Bayesian Logistic Regression

Alex B.Dunbar

8/9/2021

Clear environment, graphics and console

```
# Clear environment ####
rm(list = ls())

# Clear plots
graphics.off()  # Clears plots, closes all graphics devices

# Clear console
cat("\014")  # ctrl+L
```

Install required packages

1. Introduction

This is a Bayesian logistic regression related to red variants of the Portuguese "Vinho Verde" wine. There are 11 input variables (covariates) based on physiochemical tests. The output variable is `quality` and is a score between 0 and 10 which is based on sensory data.

2. Import data.

The dataset is described in the publication by Cortez, P., Cerdeira, A., Almeida, F., Matos, T., & Reis, J. (2009). Modeling wine preferences by data mining from physicochemical properties..

```
wine <- read.csv("winequality-red.csv")
head(wine)
```

```
##   fixed.acidity volatile.acidity citric.acid residual.sugar chlorides
## 1           7.4             0.70        0.00            1.9     0.076
## 2           7.8             0.88        0.00            2.6     0.098
## 3           7.8             0.76        0.04            2.3     0.092
## 4          11.2             0.28        0.56            1.9     0.075
## 5           7.4             0.70        0.00            1.9     0.076
## 6           7.4             0.66        0.00            1.8     0.075
##   free.sulfur.dioxide total.sulfur.dioxide density   pH sulphates alcohol
## 1                  11                   34  0.9978 3.51      0.56     9.4
## 2                  25                   67  0.9968 3.20      0.68     9.8
## 3                  15                   54  0.9970 3.26      0.65     9.8
```

```
## 4                 17                  60  0.9980 3.16      0.58      9.8
## 5                 11                  34  0.9978 3.51      0.56      9.4
## 6                 13                  40  0.9978 3.51      0.56      9.4
##   quality
## 1       5
## 2       5
## 3       5
## 4       6
## 5       5
## 6       5
```

Check for missing values

There are no `NA` values in the dataset.

```
any(is.na(wine))
```

```
## [1] FALSE
```

3. Classify "good" wines

We want to implement a logistic regression, therefore we want a response variable which assumes values either 0 or 1. Suppose we consider "good" a wine with quality above 6.5 (included).

We add a column called `good` that is `1` if the `quality` is greater than 6.5 and `0` if `quality` is less than 6.5.

We also add a column of strings called `drinkable`, relating 1 and 0 to "good" and "bad" as appropriate.

```
 wine %<>%
  mutate(good = ifelse(quality > 6, 1, 0)) %>%
  mutate(drinkable = ifelse(quality > 6, "good", "bad")) %>%
  mutate(quality = factor(quality, levels=1:10))
```

Remove spaces from names to make references easier.

```
colnames(wine) <- c("fixed_acidity", "volatile_acidity", "citric_acid",
                    "residual_sugar", "chlorides", "free_sulfur_dioxide",
                    "total_sulfur_dioxide", "density", "pH", "sulphates",
                    "alcohol", "quality", "good", "drinkable")
```

The structure of the dataset, `str(wine)`, will confirm that there are no `NA` values, the feature names, the feature type (num, Factor, chr) and the first 10 values.

```
str(wine)
```

```
## 'data.frame':    1599 obs. of  14 variables:
##  $ fixed_acidity       : num  7.4 7.8 7.8 11.2 7.4 7.4 7.9 7.3 7.8 7.5 ...
##  $ volatile_acidity    : num  0.7 0.88 0.76 0.28 0.7 0.66 0.6 0.65 0.58 0.5 ...
##  $ citric_acid         : num  0 0 0.04 0.56 0 0 0.06 0 0.02 0.36 ...
##  $ residual_sugar      : num  1.9 2.6 2.3 1.9 1.9 1.8 1.6 1.2 2 6.1 ...
```

```
##  $ chlorides           : num  0.076 0.098 0.092 0.075 0.076 0.075 0.069 0.065 0.073 0.071 ...
##  $ free_sulfur_dioxide : num  11 25 15 17 11 13 15 15 9 17 ...
##  $ total_sulfur_dioxide: num  34 67 54 60 34 40 59 21 18 102 ...
##  $ density             : num  0.998 0.997 0.997 0.998 0.998 ...
##  $ pH                  : num  3.51 3.2 3.26 3.16 3.51 3.51 3.3 3.39 3.36 3.35 ...
##  $ sulphates           : num  0.56 0.68 0.65 0.58 0.56 0.56 0.46 0.47 0.57 0.8 ...
##  $ alcohol             : num  9.4 9.8 9.8 9.8 9.4 9.4 9.4 10 9.5 10.5 ...
##  $ quality             : Factor w/ 10 levels "1","2","3","4",..: 5 5 5 6 5 5 5 7 7 5 ...
##  $ good                : num  0 0 0 0 0 0 0 1 1 0 ...
##  $ drinkable           : chr  "bad" "bad" "bad" "bad" ...
```

4. Frequentist Logistic Regression

The logistic model is computed using the generalised linear model (glm) with `good` being a function of all other covariates except `quality` and `drinkable`.

```
fit <- glm(
  good ~ . - quality - drinkable,
  data = wine,
  family = binomial(link="logit")
)
```

Significant coefficients

The significant coefficients are labelled in the last column of the summary() function. Most significant coefficients are:

- `sulphates` ($\Pr(\alpha > 6.924 = 4.39\text{e-}12)$),
- `alcohol`($\Pr(\alpha > 5.724 = 1.04\text{e-}08)$),
- `volatile_acidity` ($\Pr(\alpha > |\text{-}3.291| = 9.99\text{e-}4)$), and
- `total_sulfur_dioxide` ($\Pr(\alpha > |\text{-}3.378| = 7.31\text{e-}4)$)

```
fit %>% summary()
```

```
##
## Call:
## glm(formula = good ~ . - quality - drinkable, family = binomial(link = "logit"),
##     data = wine)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.9878  -0.4351  -0.2207  -0.1222   2.9869
##
## Coefficients:
##                       Estimate Std. Error z value Pr(>|z|)
## (Intercept)           2.428e+02  1.081e+02   2.247 0.024660 *
## fixed_acidity         2.750e-01  1.253e-01   2.195 0.028183 *
## volatile_acidity     -2.581e+00  7.843e-01  -3.291 0.000999 ***
## citric_acid           5.678e-01  8.385e-01   0.677 0.498313
## residual_sugar        2.395e-01  7.373e-02   3.248 0.001163 **
```

```
## chlorides            -8.816e+00  3.365e+00  -2.620 0.008788 **
## free_sulfur_dioxide   1.082e-02  1.223e-02   0.884 0.376469
## total_sulfur_dioxide -1.653e-02  4.894e-03  -3.378 0.000731 ***
## density              -2.578e+02  1.104e+02  -2.335 0.019536 *
## pH                    2.242e-01  9.984e-01   0.225 0.822327
## sulphates             3.750e+00  5.416e-01   6.924 4.39e-12 ***
## alcohol               7.533e-01  1.316e-01   5.724 1.04e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1269.92  on 1598  degrees of freedom
## Residual deviance:  870.86  on 1587  degrees of freedom
## AIC: 894.86
##
## Number of Fisher Scoring iterations: 6
```

5. Impact on outcome by varying total_sulfur_dioxide

By fixing all coefficients and varying one, we can see the impact of one particular coefficient has on the probabilistic outcome of the logistic regression.

