Bayesian Inference and Computation for Data Scientists

Assessment 3: Bayesian analysis project: Vino Verde

Rushmila Islam

2024-04-22

Contents

1 Introduction

For this assessment we perform Logistic regression on a true dataset which is related to red variants of the Portuguese "Vinho Verde" wine. 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.*

The input variables (based on physicochemical tests) are:

- fixed acidity
- volatile acidity
- citric acid
- residual sugar
- chlorides
- free sulfur dioxide
- total sulfur dioxide
- density
- pH
- sulphates
- alcohol

The output variable (based on sensory data) is **quality** (a score between 0 and 10).

2 Analysis

2.1 Reading dataset into R

We install necessary libraries for the analysis:

```
library(mcmc)
library(MCMCpack)
```

```
## Loading required package: coda
```

```
## Loading required package: MASS
```

```
## ##
## ## Markov Chain Monte Carlo Package (MCMCpack)
```

```
## ## Copyright (C) 2003-2024 Andrew D. Martin, Kevin M. Quinn, and Jong Hee Park
```

```
## ##
## ## Support provided by the U.S. National Science Foundation
```

```
## ## (Grants SES-0350646 and SES-0350613)
## ##
```

```
library(MASS)
library(mvtnorm)
```

We set the seed value as 1234:

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

At the beginning of the analysis we read the dataset into R.

```
Wine <- read.csv("~/Library/CloudStorage/OneDrive-Personal/UNSW/2024-02-ZZSC5960-Bayesian Inference and
                 header = TRUE, sep = ",")
```

Below is the summary statistics of the dataset:

```
summary(Wine)
```

```
##  fixed.acidity   volatile.acidity  citric.acid    residual.sugar
##  Min.   : 4.60   Min.   :0.1200   Min.   :0.000   Min.   : 0.900
##  1st Qu.: 7.10   1st Qu.:0.3900   1st Qu.:0.090   1st Qu.: 1.900
##  Median : 7.90   Median :0.5200   Median :0.260   Median : 2.200
##  Mean   : 8.32   Mean   :0.5278   Mean   :0.271   Mean   : 2.539
##  3rd Qu.: 9.20   3rd Qu.:0.6400   3rd Qu.:0.420   3rd Qu.: 2.600
##  Max.   :15.90   Max.   :1.5800   Max.   :1.000   Max.   :15.500
##    chlorides       free.sulfur.dioxide total.sulfur.dioxide    density
##  Min.   :0.01200   Min.   : 1.00       Min.   :  6.00       Min.   :0.9901
##  1st Qu.:0.07000   1st Qu.: 7.00       1st Qu.: 22.00       1st Qu.:0.9956
##  Median :0.07900   Median :14.00       Median : 38.00       Median :0.9968
##  Mean   :0.08747   Mean   :15.87       Mean   : 46.47       Mean   :0.9967
##  3rd Qu.:0.09000   3rd Qu.:21.00       3rd Qu.: 62.00       3rd Qu.:0.9978
##  Max.   :0.61100   Max.   :72.00       Max.   :289.00       Max.   :1.0037
##        pH          sulphates         alcohol         quality
##  Min.   :2.740   Min.   :0.3300   Min.   : 8.40   Min.   :3.000
##  1st Qu.:3.210   1st Qu.:0.5500   1st Qu.: 9.50   1st Qu.:5.000
##  Median :3.310   Median :0.6200   Median :10.20   Median :6.000
##  Mean   :3.311   Mean   :0.6581   Mean   :10.42   Mean   :5.636
##  3rd Qu.:3.400   3rd Qu.:0.7300   3rd Qu.:11.10   3rd Qu.:6.000
##  Max.   :4.010   Max.   :2.0000   Max.   :14.90   Max.   :8.000
```

Now we check if there are missing values (NA) and, in case there are, we need to remove them.

```
# Find missing values
print("Position of missing values ")
```

```
## [1] "Position of missing values "
```

```
which(is.na(Wine))
```

```
## integer(0)
```

```
# count total missing values
print("Count of total missing values  ")
```

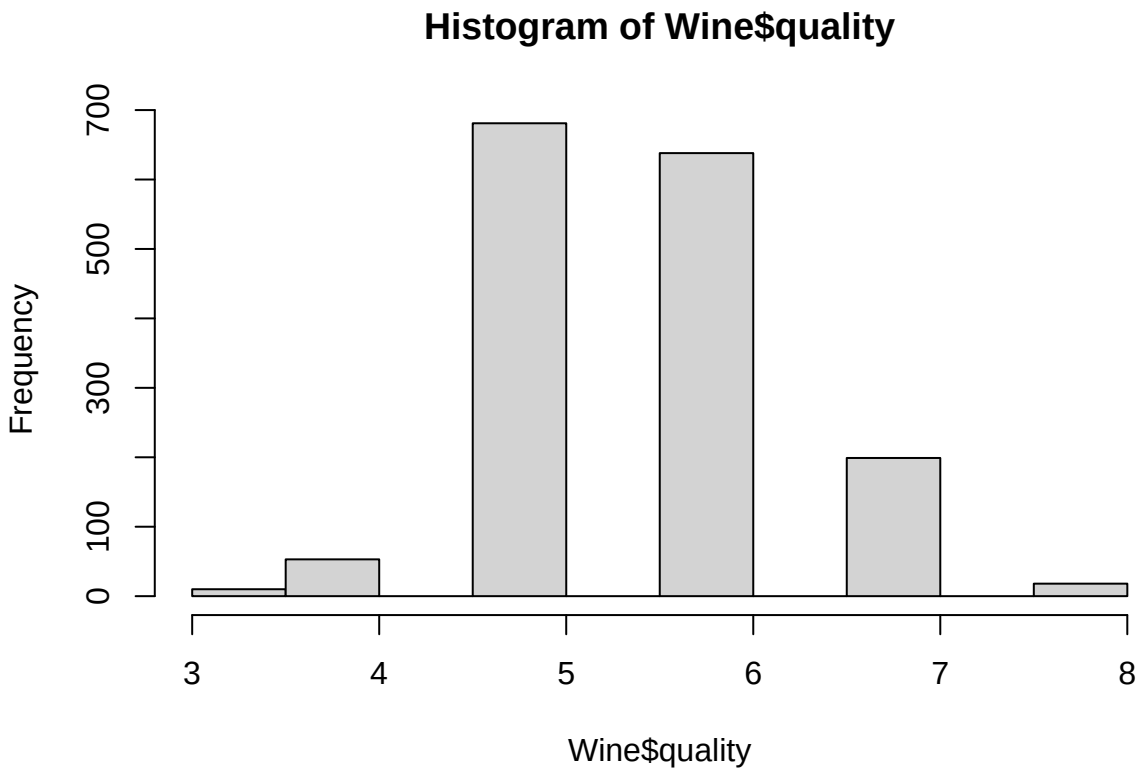
```
## [1] "Count of total missing values  "
```

```
sum(is.na(Wine))
```

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

We can see there are no missing values.

