Computer assignment 3 - Theory of Statistical Inference

Regina Crespo Lopez Oliver (20000322-8806) & Malin Mueller (20011115-T460)

2024-09-27

```
  # Byt ÅÅMMDD mot ditt födelsedatum
  set.seed(011115) # - Malin
  # set.seed(000322) # - Regina
```

```
load("proj_data.Rdata")
modell <- glm(Resultat ~ Alder + Kon + Utbildare,
              data = data_individ,
              family = "binomial")
summary(modell)
```

```
##
## Call:
## glm(formula = Resultat ~ Alder + Kon + Utbildare, family = "binomial",
##     data = data_individ)
##
## Coefficients:
##                       Estimate Std. Error z value Pr(>|z|)
## (Intercept)           0.151971   0.249661   0.609 0.542716
## Alder                -0.031394   0.008677  -3.618 0.000297 ***
## KonMan                0.090185   0.136394   0.661 0.508476
## UtbildareTrafikskola  0.916541   0.157659   5.813 6.12e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1353  on 999  degrees of freedom
## Residual deviance: 1297  on 996  degrees of freedom
## AIC: 1305
##
## Number of Fisher Scoring iterations: 4
```

```
p_var <- function(theta, X){
  # function to compute probabiloty recieving theta and X
  if (length(theta) > 1){
    p_res <- 1 / (1 + exp(-X%*%theta))
  }
  else{
    p_res <- 1 / (1 + exp(-X*theta))
  }
```

```
  return(p_res)
}

L <- function(theta, y, X){
  ## likelihood
  p_var <- p_var(theta = theta, X = X)
  #likelihood <- p_var^y %*% (1 - p_var)^(1 - y)
  likelihood <- prod(p_var^y * (1 - p_var)^(1 - y))

  return(likelihood)
}

l <- function(theta, y, X){
  ## Same thing as before, but log likelihood
  ## log likelihood
  p_var <- p_var(theta = theta, X = X)
  log_likelihood <- sum(y * log(p_var) + (1 - y) * log(1 - p_var))
  # log_likelihood <- dbninom()
  return(log_likelihood)
}

S <- function(theta, y, X){
  ## Score function
  p_var <- p_var(theta = theta, X = X)
  score <- t(X) %*% (y-p_var)
  return(score)
}

v <- function(theta, X){
  ## Vi
  p <- p_var(theta = theta, X = X)
  v_res <- p * (1-p)
  return(v_res)
}

I <- function(theta, y, X){
  ## fisher information
  v_var = v(theta,X)
  D <- diag(as.vector(v_var))
  fisher <- t(X) %*% D %*% X
  return(fisher)
  }

NR <- function(theta0, niter, y, X){
  # function that applies Newton-Raphson?s algorithm in order to compute the
  # ML-estimates in a logistic regression model, in a certain number of n
  # iterations.

  #Note: this might break if the matrix dim from x changes
    theta <- matrix(theta0, nrow = length(theta0), ncol = 1)

  for (i in 1:niter){
    score <- S(theta, y, X)
```

```
    log_likelihood <- L(theta, y, X)
    #theta <- theta + (log_likelihood/score)
    theta <- theta + solve(I(theta, y, X)) %*% score # its a plus and not a
    # minus because of how we obtain the derivative of the score function
  }

  return(theta)
}
```