As a comparison I have fixed all significant coefficients in turn (`total_sulfur_dioxide`, `volatile_acidity`, `sulphates` and `alcohol`)

First, we save coefficients to their own variable.

```
b0 <- fit$coefficients[1]  # Intercept = 242.76251933
b1 <- fit$coefficients[2]  # fixed_acidity = 0.27495289
b2 <- fit$coefficients[3]  # volatile_acidity = -2.58100211
b3 <- fit$coefficients[4]  # citric_acid = 0.56779433
b4 <- fit$coefficients[5]  # residual_sugar = 0.23946420
b5 <- fit$coefficients[6]  # chlorides = -8.81636544
b6 <- fit$coefficients[7]  # free_sulfur_dioxide = 0.01082060
b7 <- fit$coefficients[8]  # total_sulfur_dioxide = -0.01653061
b8 <- fit$coefficients[9]  # density = -257.79757874
b9 <- fit$coefficients[10]  # pH = 0.22418522
b10 <- fit$coefficients[11]  # sulphates = 3.74987886
b11 <- fit$coefficients[12]  # alcohol = 0.75333905
```

Then save each individual mean values

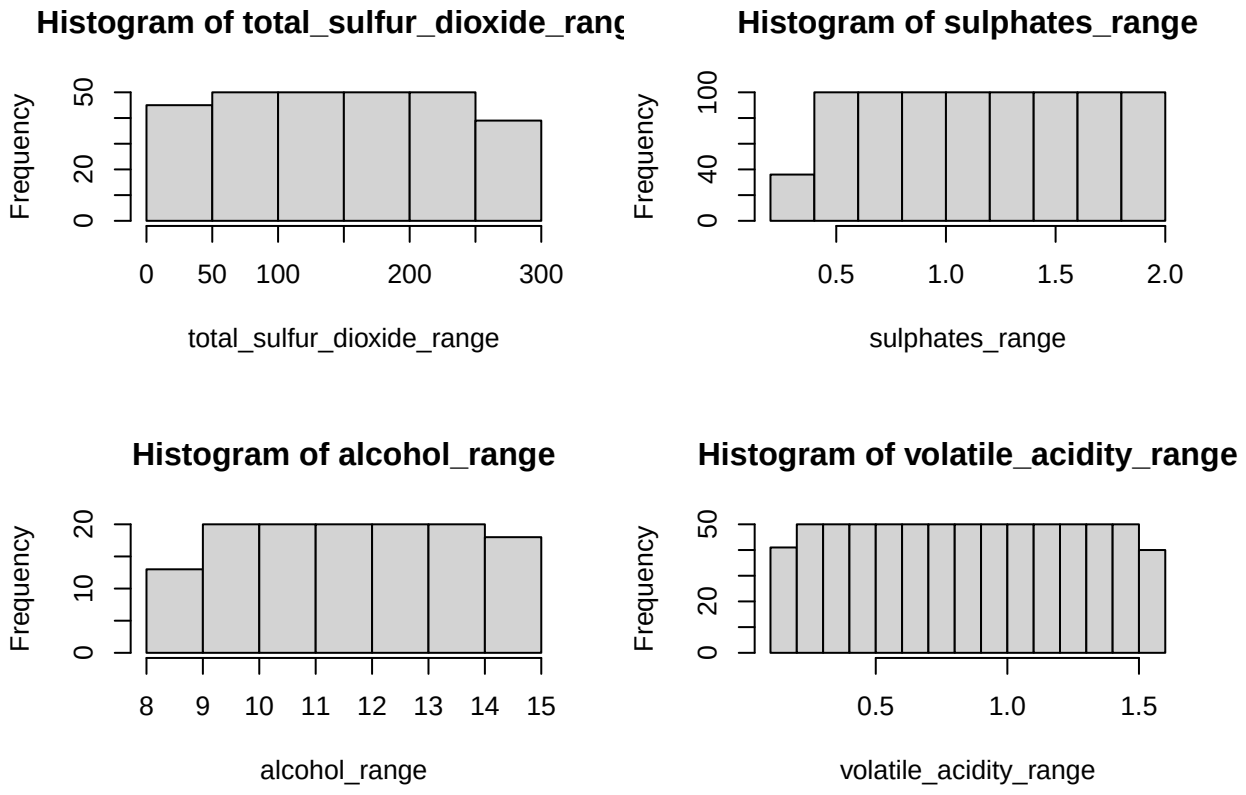
```
fixed_acidity_mean <- mean(wine$fixed_acidity)
volatile_acidity_mean <- mean(wine$volatile_acidity)
citric_acid_mean <- mean(wine$citric_acid)
residual_sugar_mean <- mean(wine$residual_sugar)
chlorides_mean <- mean(wine$chlorides)
free_sulfur_dioxide_mean <- mean(wine$free_sulfur_dioxide)
total_sulfur_dioxide_mean <- mean(wine$total_sulfur_dioxide)
density_mean <- mean(wine$density)
pH_mean <- mean(wine$pH)
sulphates_mean <- mean(wine$sulphates)
alcohol_mean <- mean(wine$alcohol)
```

Compute the range of each significant coefficient, `total_sulfur_dioxide`, `sulphates`, `alcohol`, and `volatile_acidity`

```
total_sulfur_dioxide_range <- seq(from=min(wine$total_sulfur_dioxide), to=max(wine$total_sulfur_dioxide
sulphates_range <- seq(from=min(wine$sulphates), to=max(wine$sulphates), by=0.002)
alcohol_range <- seq(from=min(wine$alcohol), to=max(wine$alcohol), by=0.05)
volatile_acidity_range <- seq(from=min(wine$volatile_acidity), to=max(wine$volatile_acidity), by=0.002)
```

Plot of the range histogram of each of the four most significant covariates.

```
par(mfrow=c(2,2))
hist(total_sulfur_dioxide_range)
hist(sulphates_range)
hist(alcohol_range)
hist(volatile_acidity_range)
```



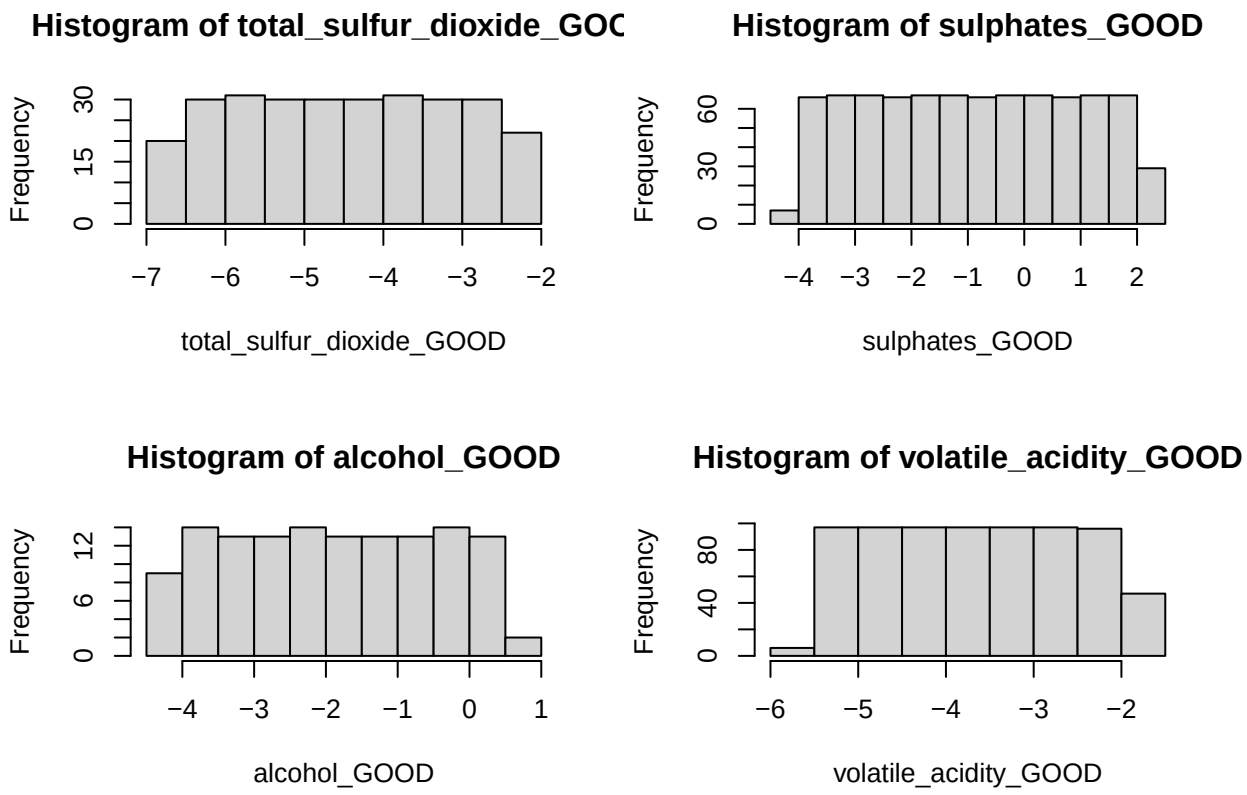
Calculate probabilities for each significant coefficient

```
total_sulfur_dioxide_GOOD <- b0 + b1*fixed_acidity_mean + b2*volatile_acidity_mean +
  b3*citric_acid_mean + b4*residual_sugar_mean + b5*chlorides_mean + b6*free_sulfur_dioxide_mean +
  b7*total_sulfur_dioxide_range + b8*density_mean + b9*pH_mean + b10*sulphates_mean + b11*alcohol_mean

sulphates_GOOD <- b0 + b1*fixed_acidity_mean + b2*volatile_acidity_mean +
  b3*citric_acid_mean + b4*residual_sugar_mean + b5*chlorides_mean + b6*free_sulfur_dioxide_mean +
  b7*total_sulfur_dioxide_mean + b8*density_mean + b9*pH_mean + b10*sulphates_range + b11*alcohol_mean
```

```
alcohol_GOOD <- b0 + b1*fixed_acidity_mean + b2*volatile_acidity_mean +
  b3*citric_acid_mean + b4*residual_sugar_mean + b5*chlorides_mean + b6*free_sulfur_dioxide_mean +
  b7*total_sulfur_dioxide_mean + b8*density_mean + b9*pH_mean + b10*sulphates_mean + b11*alcohol_range

volatile_acidity_GOOD <- b0 + b1*fixed_acidity_mean + b2*volatile_acidity_range +
  b3*citric_acid_mean + b4*residual_sugar_mean + b5*chlorides_mean + b6*free_sulfur_dioxide_mean +
  b7*total_sulfur_dioxide_mean + b8*density_mean + b9*pH_mean + b10*sulphates_mean + b11*alcohol_mean
```

