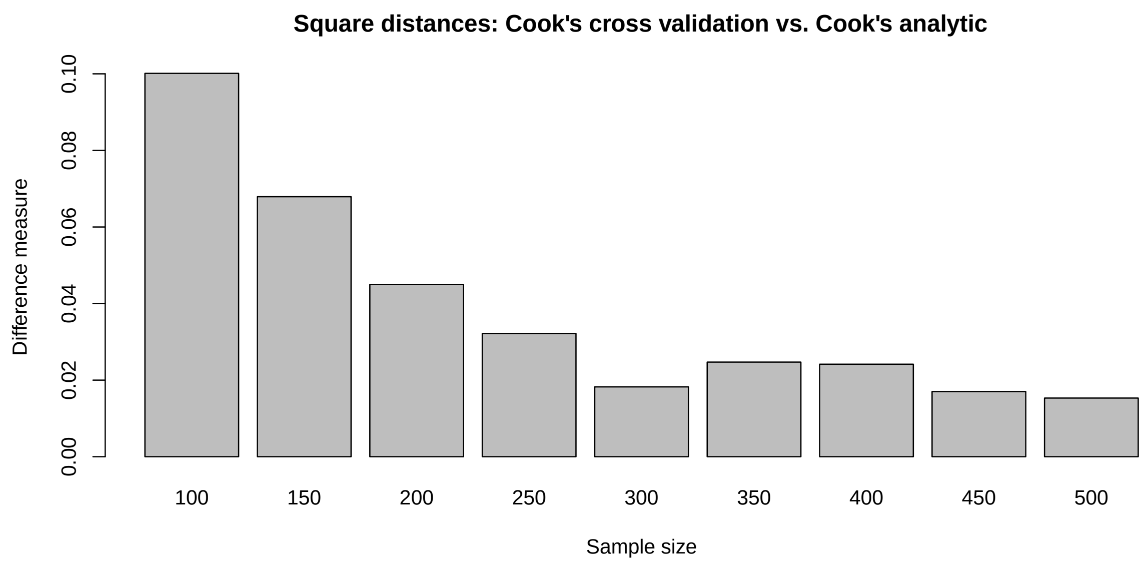


Figure 4: Cook's distance measure to quantify approximation from cross validated cook's distances of analytic cook's distances. The approximate solution with LOO-CV comes closer to the analytic solution, the more samples are drawn.

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

# Calculate differences between Cook's distance methods
cooksMeasure <- colSums((matBayesD-matFreqD)^2)
```

Cook's distance measure

Cook's distance, comarison between bayes and frequentist

4 References

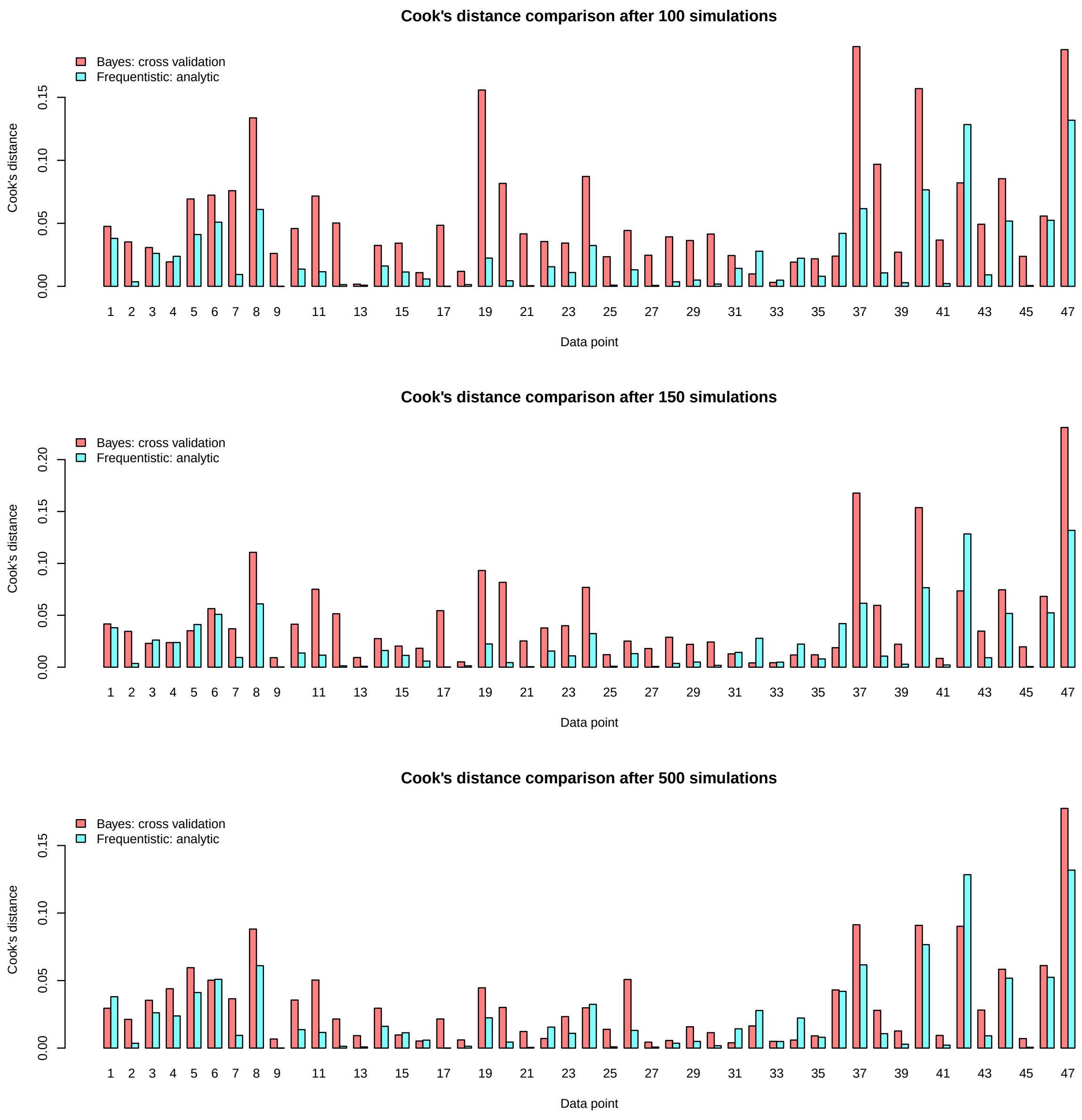


Figure 5: Pointwise cook's distance, calculated with three different sample sizes, compared to analytic cook's distance from frequentist linear regression model using least squares.