Task1

Compute AIC for the model using functions from previous assignment and make sure it agrees with R's value. You should also compute the corresponding value based on leave-one-out cross validation (see textbook (7.9)).

```
y <- matrix(data_individ$Resultat, ncol = 1)
X <- model.matrix(Resultat ~ Alder + Kon + Utbildare,
                  data = data_individ)

theta0 <- coef(modell)
MLE <- NR(theta0, 10, y, X)
log_likelihood <- l(MLE, y, X)
num_params <- length(coef(modell))  # Number of estimated parameters
aic_manual <- -2 * log_likelihood + 2 * num_params
cat("Manual AIC:", aic_manual, "\n")
```

```
## Manual AIC: 1304.958
```

```
cv_log_likelihood <- 0

n <- nrow(data_individ)  # Number of observations

# Extract the response vector 'y' and the design matrix 'X' from your data
# y <- data_individ$Resultat
# X <- model.matrix(~ Alder + Kon + Utbildare, data = data_individ)

# Loop through each observation for leave-one-out cross-validation
for (i in 1:n) {
  # Create the training set by excluding the i-th observation
  X_train <- X[-i, ]
  y_train <- y[-i]

  # Initial guess for MLE
  theta0 <- coef(modell)

  #MLE for train data
  MLE_i <- NR(theta0, 10, y_train, X_train)

  # Calculate the log-likelihood for the left-out i-th obs
  # but using the MLE calculated from the training data

  # extract i-th obs
  X_test <- X[i, , drop = FALSE]
```

```
  y_test <- y[i]

  # Compute the log-likelihood for the left-out observation
  # from formula 7.9 the resulting cross-validated average log-likelihood
  # is a sum of all of the i likelihoods
  cv_log_likelihood <- cv_log_likelihood + l(MLE_i, y_test, X_test)
}

# Compute the cross-validated average log-likelihood (divide by n obs)
kcv <- cv_log_likelihood / n
cat("Cross-validated log-likelihood:", kcv, "\n")
```

```
## Cross-validated log-likelihood: -0.6525048
```

```
# Use kcv to calculate the LOOCV AIC :
# AIC = -2 * log-likelihood + 2 * number of parameters
num_params <- length(theta0)  # Number of estimated parameters = 4
loo_cv_aic <- -2 * cv_log_likelihood + 2 * num_params
cat("LOO-CV AIC:", loo_cv_aic, "\n")
```

```
## LOO-CV AIC: 1313.01
```

This AIC value almost matches the true value - it may be because it is more generalized since it incorporates how well the model works on unseen data (x_i in this case).

Task 2

Write a function post <- function(theta, y, X){...} that evaluates the posterior density (up to a multiplicative constant) at theta, given y and X as in previous assignments.

```
# Function to compute posterior density up to a constant
post <- function(theta, y, X) {
  # doing the posterior in terms of loglikehood to make operations easier
  eta <- X %*% theta

  # Log-likelihood (numerically stable form)
  log_likelihood <- sum(y * eta - log(1 + exp(eta)))

  #log_likelihood <- l(theta, y, X)


  # Prior: N(0, 100I) -> log density of a normal distribution
  log_prior <- -0.5 * sum(theta^2 / 100)

  return(log_likelihood+log_prior)
}

Xtest <- cbind(1, 18:25, rep(c(0, 1), 4), rep(c(1, 1, 0, 0), 2))
ytest <- c(rep(TRUE, 4), rep(FALSE, 4))
# do the exponential to get it back in terms of likelihood
exp(post(c(260, -10, 10, -20), ytest, Xtest)) / exp(post(c(270, -15, 15, -25),
                                                         ytest , Xtest))
```

```
## [1] 3.707555e+25
```

This is the same result as expected, which proves that our procedure is correct.