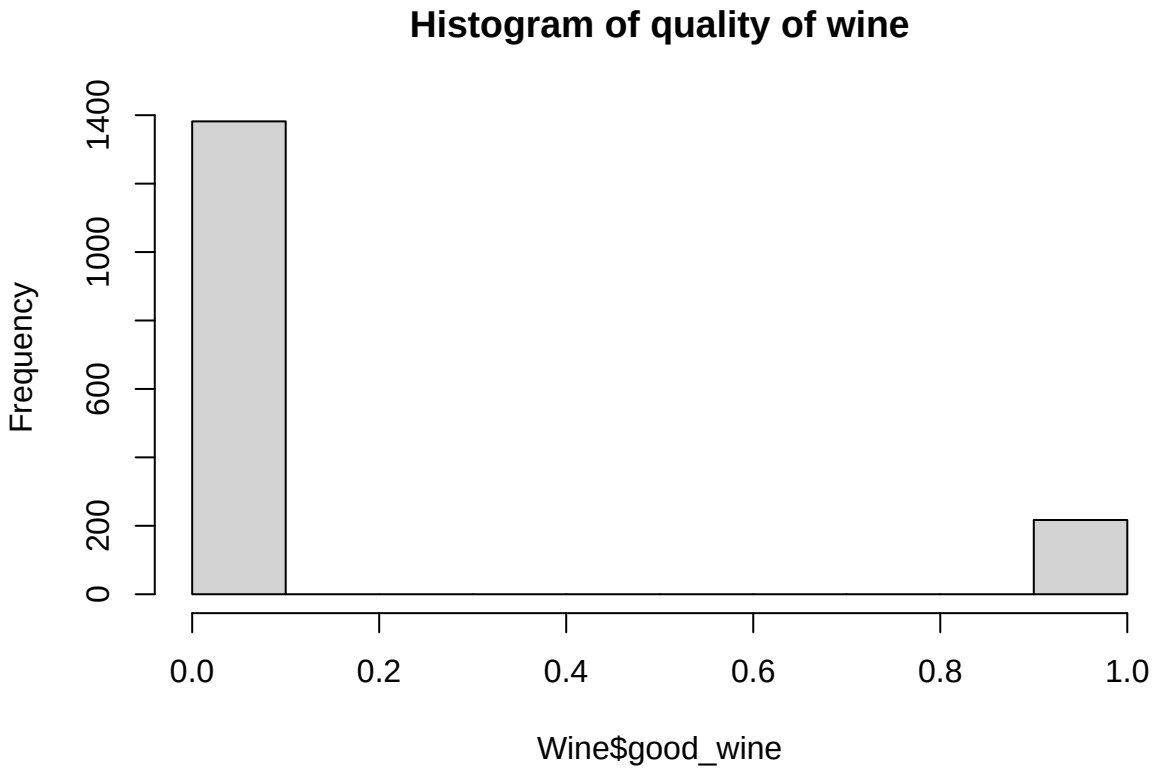
We analyze the `quality` column to check the variance of the data.



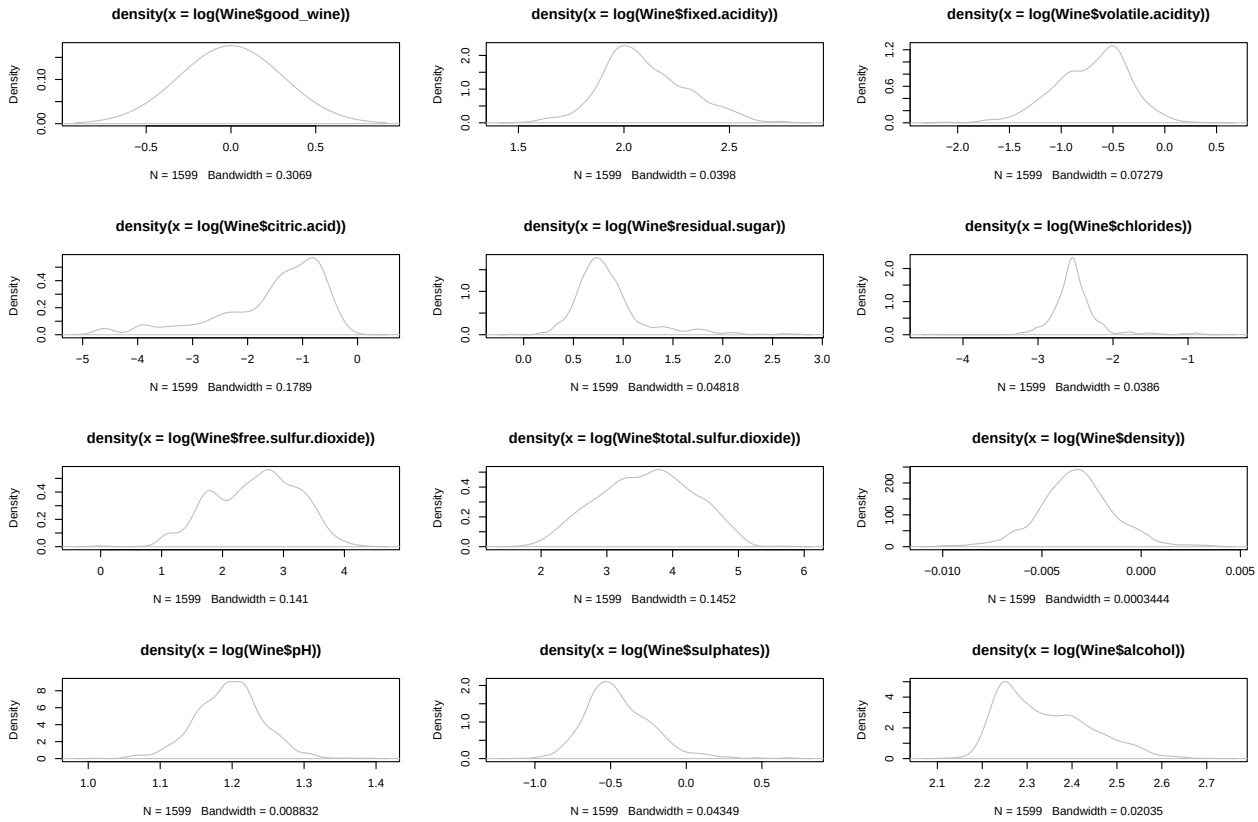
Before performing Logistic regression on the dataset, we need to specify the response variable which assume values either 0 or 1. In this analysis we are using the `quality` as response variable and we consider "good" wine with `quality` above 6.5 (included).

```
# Create a new variable good_wine for response variable with values 0, 1
Wine$good_wine <- ifelse(Wine$quality >= 6.5, 1, 0)
table(Wine$good_wine)
```

```
##
##    0    1
## 1382  217
```



We plot the densities of the covariants (log scale) to see the distribution of the parameters.