Histograms of what????

```
par(mfrow=c(2,2))
hist(total_sulfur_dioxide_GOOD)
hist(sulphates_GOOD)
hist(alcohol_GOOD)
hist(volatile_acidity_GOOD)
```

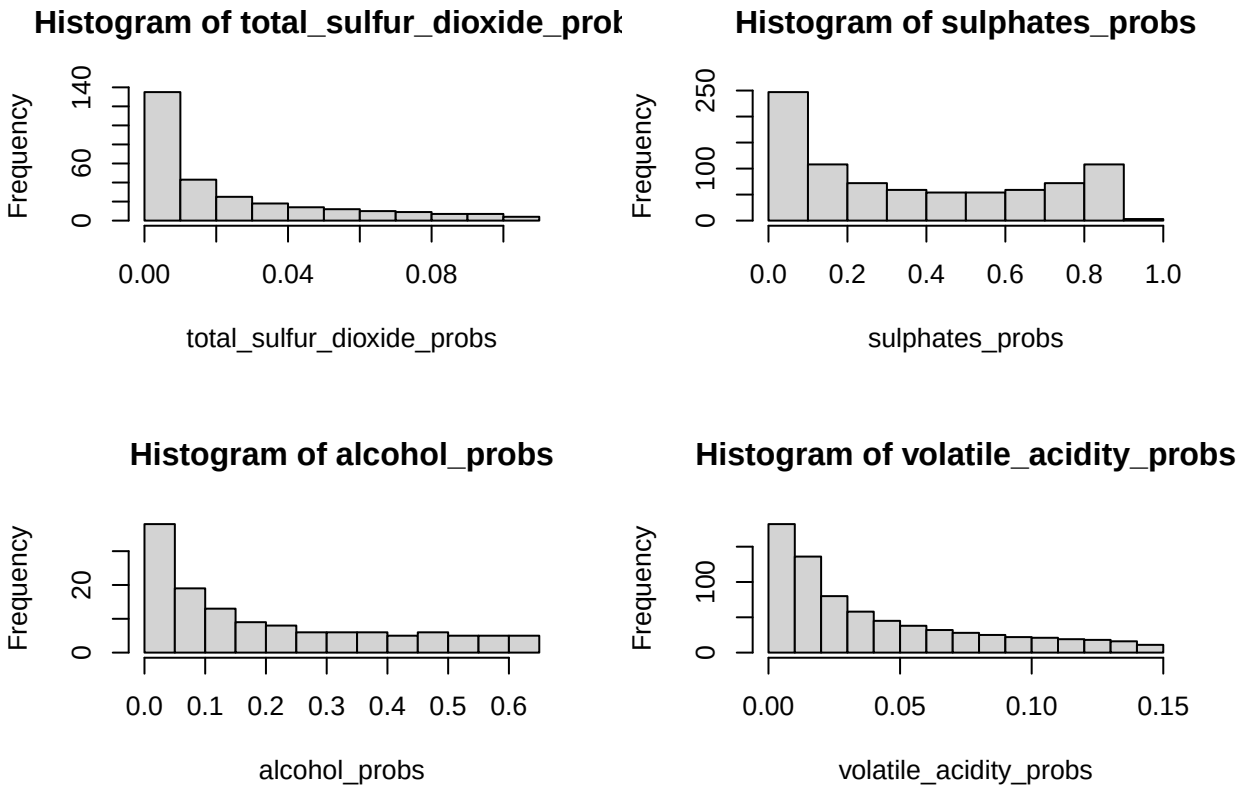


Calculate logistic probabilities for each

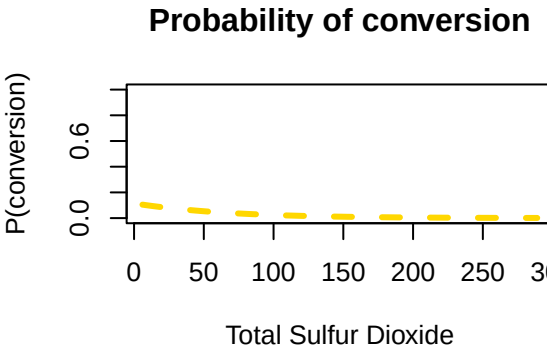
```
total_sulfur_dioxide_probs <- exp(total_sulfur_dioxide_GOOD)/(1 + exp(total_sulfur_dioxide_GOOD))
sulphates_probs <- exp(sulphates_GOOD) / (1 + exp(sulphates_GOOD))
alcohol_probs <- exp(alcohol_GOOD) / (1 + exp(alcohol_GOOD))
volatile_acidity_probs <- exp(volatile_acidity_GOOD) / (1 + exp(volatile_acidity_GOOD))
```

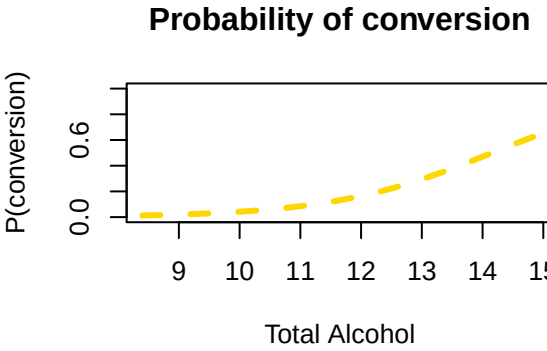
Plot logistic probabilities

```
par(mfrow=c(2,2))
hist(total_sulfur_dioxide_probs)
hist(sulphates_probs)
hist(alcohol_probs)
hist(volatile_acidity_probs)
```



Plot the results.