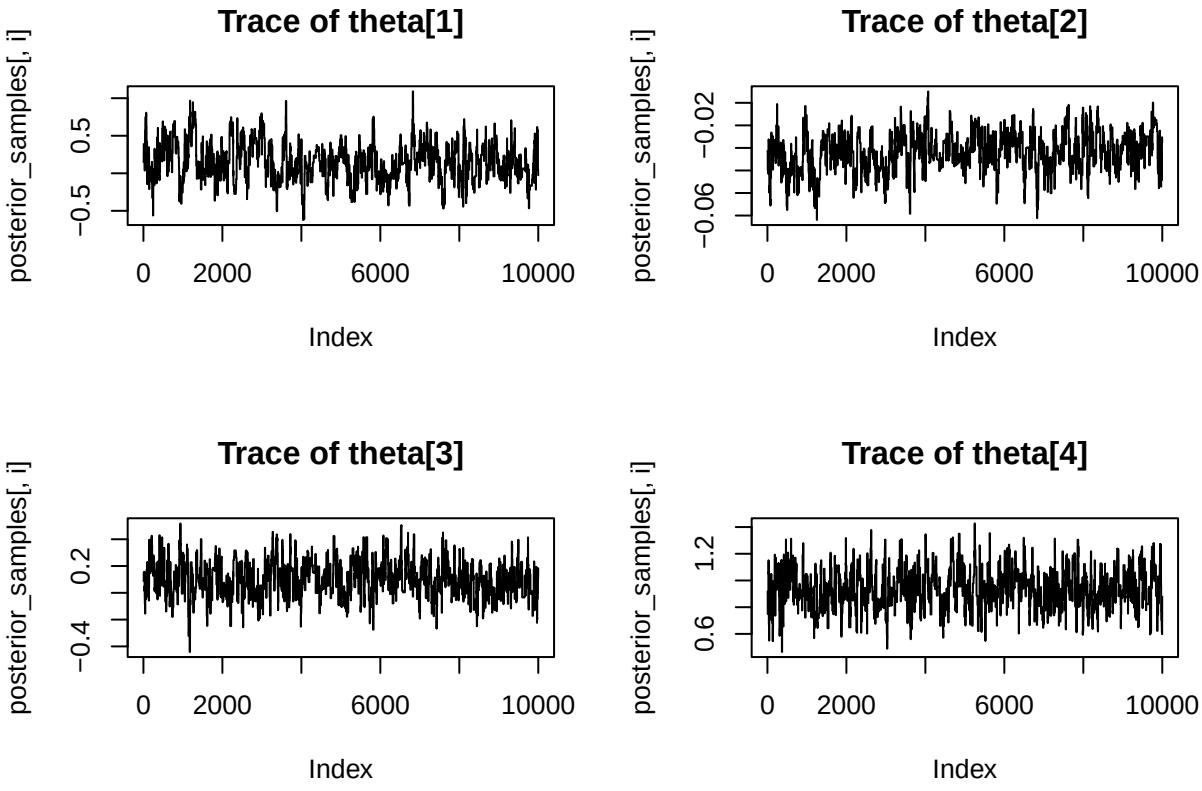
Task 3

The Metropolis-Hastings algorithm generates a sequence of samples that approximate the posterior distribution - constructs a Markov chain — a sequence of parameter values where each value only depends on the previous one. After many iterations: Over time, the sequence of theta values will approximate the posterior distribution.