We can see that most of the covariants are normally distributed, with varying mean and wide range of variances.

2.2 Logistic Regression

At this stage we run frequentist analysis on the logistic model using the `glm()` function and analyze the significant coefficient of the model.

2.3 Frequentist analysis on the logistic model

We use `glm()` function to run the logistic regression on the model. In this model we use all the variables from the dataset. For this analysis, the response variable we set is the quality of the wine which is discrete variable that only takes two values, either 0 or 1. Thus, we set the Bernoulli probability model which is a memebr of Binomial family. As we will be performing Logistic regression so the link function in this case is `logit`.

```
# Logistic model using glm()
model <- glm(Wine$good_wine ~ Wine$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,
               data = Wine, family = binomial(link="logit"))
```

Significant coefficients of the model:

```
# Summary of the model
summary(model)
```

```
##
## Call:
## glm(formula = Wine$good_wine ~ Wine$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, family = binomial(link = "logit"),
##     data = Wine)
##
## Coefficients:
##                              Estimate Std. Error z value Pr(>|z|)
## (Intercept)                 2.428e+02  1.081e+02   2.247 0.024660 *
## Wine$fixed.acidity          2.750e-01  1.253e-01   2.195 0.028183 *
## Wine$volatile.acidity      -2.581e+00  7.843e-01  -3.291 0.000999 ***
## Wine$citric.acid            5.678e-01  8.385e-01   0.677 0.498313
## Wine$residual.sugar         2.395e-01  7.373e-02   3.248 0.001163 **
## Wine$chlorides             -8.816e+00  3.365e+00  -2.620 0.008788 **
## Wine$free.sulfur.dioxide    1.082e-02  1.223e-02   0.884 0.376469
## Wine$total.sulfur.dioxide -1.653e-02  4.894e-03  -3.378 0.000731 ***
## Wine$density               -2.578e+02  1.104e+02  -2.335 0.019536 *
## Wine$pH                     2.242e-01  9.984e-01   0.225 0.822327
## Wine$sulphates              3.750e+00  5.416e-01   6.924 4.39e-12 ***
## Wine$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
```

From the coefficients summary we observe that for some variables (e.g., fixed.acidity, residual.sugar, density) appear to be positive, while others (e.g., volatile.acidity, chlorides, total.sulfur.dioxide) are negative. This suggests that increasing the levels of some variables might be associated with higher predicted quality scores. On the other hand, increasing the levels of other variables might be linked to lower predicted quality scores.

The interpretation of the logistic regression function depends on the link function, here in log-odds. For example, the odds of having good quality wine, if the total sulfur dioxide level is, $exp(-0.0165) = 0.9836$ times that of the odds for not having a good quality wine when other variables are fixed.

We see that the Fisher scoring iteration is 6 here. That implies 6 iterations of the iterative algorithm were needed to reach the convergence.

2.4 Estimate the probabilities of "success" varying total.sulfur.dioxide

To compute the probability of having "good" wine varying `total.sulfur.dioxide` we need to fix the other covariants at its mean level. We get the 1000 samples ranging from minimum to maximum values of the `total.sulfur.dioxide` variable and then we calculate the probability of having success.

```
total_sulfur_dioxide_range1 <- seq(from=min(Wine$total.sulfur.dioxide),
                                   to=max(Wine$total.sulfur.dioxide),
                                   length.out = 1000)
```

Mean calculation for the covariants:

```
fixed_acidity_mean <- mean(Wine$volatile.acidity)
volatile_acidity_mean <- mean(Wine$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)

b0 <- model$coef[1]
b1 <- model$coef[2]
b2 <- model$coef[3]
b3 <- model$coef[4]
b4 <- model$coef[5]
b5 <- model$coef[6]
b6 <- model$coef[7]
b7 <- model$coef[8]
b8 <- model$coef[9]
b9 <- model$coef[10]
b10 <- model$coef[11]
b11 <- model$coef[12]
```

Fixing all the covariants at mean:

```
total_sulfur_dioxide_model1 <- 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_range1 +
  b8*density_mean + b9*pH_mean + b10*sulphates_mean + b11*alcohol_mean
```

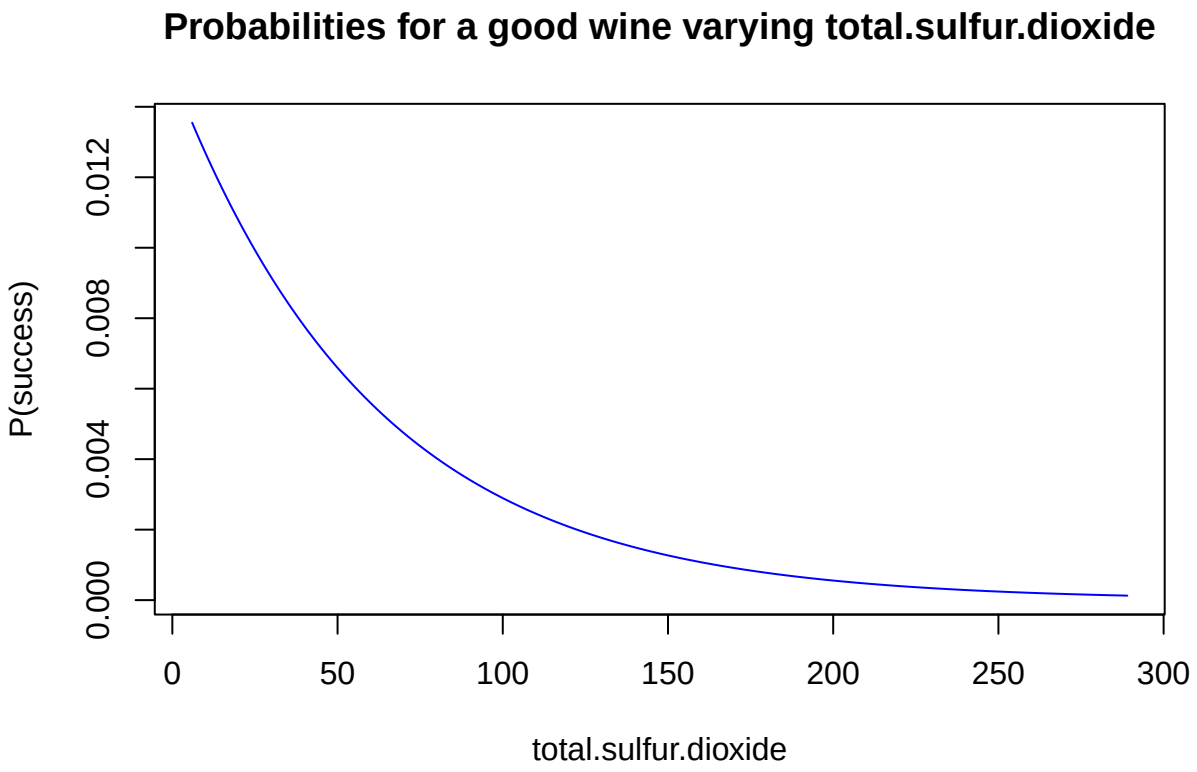
Calculating the probability of having success:

```
total_sulfur_dioxide_probs1 <-
  exp(total_sulfur_dioxide_model1)/
  (1 + exp(total_sulfur_dioxide_model1))
```

Plotting the probability of having success varying total.sulfur.dioxide:

```
par(mfrow=c(1,1))

plot(total_sulfur_dioxide_range1, total_sulfur_dioxide_probs1,
     type="l",
     lty=1,
     col="blue",
     xlab="total.sulfur.dioxide",
     ylab="P(success)",
     main="Probabilities for a good wine varying total.sulfur.dioxide")
```



2.5 Bayesian Analysis of the logistic model

In logistic regression, the response variable is binary (0,1). The values 0 and 1 can be interpreted necessarily as labels for two possible output ('success' and 'failure'). This model, then, can be used for basic classification in two classes.