The plots of each of the four covariates are the probabilities across the range of

6. Bayesian Logistic Regression

The following is a Bayesian logistic regression analysis.

To perform a Bayesian analysis via Metropolis-Hastings algorithm we write a function defining the target distribution of the Beta coefficients. We work in terms of log posterior to avoid numerical problems with computer computation.

Likelihood.

The likelihood is a product of Bernoulli likelihoods for each combination of covariate and response variable for the i-th term.

$$L(\beta; y, x) = \prod_{i=1}^{n} \left(\frac{\exp(x_i\beta)}{1+\exp(x_i\beta)} \right)^{y_i} \left(1 - \frac{\exp(x_i\beta)}{1+\exp(x_i\beta)} \right)^{1-y_i}$$

Log-likelihood

$$\log L(\beta; y, x) = \sum_{i=1}^{n} y_i(x_i\beta) - \log[1+\exp(x_i\beta)]$$

Prior

Independent normal prior distributions for each $\beta_j, j = 0, 1, \ldots, k$, such that $\beta_j \sim N(0, 100)$

Log posterior distribution

```
lpost.LR <- function(beta, x, y) {
  # beta: vector of coefficients
  # x: covariates (regressors)
  # y: response variable

  # eta is the matrix product of the covariates and the coefficients
  eta <- as.numeric(x %*% beta)
  # logp: y=1, in terms of logistic function
  logp <- eta - log(1 + exp(eta))  # in log scale
  # logq: y=0, expressed in terms of logistic function
  logq <- log(1 - exp(logp))
  # sum of both contributions: when y=1 + when y=0
  logl <- sum(logp[y==1]) + sum(logq[y==0])
  # log prior: normal indep prior dists $N(0, 100)$
  lprior <- sum(dnorm(beta, 0, 10,log = T))
  # return log posterior (likelihood + prior)
  return(logl + lprior)
}
```

The MLE for each coefficient can be used to initialise the simulation matrix

```
fit <- glm(
  #good ~ volatile_acidity + total_sulfur_dioxide + sulphates + alcohol,  # test model
  good ~ . - quality - drinkable,  #  full model
  data = wine,
  family = binomial(link="logit")
)
```

Set seed for reproducible results

```
set.seed(1234)
```

Number of simulations

```
S <- 10^4
```

Set `X` and `y` variables

```
# X=cbind(rep(1,nrow(wine)), wine$volatile_acidity, wine$total_sulfur_dioxide,
#          wine$sulphates, wine$alcohol)

X = cbind(rep(1, nrow(wine)), wine$fixed_acidity, wine$volatile_acidity, wine$citric_acid,
          wine$residual_sugar, wine$chlorides, wine$free_sulfur_dioxide, wine$total_sulfur_dioxide,
          wine$density, wine$pH, wine$sulphates, wine$alcohol)

y <- wine$good[1]
```

Initialisations for the coefficients

The first initialisation contains the MLE estimates for each coefficient. The remaining three initialisations are randomly selected values between 500 and 1000 from the uniform distribution $runif(48, 500, 1000)$.

```
# init <- matrix(data=c(runif(20, min = 100, max = 200)), nrow=4, ncol=5, byrow = T)  # for testing
init <- matrix(data=c(runif(48, min = 500, max = 1000)), nrow=4, ncol=12, byrow = T)
```

Run a Metropolis-Hastings algorithm

```
# First initialisation
beta_mat_init1 <- matrix(NA, nrow = S, ncol = ncol(X))
k <- ncol(beta_mat_init1)
#beta_mat_init1[1,] <- init[1,]
beta_mat_init1[1,] <- as.numeric(coefficients(fit))  # initialise with MLE of each coefficient
acc <- 0
for (iter in 2:S) {
  # simulate all (k) values using previous value of beta as mean and set sd
  beta_star <- rnorm(k, beta_mat_init1[iter-1,], 5)
  # compute target distribution for proposed value
  newpost = lpost.LR(beta_star, X, y)
  # compute target distribution for old value
  oldpost = lpost.LR(beta_mat_init1[iter-1,], X, y)  # symmetric dist => no ratio computed

  # acceptance step, in log scale
  if (runif(1,0,1) > exp(newpost - oldpost)) {
    # chain doesn't move
    beta_mat_init1[iter,] = beta_mat_init1[iter-1,]
  } else {
    # add to chain and add 1 to counter
    beta_mat_init1[iter,] = beta_star
    acc=acc + 1
  }
  #if (iter%%1000 == 0) {print(c(iter, acc/iter))}
}
print(c(iter, acc/iter))
```

```
## [1] 1.000e+04 3.046e-01
```

```
# Second initialisation
beta_mat_init2 <- matrix(NA, nrow = S, ncol = ncol(X))
k <- ncol(beta_mat_init2)
beta_mat_init2[1,] <- init[2,]
acc <- 0
for(iter in 2:S){
  beta_star <- rnorm(k, beta_mat_init2[iter-1,], 5)
  newpost = lpost.LR(beta_star, X, y)
  oldpost = lpost.LR(beta_mat_init2[iter-1,], X, y)
  if (runif(1,0,1) > exp(newpost - oldpost)) {
    beta_mat_init2[iter,] = beta_mat_init2[iter-1,]
  } else {
```

```
    beta_mat_init2[iter,] = beta_star
    acc=acc + 1
  }
  #if (iter%%1000==0) {print(c(iter,acc/iter))}
}
print(c(iter, acc/iter))
```

```
## [1] 1.000e+04 2.045e-01
```

```
# Third initialisation
beta_mat_init3 <- matrix(NA,nrow=S,ncol=ncol(X))
k <- ncol(beta_mat_init3)
beta_mat_init3[1,] <- init[3,]
acc <- 0
for(iter in 2:S){
  beta_star <- rnorm(k,beta_mat_init3[iter-1,], 5)
  newpost=lpost.LR(beta_star,X,y)
  oldpost=lpost.LR(beta_mat_init3[iter-1,],X,y)
  if(runif(1,0,1)>exp(newpost-oldpost)){
    beta_mat_init3[iter,]=beta_mat_init3[iter-1,]
  } else{
    beta_mat_init3[iter,]=beta_star
    acc=acc+1
  }
  #if(iter%%1000==0){print(c(iter,acc/iter))}
}
print(c(iter, acc/iter))
```