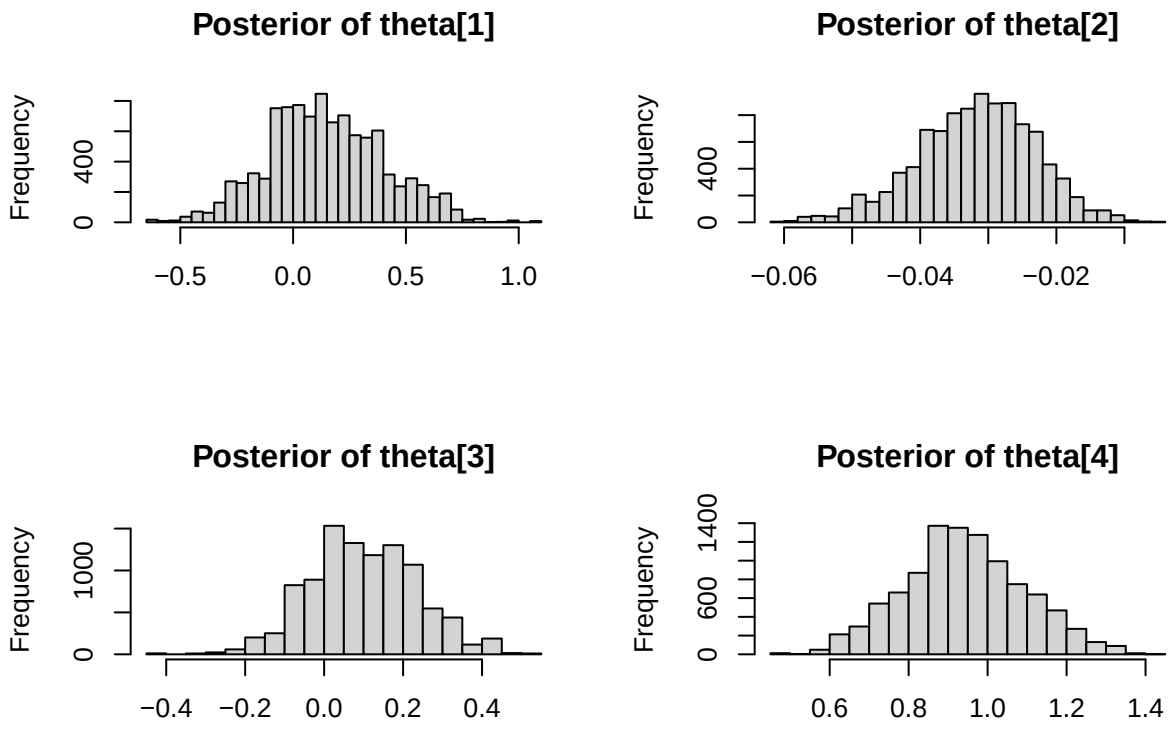
```
mh_algorithm <- function(theta, y, X, sigma, n_iter = 10000) {
  # Number of parameters
  n_params <- length(theta)

  # Store the samples
  theta_samples <- matrix(NA, nrow = n_iter, ncol = n_params)
  theta_samples[1, ] <- theta  # Set the starting point

  # Sample a candidate point or proposal from the proposal distribution
  for (t in 2:n_iter) {
    # Previous theta
    theta_prev <- theta_samples[t - 1, ]

    # Sample candidate point theta* from normal distribution (proposal)
    #theta_star <- rnorm(n_params, mean = theta_prev, sd = proposal_sd)

    theta_star <- theta_samples[t-1,] + rnorm(n_params) * sigma
    # Compute acceptance probability (numerically stable)
    alpha <- exp(post(theta_star, y, X))/exp(post(theta_prev, y, X))
    alpha <- min(1, alpha)

    # Draw a uniform random number
    u <- runif(1)

    # Accept or reject the candidate
    if (u <= alpha) {
      theta_samples[t, ] <- theta_star  # Accept the candidate
    } else {
      theta_samples[t, ] <- theta_prev  # Reject and keep the previous value
    }
  }

  return(theta_samples)  # Return the matrix of samples
}
```

```
mle_val <- NR(coef(modell), 10, y, X)
mle_se <- summary(modell)$coefficients[,"Std. Error"]

# Run the Metropolis-Hastings algorithm
posterior_samples <- mh_algorithm(mle_val, data_individ$Resultat, X, mle_se,
                                  n_iter = 10000)
```

```r
par(mfrow = c(2, 2))  # 2x2 grid for plots
for (i in 1:ncol(posterior_samples)) {
  plot(posterior_samples[, i], type = "l",
       main = paste("Trace of theta[", i, "]", sep=""))
}
```




```r
par(mfrow = c(2, 2))  # 2x2 grid for plots
for (i in 1:ncol(posterior_samples)) {
  hist(posterior_samples[, i], breaks = 30,
       main = paste("Posterior of theta[", i, "]", sep=""), xlab = "")
}
```




```
posterior_means <- colMeans(posterior_samples)
cred_intervals <- apply(posterior_samples, 2, function(x) quantile(x,
                                                    probs = c(0.025, 0.975)))

posterior_means  # Posterior means
```

```
## [1]  0.14694354 -0.03175377  0.10010683  0.94163941
```

```
cred_intervals  # 95% credible intervals using quantiles
```

```
##             [,1]        [,2]       [,3]      [,4]
## 2.5%  -0.3417676 -0.05006742 -0.1547903 0.6488869
## 97.5%  0.6655177 -0.01584718  0.3920263 1.2446546
```

```
x_star <- c(1, 22, sex= 0, 1)  #

#collecting the samples that correspond to x star
p_star_samples <- p_var(x_star, posterior_samples)

#probability of a pass using posterior samples
p_star_mean <- mean(p_star_samples)  # Posterior mean of P(Y* = 1 | y)

p_star_mean
```

```
## [1] 0.5958111
```