2.5.1 Function for the log posterior distribution

The likelihood function for the logistic regression model is:

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

We need to define a prior distribution for the coefficients β. We use the most common priors for the linear model $\pi(\beta) = N_k(\beta_0, \Omega)$, is a normal prior in the form: $\beta_j \sim N(\mu_j, \sigma^2)$. The most common choice for μ is zero, and σ is usually chosen to be large enough to be considered as non-informative, common choices being in the range from $\sigma = 10$ to $\sigma = 100$. For this analysis we are using a vaguely informative prior distribution for β, $\beta_j \sim N(0, 100)$.

Now combining this prior distribution with the likelihood function produces the posterior distribution which is not in closed form. As we cannot analytically derive the posterior distribution of the vector of the parameters β, we need to make use of the Metropolis-Hastings algorithm and at the first step we need to write a function which will be the target distribution of the algorithm - i.e., $\pi(\beta|\mathbf{y}, \mathbf{x})$.

When computing posterior density ratios as required by the Metropolis-Hastings algorithm, the numbers involved can be very small; therefore always better to work in log-scale.

The log-likelihood function is:

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

Function for log posterior can be written below:

```
# Write R function for the log posterior distribution
lpost.LR <- function(beta,x,y)
{
  eta <- as.numeric(x %*% beta)

  logp <- eta - log(1+exp(eta))
  logq <- log(1-exp(logp))

  #log-likelihood
  logl <- sum(logp[y==1]) + sum(logq[y==0])

  #prior
  lprior <- sum(dnorm(beta,0,10,log=T))

  #posterior
  posterior <- logl + lprior

  return(posterior)
}
```

2.5.2 Fix number of simulation

For the purpose of the analysis we fix the number of simulation to 10^4.

```
# Fix number of simulations
S <- 10^4
```

2.5.3 Initialization with MLE

Now we need to choose initial values and the proposal distribution. We choose to initialize with maximum likelihood estimate (MLE) that we derive from the `glm()` function.

```r
X1=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)

X <- X1[1,] #covariants

y <- Wine$good_wine[1] #response variable

# Checking for NA
for(i in 1:nrow(X1))
{
  if(sum(is.na(X1[i,]))==0){
    X <- rbind(X,X1[i,])
    y <- c(y,Wine$good_wine[i])
  }
}

# Cretate beta matrix
beta_mat <- matrix(NA,nrow=S,ncol=ncol(X))
# Initializing beta matrix
beta_mat[1,] <- as.numeric(coefficients(model))
```

As we observe that the absolute value of the coefficients is quite variable, we want to use different tuning standard deviations for each parameter. We simulate all the coefficients together, and link the variance-covariance matrix to the covariates and this way the absolute value of the coefficients is linked to the variance associated to the MLE. This means that we want to use a multivariate normal distribution.

$$\beta^* \sim N(\beta^{iter-1}, c \times (\mathbf{X^T X})^{-1})$$

where c is a tuning parameter (for example: $c = 0.7$).

```r
# For prediction
y_new <- c(1)
x_new <- c(1,7.5,0.6,0.0,1.70,0.085,5,45,0.9965,3.40,0.63,12)

Omega_prop <- solve(t(X) %*% X)
k <- ncol(beta_mat)
acc <- 0

for(iter in 2:S)
{
  # 1. Propose a new set of values
  beta_star <- rmvnorm(1,beta_mat[iter-1,], 0.7*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_mat[iter-1,],ncol=1),X,y)

  # 3. Acceptance step
  if(runif(1,0,1)>exp(newpost-oldpost)){
    beta_mat[iter,]=beta_mat[iter-1,]
  } else{
    beta_mat[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_mat[iter,] * x_new) ) /
    (1 + exp(sum(beta_mat[iter,] * x_new) ))
  y_new[iter] <- rbinom(1,1,prob=p_new)

}
```