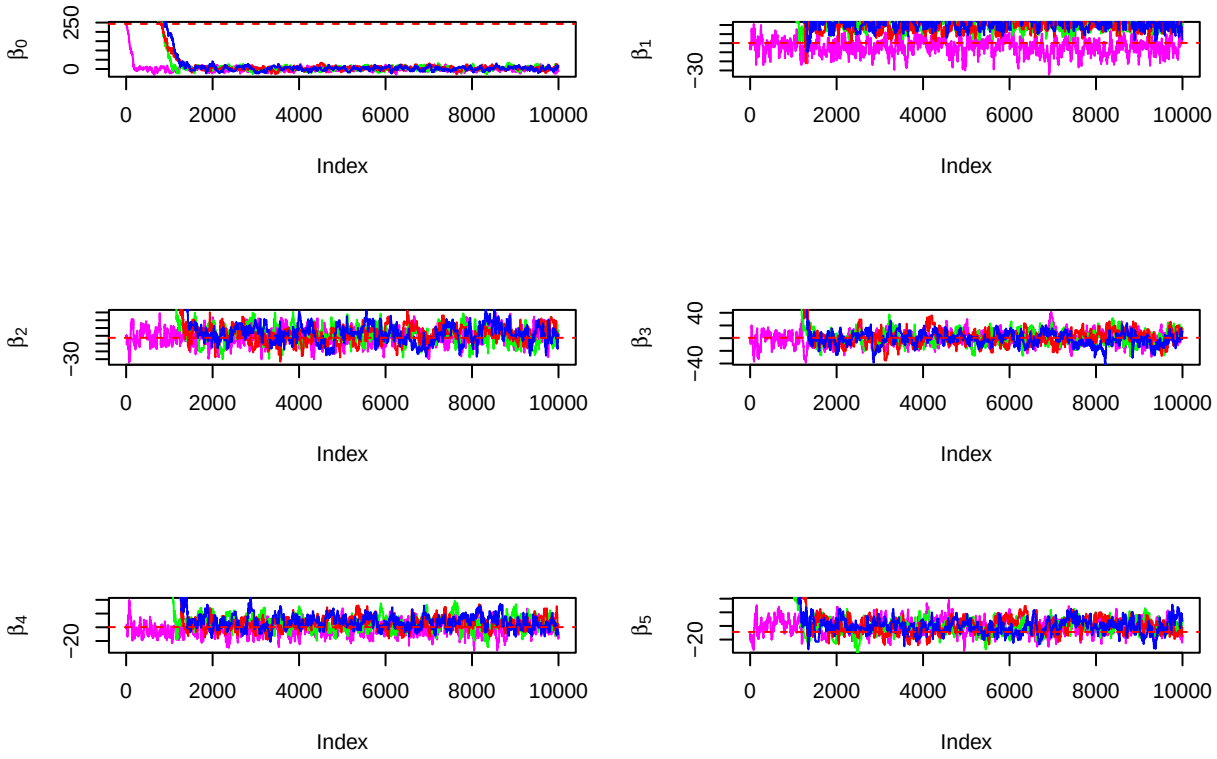
```
## [1] 1.000e+04 2.057e-01
```

```
# Forth initialisation
beta_mat_init4 <- matrix(NA,nrow=S,ncol=ncol(X))
k <- ncol(beta_mat_init4)
beta_mat_init4[1,] <- init[4,]
acc <- 0
for(iter in 2:S){
  beta_star <- rnorm(k,beta_mat_init4[iter-1,], 5)
  newpost=lpost.LR(beta_star,X,y)
  oldpost=lpost.LR(beta_mat_init4[iter-1,],X,y)
  if(runif(1,0,1)>exp(newpost-oldpost)){
    beta_mat_init4[iter,]=beta_mat_init4[iter-1,]
  } else{
    beta_mat_init4[iter,]=beta_star
    acc=acc+1
  }
  #if(iter%%1000==0){print(c(iter,acc/iter))}
}
print(c(iter, acc/iter))
```

```
## [1] 1.00e+04 2.14e-01
```

Plot the chains for each coefficient (same plot)

```
par(mfrow=c(3,2))
plot(beta_mat_init1[,1], type="l", col="magenta", ylab=expression(beta[0]))
lines(beta_mat_init2[,1], type="l", col="green")
lines(beta_mat_init3[,1], type="l", col="red")
lines(beta_mat_init4[,1], type="l", col="blue")
abline(h=fit$coefficients[1],col="red",lty=2)

plot(beta_mat_init1[,2], type="l", col="magenta", ylab=expression(beta[1]))
lines(beta_mat_init2[,2], type="l", col="green")
lines(beta_mat_init3[,2], type="l", col="red")
lines(beta_mat_init4[,2], type="l", col="blue")
abline(h=fit$coefficients[2],col="red",lty=2)

plot(beta_mat_init1[,3], type="l", col="magenta", ylab=expression(beta[2]))
lines(beta_mat_init2[,3], type="l", col="green")
lines(beta_mat_init3[,3], type="l", col="red")
lines(beta_mat_init4[,3], type="l", col="blue")
abline(h=fit$coefficients[3],col="red",lty=2)

plot(beta_mat_init1[,4], type="l", col="magenta", ylab=expression(beta[3]))
lines(beta_mat_init2[,4], type="l", col="green")
lines(beta_mat_init3[,4], type="l", col="red")
lines(beta_mat_init4[,4], type="l", col="blue")
abline(h=fit$coefficients[4],col="red",lty=2)

plot(beta_mat_init1[,5], type="l", col="magenta", ylab=expression(beta[4]))
lines(beta_mat_init2[,5], type="l", col="green")
lines(beta_mat_init3[,5], type="l", col="red")
lines(beta_mat_init4[,5], type="l", col="blue")
abline(h=fit$coefficients[5],col="red",lty=2)

plot(beta_mat_init1[,6], type="l", col="magenta", ylab=expression(beta[5]))
lines(beta_mat_init2[,6], type="l", col="green")
lines(beta_mat_init3[,6], type="l", col="red")
lines(beta_mat_init4[,6], type="l", col="blue")
abline(h=fit$coefficients[6],col="red",lty=2)
```



Comment

```
fit$coefficients[1:6]
```

```
##      (Intercept)    fixed_acidity volatile_acidity      citric_acid
##      242.7625193        0.2749529       -2.5810021        0.5677943
##   residual_sugar        chlorides
##        0.2394642       -8.8163654
```

β_0 (`Intercept`: 242.7625): No convergence with MLE. All four chains do converge to a value around zero.

β_1 (`fixed_acidity`: 0.275): First chain containing MLE converges to a value close but slightly under the coefficient. The other three chains containing random initialisations converge to a value approximately 20-30.

β_2 (`volative_acidity`: -2.581): All four chains converge to a value close to the coefficient.

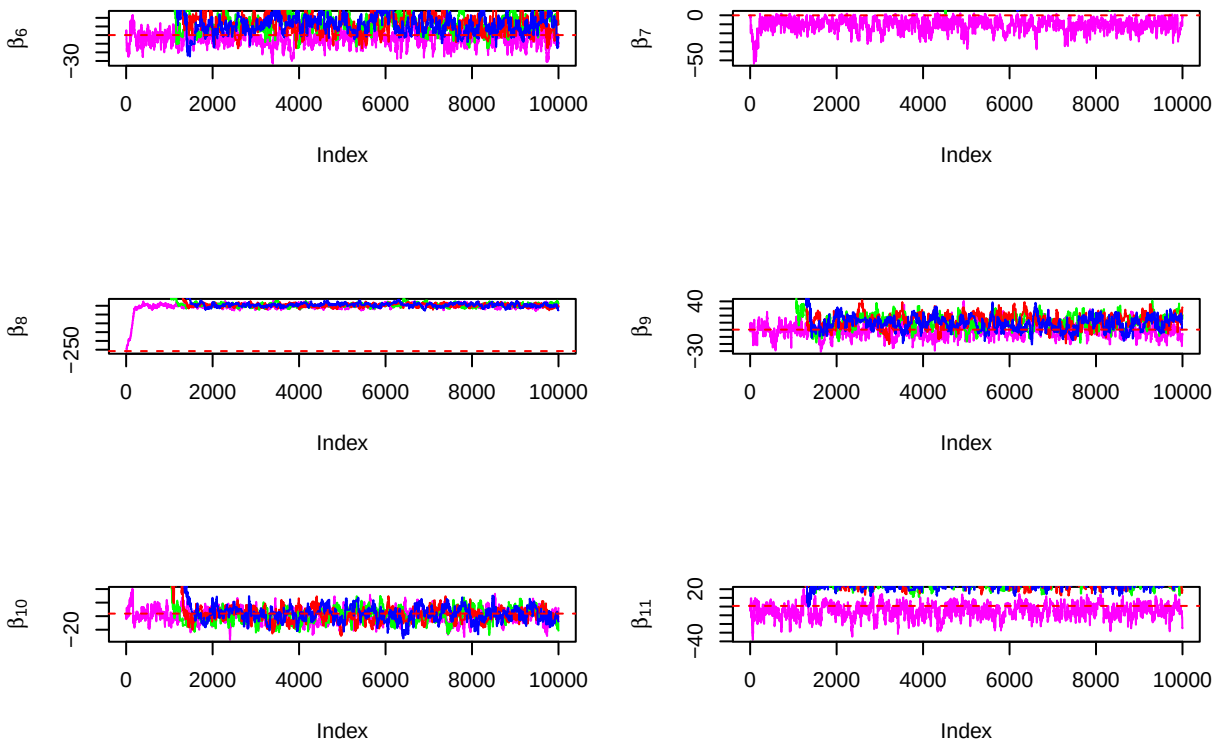
β_3 (`citric_acid`: 0.568): All four chains converge to a value close to the coefficient.