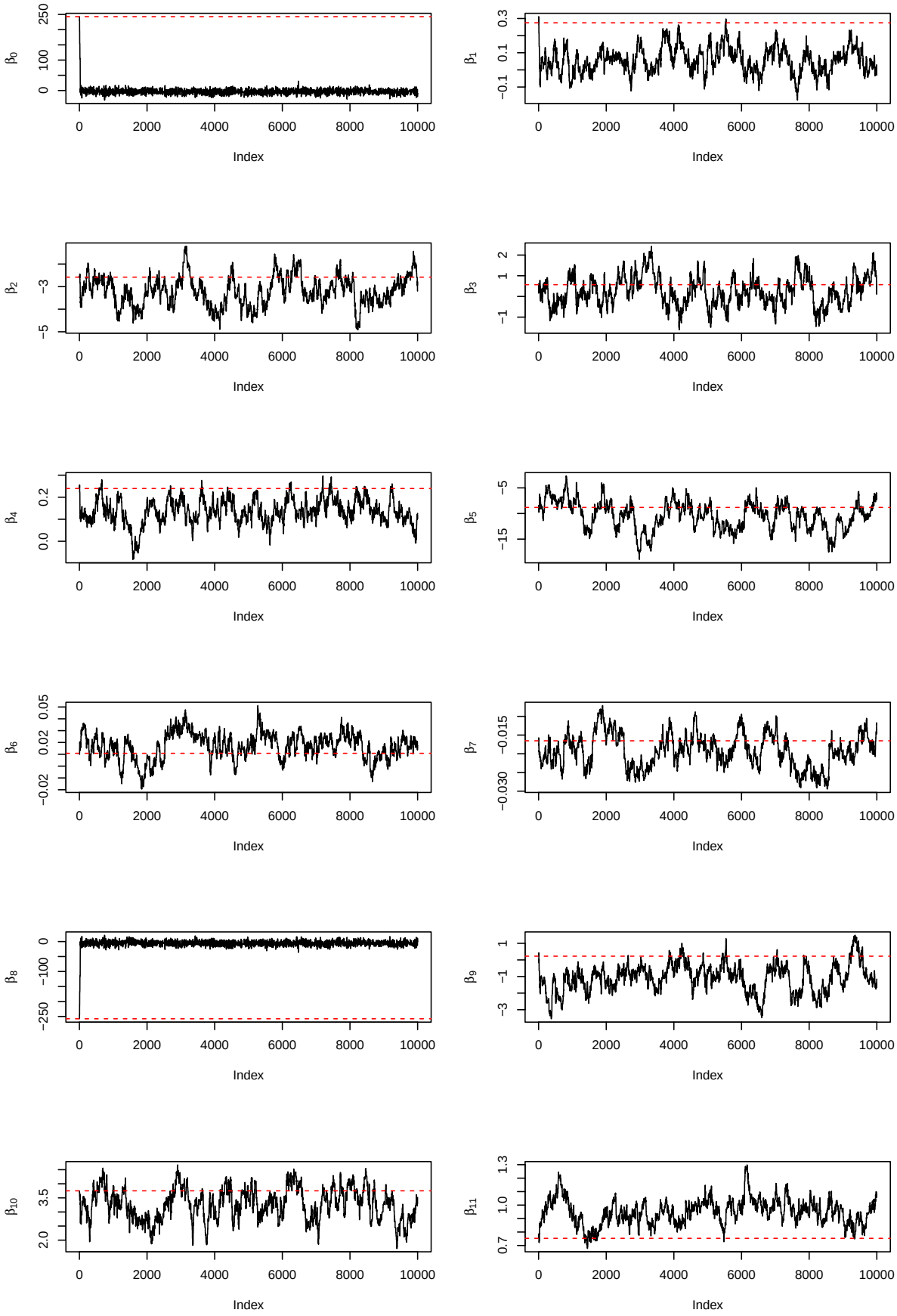
```
## [1] 1000.000    0.266
## [1] 2000.0000    0.2545
## [1] 3000.0000000    0.2493333
## [1] 4000.00000    0.23975
## [1] 5000.000    0.243
## [1] 6000.0000    0.2445
## [1] 7000.0000000    0.2445714
## [1] 8000.000000    0.245625
## [1] 9000.0000000    0.2435556
## [1] 1.000e+04 2.414e-01
```

Acceptance rate is approximate around 24% in this case.

```
acc/iter
```

```
## [1] 0.2414
```

Below is the plot of the markov chain for the initialization with MLE:


β0
250
100
0
β1
0.3
0.1
−0.1
β2
−3
−5
β3
2
1
−1
β4
0.2
0.0
β5
−5
−15
β6
0.05
0.02
−0.02
β7
−0.015
−0.030
β8
0
−100
−250
β9
1
−1
−3
β10
3.5
2.0
β11
1.3
1.0
0.7
0
2000
4000
6000
8000
10000
Index

After tuning the c we get the desired acceptance rate (24.5%) for $c = 0.7$. If we look at the chains we will see that some of the parameters converged and some are started to converge. We also observe that not all the convergence are around MLE.

2.5.4 Initialization with 4 different initial values

Now we need to vary the initial values for the coefficients. It is recommended to perform the analysis with several initial values for the coefficient and then see if all the chains converge to the same value. Here we choose 4 different initializations for the coefficient to run the Metropolis-Hasting algorithm. One is very common choice to initialize with maximum likelihood estimate (MLE) that we derive from the `glm()` function that we have already analyzed in the earlier section. Another choice is to initialize at zero for all the coefficients. For the other two choice we have choose some random methods, say for init2 we have used the coefficients multiplied by 0.5 and for init4 we have multiplied by 0.33.

- init1 - coefficients of the model
- init2 - coefficients of multiplied by 0.5
- init3 - coefficients of multiplied by 0.33
- init4 - all coefficients sets to 0

```
# Initializing with 4 different set of coefficients
init1 <- as.numeric(coefficients(model))
init2 <- as.numeric(coefficients(model)*0.5)
init3 <- as.numeric(coefficients(model)*0.33)
init4 <- as.numeric(coefficients(model)*0)
```

2.5.5 Use of Metropolis-Hastings for each initializations

We have already observe the result of running the Metropolis-Hasting algorithm with the standard initialization (MLE). In this section we run it again for all the 4 initializations we mentioned.

Initializing function to run for 4 different initalizations:

```
run_sim <- function(S, init, X, y, c){

  beta <- matrix(NA, nrow=S, ncol=ncol(X))
  beta[1,] <- as.numeric(init)

  Omega_prop <- solve(t(X) %*% X)

  y_new <- c()
  acc <- 0

  # Metropolis-Hastings algorithm
  for(i in 2:S)
  {
    # 1. Propose a new set of values
    beta_star <- rmvnorm(1, beta[i-1,], c*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[i-1,], ncol=1), X, y)

    # 3. Acceptance step
    if(runif(1,0,1)>exp(newpost-oldpost)){
```

```
      beta[i,]=beta[i-1,]
    }else{
      beta[i,]=beta_star
      acc=acc+1
    }

    # 4. Print the stage of the chain
    if(i%%1000==0){print(c(i, acc/i))}
    acc_ratio <- acc/i

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


  #return(cbind(beta_mat2, y_new, acc_ratio))
  return(list(beta=beta, y_new=y_new, acc_ratio=acc_ratio))
}
```

Running with 4 initializations:

Acceptance ratios of the simulations:

```
run1$acc_ratio
```

```
## [1] 0.2427
```

```
run2$acc_ratio
```

```
## [1] 0.25
```

```
run3$acc_ratio
```

```
## [1] 0.2491
```

```
run4$acc_ratio
```

```
## [1] 0.2488
```

2.5.6 Plot chains for each coefficients and comment

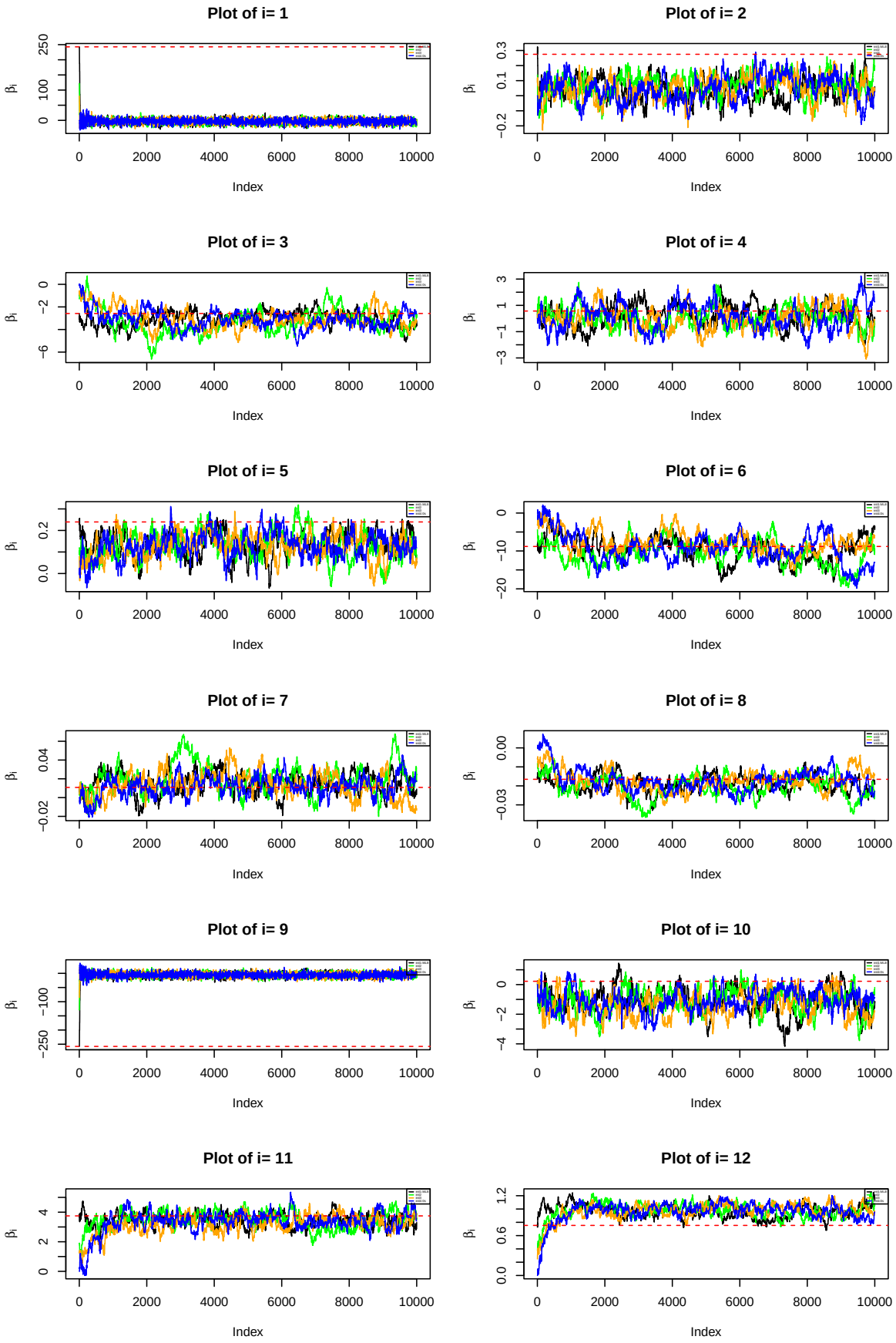
Plotting the chains:

```
# Plotting the chains for each coefficients
#par(mfrow=c(3,4))
par(mfrow=c(6,2))
plotting <- function(run1, run2, run3, run4){

  for(i in 1:ncol(X1)){
    #par(mfrow=c(4,3))
    plot(run1$beta[, i], type = 'l',
         ylab = expression(beta[i]),
         main = paste('Plot of i=', i),
         ylim = c(min(min(run1$beta[, i]), min(run2$beta[, i]), min(run3$beta[, i]), min(run4$beta[, i]
                  max(max(run1$beta[, i]), max(run2$beta[, i]), max(run3$beta[, i]), max(run4$beta[, i]
    lines(run2$beta[, i], type = "l", col = "green")
```

```
    lines(run3$beta[, i], type = "l", col = "orange")
    lines(run4$beta[, i], type = "l", col = "blue")
    abline(h=model$coefficients[i], col="red", lty=2)
    legend('topright',
           legend = c('init1:MLE', 'init2', 'init3', 'init4:0s'),
           col = c('black', 'green', 'orange', 'blue'),
           lty=1,
           cex=0.2)
  }
}

plotting(run1, run2, run3, run4)
```


Plot of i= 1
Plot of i= 2
Plot of i= 3
Plot of i= 4
Plot of i= 5
Plot of i= 6
Plot of i= 7
Plot of i= 8
Plot of i= 9
Plot of i= 10
Plot of i= 11
Plot of i= 12
βi
Index

In the plotting we observe several chains at overdispersed starting points. We see for some of the covarints the burn in is around 5000 iterations when all the chains converge to the same point. We also observe that almost all the covariants are converging around MLE whereas others are converging at a point where the data points are.

We can also see that the variance of the values between chains and within chains are very low.

2.6 Approximate the posterior predictive distribution of an unobserved variable

Prediction for an unobserved variable for the given parameter:

Unobserved variable with the parameters:

- fixed acidity: 7
- volatile acidity: 0.6
- citric acid: 0.0
- residual sugar: 1.70
- chlorides: 0.085
- free sulfur dioxide: 5
- total sulfur dioxide: 45
- density: 0.9965
- pH: 3.40
- sulphates: 0.63
- alcohol: 12

Now we that given the values of the parameters below, we get the probabilities of having a good wine is very low. For the 4 initializations we get approximately 12.8%, 11.2%, 15.6% and 14.1% accordingly.

```
cat("Prediction for run1:",prob_run1, "\n")
```

```
## Prediction for run1: 0.1240124
```

```
print(table(run1$y_new))
```

```
##
##    0    1
## 8759 1240
```

```
cat("Prediction for run2:",prob_run2, "\n")
```

```
## Prediction for run2: 0.109711
```

```
print(table(run2$y_new))
```

```
##
##    0    1
## 8902 1097
```

```
cat("Prediction for run3:",prob_run3, "\n")
```

```
## Prediction for run3: 0.1411141
```

```
print(table(run3$y_new))
```

```
##
##    0    1
## 8588 1411
```

```
cat("Prediction for run4:",prob_run4, "\n")
```