β_4 (`residual_sugar`: 0.239): All four chains converge to a value close to the coefficient.

β_5 (`chlorides`: -8.816): All four chains converge to a value close to the coefficient.

```
par(mfrow=c(3,2))

plot(beta_mat_init1[,7], type="l", col="magenta", ylab=expression(beta[6]))
lines(beta_mat_init2[,7], type="l", col="green")
lines(beta_mat_init3[,7], type="l", col="red")
lines(beta_mat_init4[,7], type="l", col="blue")
```

```
abline(h=fit$coefficients[7],col="red",lty=2)

plot(beta_mat_init1[,8], type="l", col="magenta", ylab=expression(beta[7]))
lines(beta_mat_init2[,8], type="l", col="green")
lines(beta_mat_init3[,8], type="l", col="red")
lines(beta_mat_init4[,8], type="l", col="blue")
abline(h=fit$coefficients[8],col="red",lty=2)

plot(beta_mat_init1[,9], type="l", col="magenta", ylab=expression(beta[8]))
lines(beta_mat_init2[,9], type="l", col="green")
lines(beta_mat_init3[,9], type="l", col="red")
lines(beta_mat_init4[,9], type="l", col="blue")
abline(h=fit$coefficients[9],col="red",lty=2)

plot(beta_mat_init1[,10], type="l", col="magenta", ylab=expression(beta[9]))
lines(beta_mat_init2[,10], type="l", col="green")
lines(beta_mat_init3[,10], type="l", col="red")
lines(beta_mat_init4[,10], type="l", col="blue")
abline(h=fit$coefficients[10],col="red",lty=2)

plot(beta_mat_init1[,11], type="l", col="magenta", ylab=expression(beta[10]))
lines(beta_mat_init2[,11], type="l", col="green")
lines(beta_mat_init3[,11], type="l", col="red")
lines(beta_mat_init4[,11], type="l", col="blue")
abline(h=fit$coefficients[11],col="red",lty=2)

plot(beta_mat_init1[,12], type="l", col="magenta", ylab=expression(beta[11]))
lines(beta_mat_init2[,12], type="l", col="green")
lines(beta_mat_init3[,12], type="l", col="red")
lines(beta_mat_init4[,12], type="l", col="blue")
abline(h=fit$coefficients[12],col="red",lty=2)
```



Comment

```
fit$coefficients[7:12]
```

```
##  free_sulfur_dioxide total_sulfur_dioxide             density
##           0.01082060          -0.01653061        -257.79757874
##                   pH            sulphates             alcohol
##           0.22418522            3.74987886           0.75333905
```

β_6 (`free_sulfur_dioxide`: 0.011): First chain containing MLE converges to a value close but slightly under the coefficient. The other three chains containing random initialisations converge to a value approximately 20-30.

β_7 (`total_sulfur_dioxide`: -0.017): First chain converges to a value just below coefficient but chains 2-4 don't converge to coefficient.

β_8 (`density`: -257.798): First chain converges to a value at about the coefficient but chains 2-4 converge to a value slightly above the coefficient

β_9 (`pH`: 0.224): All four chains converge to a value close to the coefficient.

β_{10} (`sulphates`: 3.750): All four chains converge to a value close to the coefficient.

β_{11} (`alcohol`: 0.753): First chain converges to a value at about the coefficient but chains 2-4 converge to a value above the coefficient

7. Posterior Predictive Distribution

Approximate the posterior predictive distribution of an unobserved variable characterised by the following values for each covariate:

```
fixed_acidity <- 7.5
volatile_acidity <- 0.6
citric_acid <- 0.0
residual_sugar <- 1.7
chlorides <- 0.085
free_sulfur_dioxide <- 5
total_sulfur_dioxide <- 45
density <- 0.9965
pH <- 3.4
sulphates <- 0.63
alcohol <- 12
```

```
S <- 20000
beta_mat2 <- matrix(NA, nrow = S, ncol = ncol(X))
beta_mat2[1,] <- as.numeric(coefficients(fit))

y_new <- c(1)
# x_new <- c(1, 5, 0.63, 12, 0.6)  # for testing
x_new <- c(1, fixed_acidity, volatile_acidity, citric_acid, residual_sugar,
           chlorides, free_sulfur_dioxide, total_sulfur_dioxide, density,
           pH, sulphates, alcohol)
```

new model

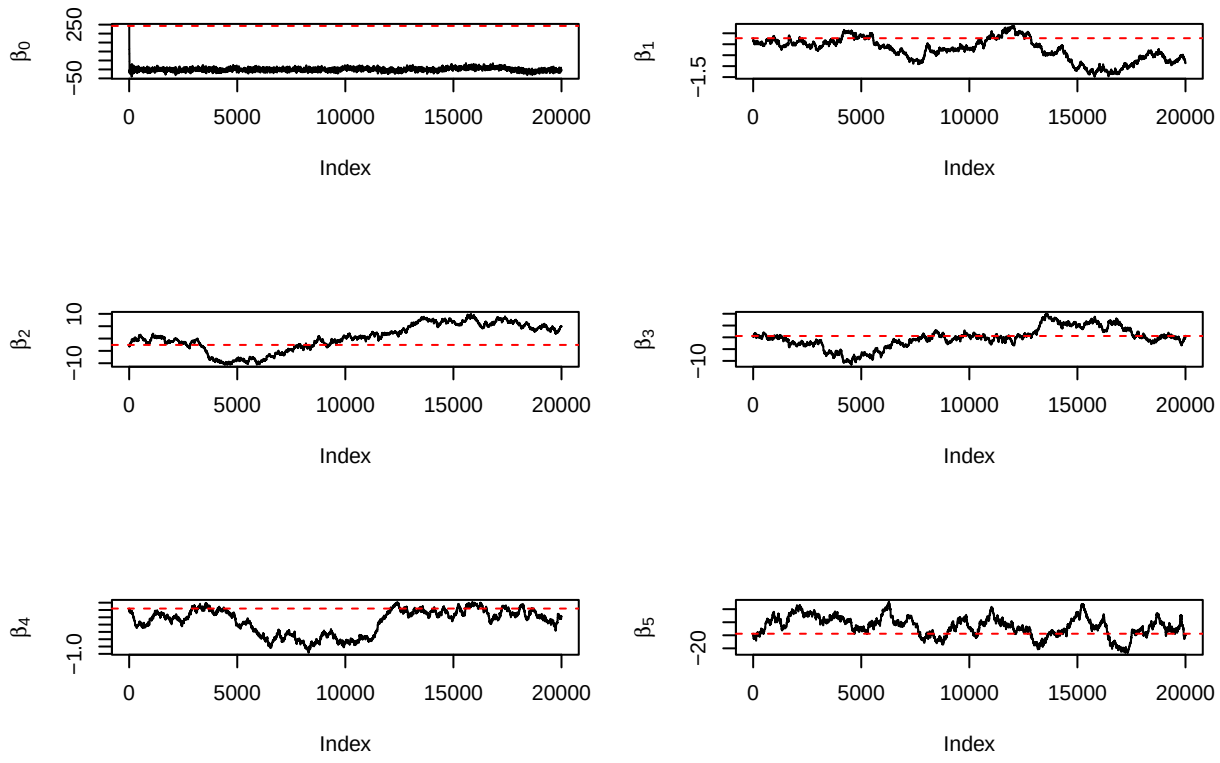
```
library(mvtnorm)

# prediction

Omega_prop <- solve(t(X) %*% X)
k <- ncol(beta_mat2)
acc <- 0
for(iter in 2:S)
{
  # 1. Propose a new set of values
  beta_star <- rmvnorm(1, beta_mat2[iter-1,], 1.5 * Omega_prop)

  # 2. Compute the posterior density on the proposed value and on the old value
  newpost=lpost.LR(t(beta_star), X, y)
  oldpost=lpost.LR(matrix(beta_mat2[iter-1,], ncol=1), X, y)

  # 3. Acceptance step
  if (runif(1, 0, 1) > exp(newpost - oldpost)) {
    beta_mat2[iter,] = beta_mat2[iter-1,]
  } else {
    beta_mat2[iter,] = beta_star
    acc = acc + 1
  }
  # 4. Print the stage of the chain
```

```
  if (iter%%1000 == 0){ print(c(iter, acc/iter)) }

  # 5. Prediction
  p_new <- exp(sum(beta_mat2[iter,] * x_new) ) / (1 + exp(sum(beta_mat2[iter,] * x_new) ))
  y_new[iter] <- rbinom(1,1,prob=p_new)
}
```

```
## [1] 1000.000    0.197
## [1] 2000.0000    0.1945
## [1] 3000.0000000    0.1936667
## [1] 4000.000    0.201
## [1] 5000.0000    0.2026
## [1] 6000.0000    0.2035
## [1] 7000.0000000    0.2041429
## [1] 8000.00000    0.20125
## [1] 9000.0000000    0.2042222
## [1] 1.000e+04 2.048e-01
## [1] 1.100000e+04 2.048182e-01
## [1] 1.200000e+04 2.051667e-01
## [1] 1.300000e+04 2.047692e-01
## [1] 1.40e+04 2.04e-01
## [1] 1.500e+04 2.044e-01
## [1] 1.6000e+04 2.0525e-01
## [1] 1.700000e+04 2.058235e-01
## [1] 1.800000e+04 2.055556e-01
## [1] 1.900000e+04 2.061579e-01
## [1] 2.0000e+04 2.0725e-01
```