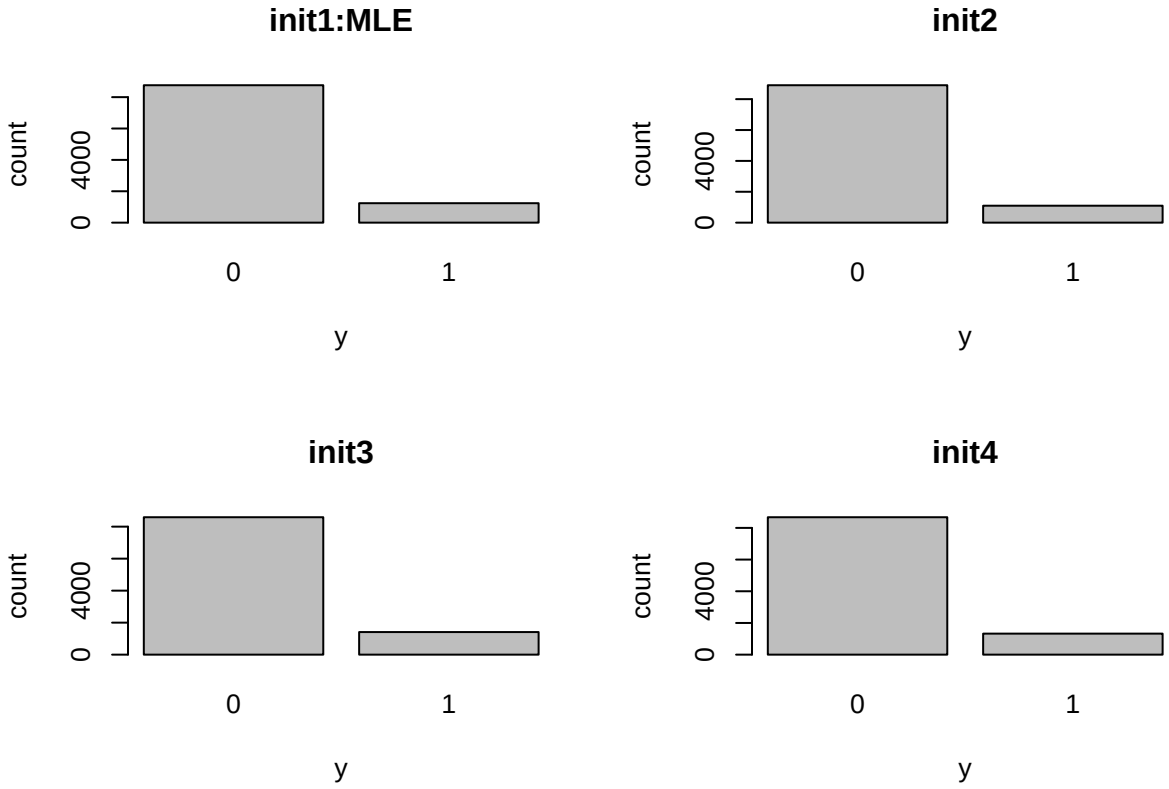
```
## Prediction for run4: 0.1327133
```

```
print(table(run4$y_new))
```

```
##
##    0    1
## 8672 1327
```

We plot the histogram for the predictions:

```
#Barplots
par(mfrow=c(2,2))
barplot(pred_run1, main="init1:MLE", ylab="count", xlab="y")
barplot(pred_run2, main="init2", ylab="count", xlab="y")
barplot(pred_run3, main="init3", ylab="count", xlab="y")
barplot(pred_run4, main="init4", ylab="count", xlab="y")
```

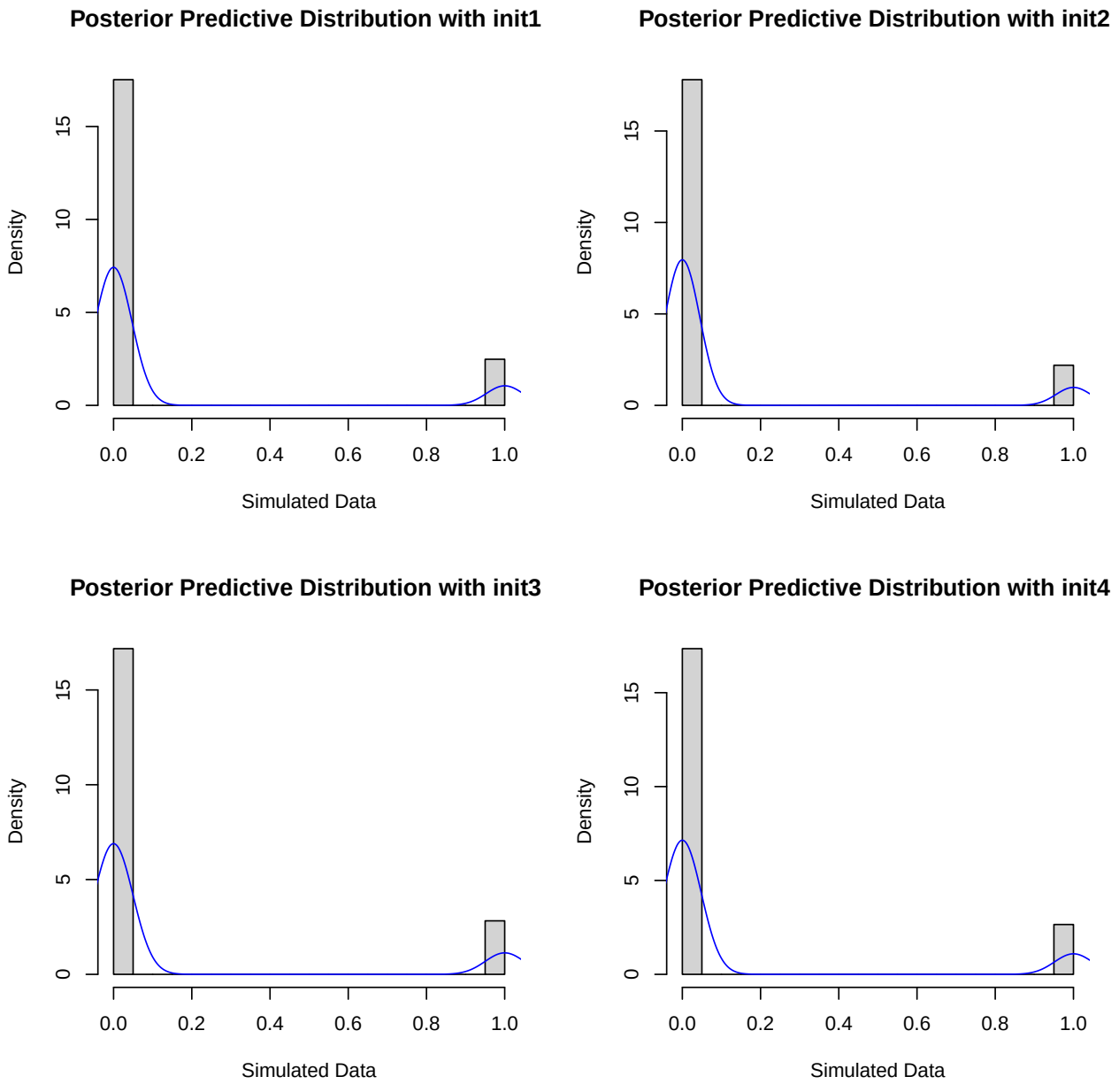


We can observe from the plots we can say that for the given parameter probability of having not quality wine is very low.

Plotting the density of the predictive distribution for all four initializations:

```
#PDF
par(mfrow=c(2,2))
# Visualize the simulated data
hist(run1$y_new, freq = FALSE,
     main = "Posterior Predictive Distribution with init1 ",
     xlab = "Simulated Data")
     lines(density(run1$y_new), col = "blue")
hist(run2$y_new, freq = FALSE,
     main = "Posterior Predictive Distribution with init2 ",
     xlab = "Simulated Data")
```

```r
lines(density(run2$y_new), col = "blue")
hist(run3$y_new, freq = FALSE,
     main = "Posterior Predictive Distribution with init3 ",
     xlab = "Simulated Data")
lines(density(run3$y_new), col = "blue")
hist(run4$y_new, freq = FALSE,
     main = "Posterior Predictive Distribution with init4 ",
     xlab = "Simulated Data")
lines(density(run4$y_new), col = "blue")
```



2.7 Use metrop() for analysis

In this section we do the same analysis with `metrop()` function which is available in the `mcmc` package.