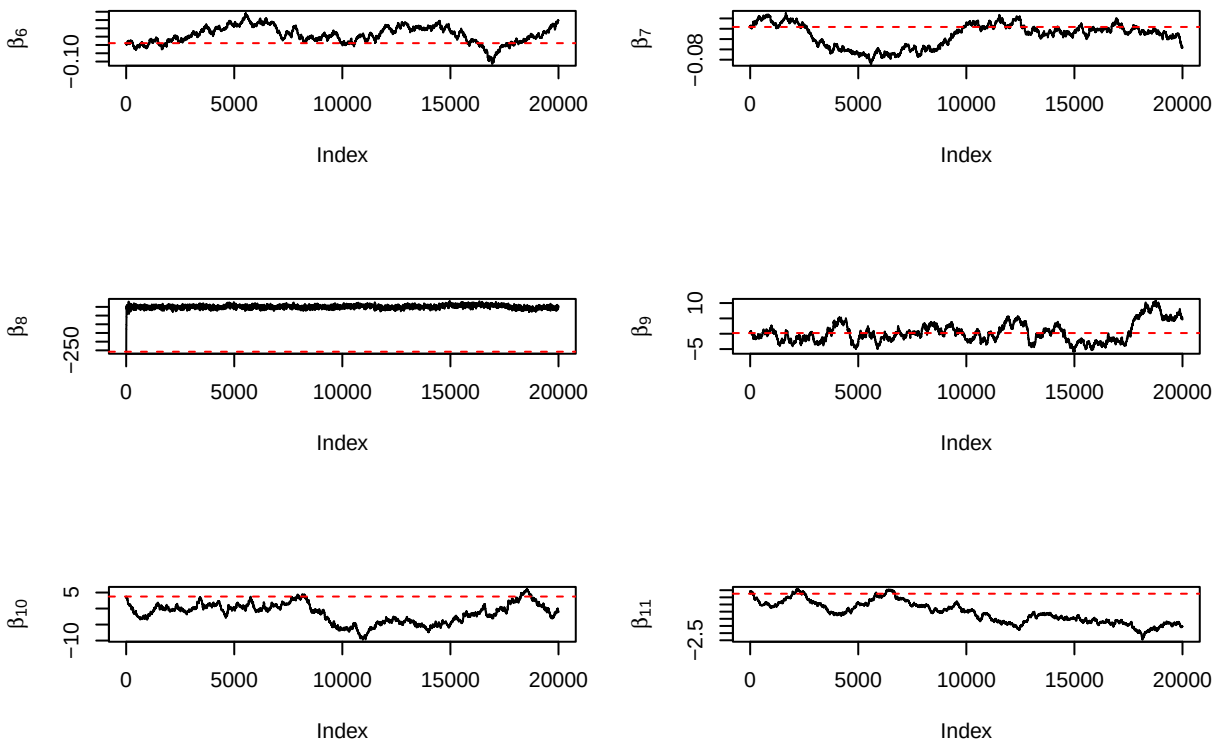
Plots

```
par(mfrow=c(3,2))
plot(beta_mat2[,1],type="l", ylab=expression(beta[0]))
abline(h=fit$coefficients[1],col="red",lty=2)
plot(beta_mat2[,2],type="l", ylab=expression(beta[1]))
abline(h=fit$coefficients[2],col="red",lty=2)
plot(beta_mat2[,3],type="l", ylab=expression(beta[2]))
abline(h=fit$coefficients[3],col="red",lty=2)
plot(beta_mat2[,4],type="l", ylab=expression(beta[3]))
abline(h=fit$coefficients[4],col="red",lty=2)
plot(beta_mat2[,5],type="l", ylab=expression(beta[4]))
abline(h=fit$coefficients[5],col="red",lty=2)
plot(beta_mat2[,6],type="l", ylab=expression(beta[5]))
abline(h=fit$coefficients[6],col="red",lty=2)
```



Plots

```
par(mfrow=c(3,2))
plot(beta_mat2[,7],type="l", ylab=expression(beta[6]))
abline(h=fit$coefficients[7],col="red",lty=2)
plot(beta_mat2[,8],type="l", ylab=expression(beta[7]))
abline(h=fit$coefficients[8],col="red",lty=2)
plot(beta_mat2[,9],type="l", ylab=expression(beta[8]))
abline(h=fit$coefficients[9],col="red",lty=2)
plot(beta_mat2[,10],type="l", ylab=expression(beta[9]))
abline(h=fit$coefficients[10],col="red",lty=2)
plot(beta_mat2[,11],type="l", ylab=expression(beta[10]))
abline(h=fit$coefficients[11],col="red",lty=2)
plot(beta_mat2[,12],type="l", ylab=expression(beta[11]))
abline(h=fit$coefficients[12],col="red",lty=2)
```




```
table(y_new[15000:20000])
```

```
##
##    0
## 5001
```

8. metrop() analysis of Q6

Implementation of the `metrop()` function was through the MCMC Package Example, Charles J. Geyer

```
library(mcmc)
# out <- glm(wine$quality ~ wine$volatile_acidity + wine$total_sulfur_dioxide + wine$sulphates + wine$a
# summary(out)

# full model
out <- glm(good ~ . - quality - drinkable, data=wine, family=binomial, x=TRUE)
```

```
lupost_factory <- function(x, y) function(beta) {
  eta <- as.numeric(x %*% beta)
  logp <- ifelse(eta < 0, eta - log1p(exp(eta)), - log1p(exp(- eta)))
  logq <- ifelse(eta < 0, - log1p(exp(eta)), - eta - log1p(exp(- eta)))
  logl <- sum(logp[y == 1]) + sum(logq[y == 0])
  return(logl - sum(beta^2) / 8)
```

```
}
lupost <- lupost_factory(out$x, out$y)
```

```
set.seed(1234)
beta.init <- as.numeric(coefficients(out))
out <- metrop(lupost, beta.init, 1e3)

names(out)
```

```
##  [1] "accept"       "batch"        "initial"      "final"        "accept.batch"
##  [6] "initial.seed" "final.seed"   "time"         "lud"          "nbatch"
## [11] "blen"         "nspac"        "scale"        "debug"
```