Compare results with plot

```
x0 <- X
y0 <- Wine$good_wine[1]
Omega_prop0 <- solve(t(x0) %*% x0)

# Using metro()
m_out1 <- metrop(obj=lpost.LR, initial=init1, nbatch=S,
                 x=x0, y=y0, scale=6)
m_out2 <- metrop(obj=lpost.LR, initial=init2, nbatch=S,
                 x=x0, y=y0, scale=6)

m_out3 <- metrop(obj=lpost.LR, initial=init3, nbatch=S,
                 x=x0, y=y0, scale=6)

m_out4 <- metrop(obj=lpost.LR, initial=init4, nbatch=S,
                 x=x0, y=y0, scale=6)




# Print summary of the MCMC output
(m_out1$accept)
```

```
## [1] 0.2323
```

```
(m_out2$accept)
```

```
## [1] 0.2309
```

```
(m_out3$accept)
```

```
## [1] 0.2306
```

```
(m_out4$accept)
```

```
## [1] 0.2253
```

After varying with sevaral scalar value for the scale parameter we fix the value 6 where we approximate.

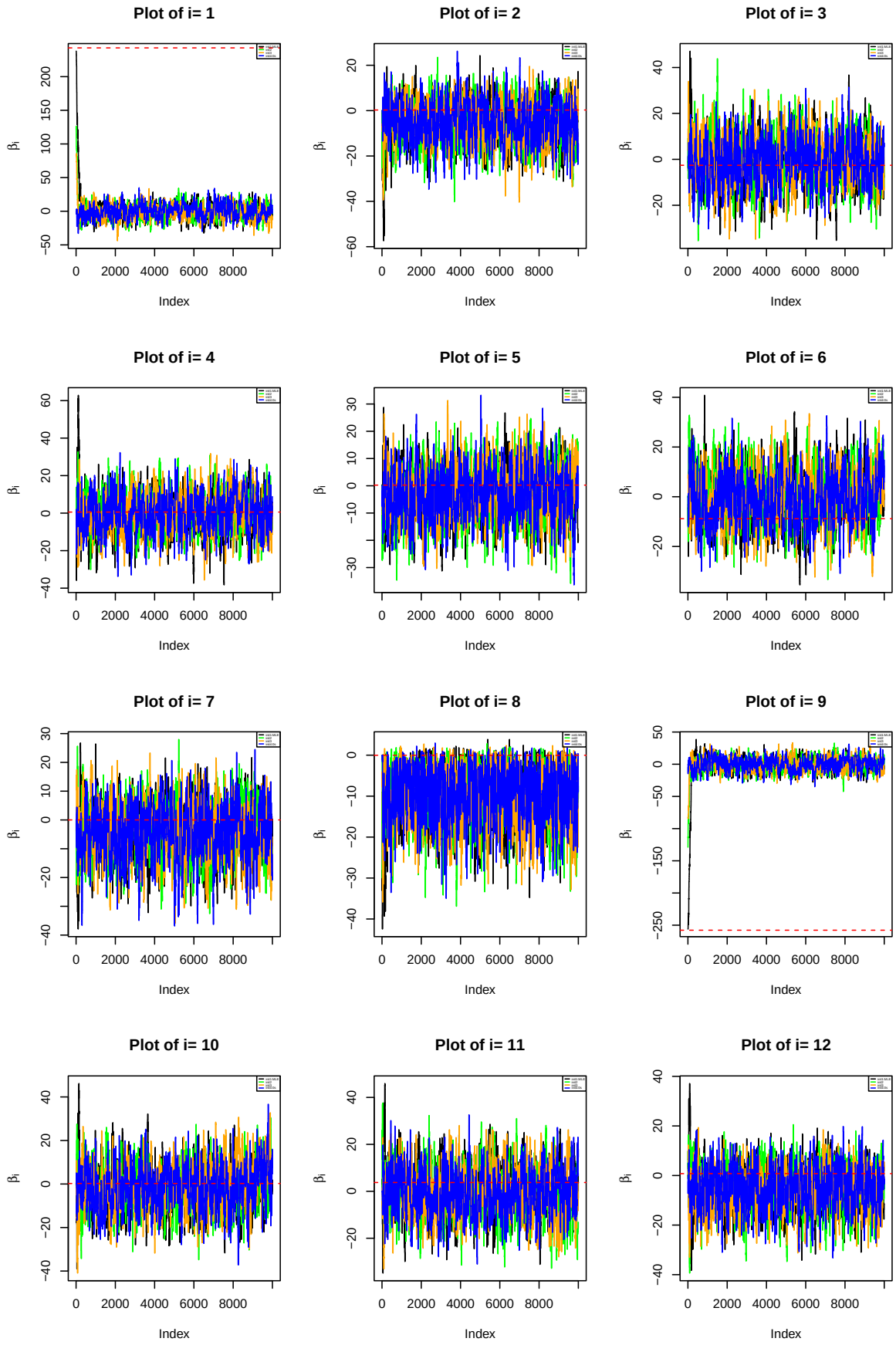
Plotting for the markov chains with 4 initializations:

```
# Plotting metrop() output for 4 initialization
par(mfrow=c(4,3))
plotting_metrop <- function(m_out1, m_out2, m_out3, m_out4){

  for(i in 1:ncol(X1)){
    plot(m_out1$batch[, i], type = 'l',
         ylab = expression(beta[i]),
         main = paste('Plot of i=', i),
         ylim = c(min(min(m_out1$batch[, i]), min(m_out2$batch[, i]),
                      min(m_out3$batch[, i]), min(m_out4$batch[, i])),
                      max(max(m_out1$batch[, i]), max(m_out2$batch[, i]),
                      max(m_out3$batch[, i]), max(m_out4$batch[, i]))))
    lines(m_out2$batch[, i], type = "l", col = "green")
    lines(m_out3$batch[, i], type = "l", col = "orange")
    lines(m_out4$batch[, i], type = "l", col = "blue")
    abline(h=model$coefficients[i], col="red", lty=2)
    legend('topright',
```

```
           legend = c('init1:MLE', 'init2', 'init3', 'init4:0s'),
           col = c('black', 'green', 'orange', 'blue'),
           lty=1,
           cex=0.2)
  }
}

plotting_metrop(m_out1, m_out2, m_out3, m_out4)
```


Plot of i= 1
Plot of i= 2
Plot of i= 3
Plot of i= 4
Plot of i= 5
Plot of i= 6
Plot of i= 7
Plot of i= 8
Plot of i= 9
Plot of i= 10
Plot of i= 11
Plot of i= 12
βi
Index