Look at the acceptance rate.

```
out$accept
```

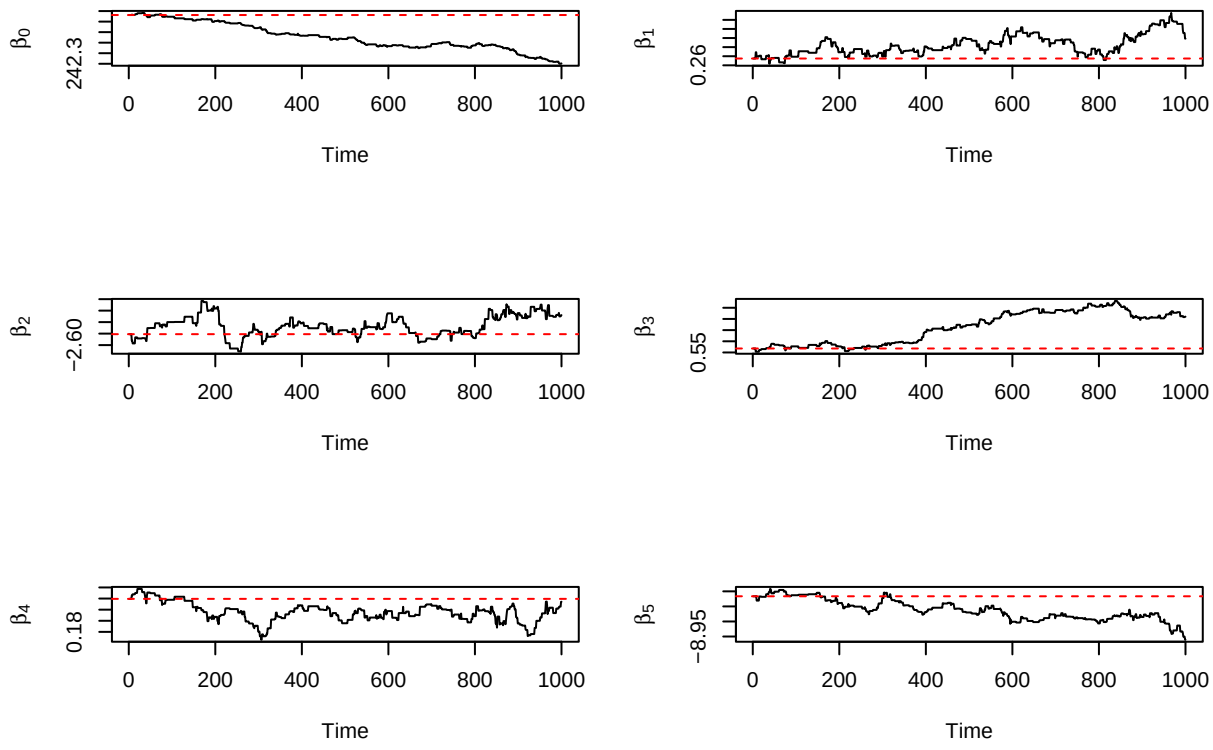
```
## [1] 0
```

This is very low, so we can adjust the scale parameter to find an acceptance rate around 20%.

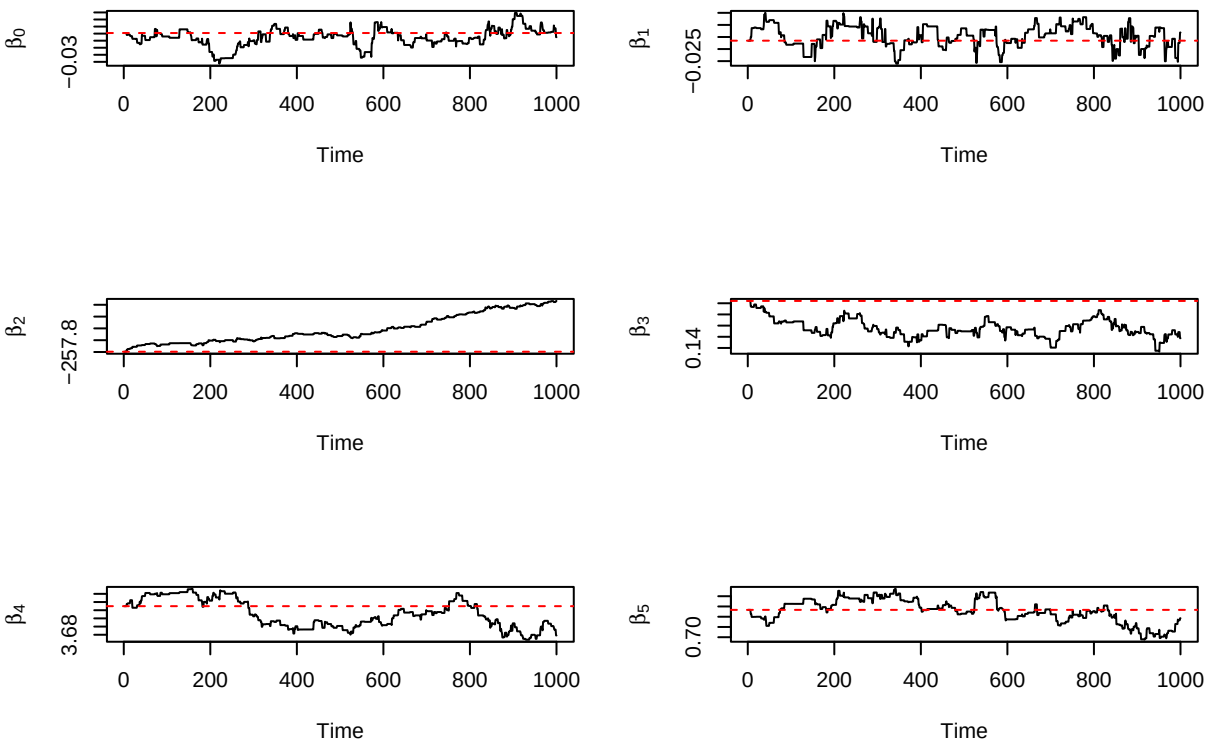
```
out <- metrop(out, scale = 0.0075)
out$accept
```

```
## [1] 0.213
```

```
par(mfrow=c(3,2))
plot(ts(out$batch)[,1], ylab=expression(beta[0]))
abline(h=fit$coefficients[1],col="red",lty=2)
plot(ts(out$batch)[,2], ylab=expression(beta[1]))
abline(h=fit$coefficients[2],col="red",lty=2)
plot(ts(out$batch)[,3], ylab=expression(beta[2]))
abline(h=fit$coefficients[3],col="red",lty=2)
plot(ts(out$batch)[,4], ylab=expression(beta[3]))
abline(h=fit$coefficients[4],col="red",lty=2)
plot(ts(out$batch)[,5], ylab=expression(beta[4]))
abline(h=fit$coefficients[5],col="red",lty=2)
plot(ts(out$batch)[,6], ylab=expression(beta[5]))
abline(h=fit$coefficients[6],col="red",lty=2)
```




```
par(mfrow=c(3,2))
plot(ts(out$batch)[,7], ylab=expression(beta[0]))
abline(h=fit$coefficients[7],col="red",lty=2)
plot(ts(out$batch)[,8], ylab=expression(beta[1]))
abline(h=fit$coefficients[8],col="red",lty=2)
plot(ts(out$batch)[,9], ylab=expression(beta[2]))
abline(h=fit$coefficients[9],col="red",lty=2)
plot(ts(out$batch)[,10], ylab=expression(beta[3]))
abline(h=fit$coefficients[10],col="red",lty=2)
plot(ts(out$batch)[,11], ylab=expression(beta[4]))
abline(h=fit$coefficients[11],col="red",lty=2)
plot(ts(out$batch)[,12], ylab=expression(beta[5]))
abline(h=fit$coefficients[12],col="red",lty=2)
```



VISUAL COMPARISON