Red wine quality analysis: a bayesian approach to binary and polychotomous response data

Giulia Cataldo, Bianca Cattaneo

1. DATA PRESENTATION and PREPARATION

The ***winequality-red*** dataset contains a total of 12 variables, which were recorded for 1599 observations. The dataset consists of red wine samples. The explanatory variables include objective tests (e.g. pH values) and the response variable is based on sensory data (median of at least 3 evaluations made by wine experts). Each expert graded the wine quality between 0 (very bad) and 10 (excellent). Actually, in this list, there are only varieties of red wine which have been graded between 3 and 8.

We now start a first exploratory analysis

```
data <- read.csv("winequality-red.csv")
res1 <- as.matrix(head(data[,1:6]))
res2 <- as.matrix(head(data[,7:12]))
```

The first few values of each variable:

fixed.acidity	volatile.acidity	citric.acid	residual.sugar	chlorides	free.sulfur.dioxide
7.4	0.70	0.00	1.9	0.076	11
7.8	0.88	0.00	2.6	0.098	25
7.8	0.76	0.04	2.3	0.092	15
11.2	0.28	0.56	1.9	0.075	17
7.4	0.70	0.00	1.9	0.076	11
7.4	0.66	0.00	1.8	0.075	13

total.sulfur.dioxide	density	pH	sulphates	alcohol	quality
34	0.9978	3.51	0.56	9.4	5
67	0.9968	3.20	0.68	9.8	5
54	0.9970	3.26	0.65	9.8	5
60	0.9980	3.16	0.58	9.8	6
34	0.9978	3.51	0.56	9.4	5
40	0.9978	3.51	0.56	9.4	5

```
# checking for missing values
apply(is.na(data), 2, which)
```

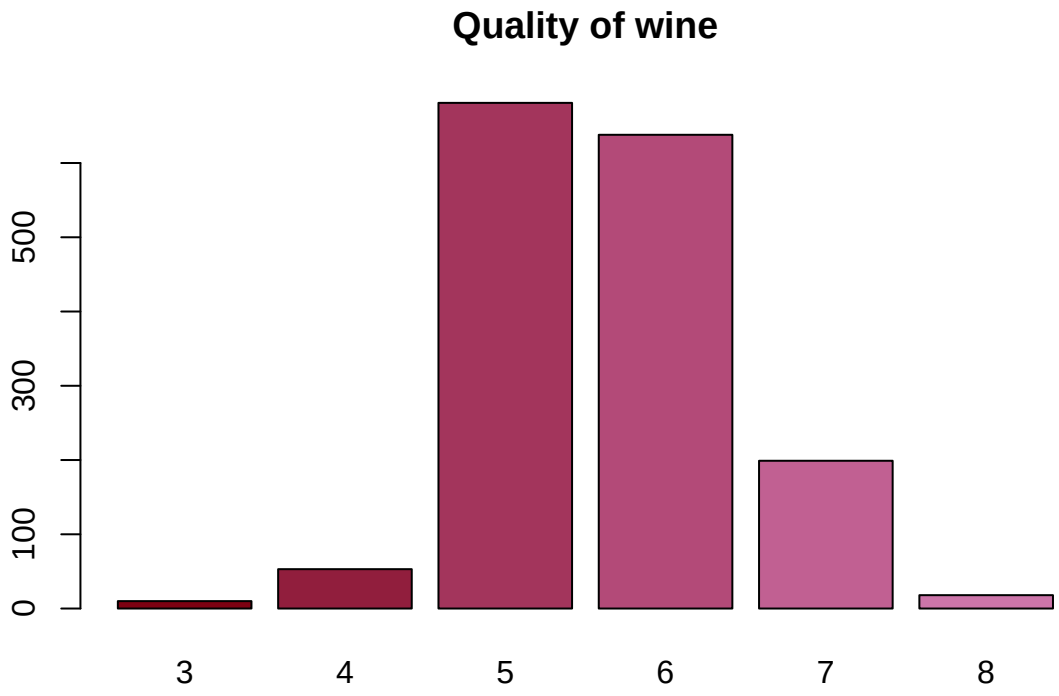
```
## integer(0)
```

```
# checking the frequency of each category
c = c(sum(as.numeric(data$quality==3)),sum(as.numeric(data$quality==4)),
      sum(as.numeric(data$quality==5)),sum(as.numeric(data$quality==6)),
      sum(as.numeric(data$quality==7)),sum(as.numeric(data$quality==8)))

res <- matrix(c, ncol=6)
colnames(res) <- c('3', '4', '5', '6', '7', '8')
row.names(res) <- 'Frequencies'
```

	3	4	5	6	7	8
Frequencies	10	53	681	638	199	18

We can see that there are no missing data and that not all the categories seem to be well represented.

```
my_col = hcl.colors(n = 13, palette='Red-Purple', fixup=TRUE)
Tab_prog = table(data$quality)
barplot(Tab_prog, main = "Quality of wine", col = my_col)
```

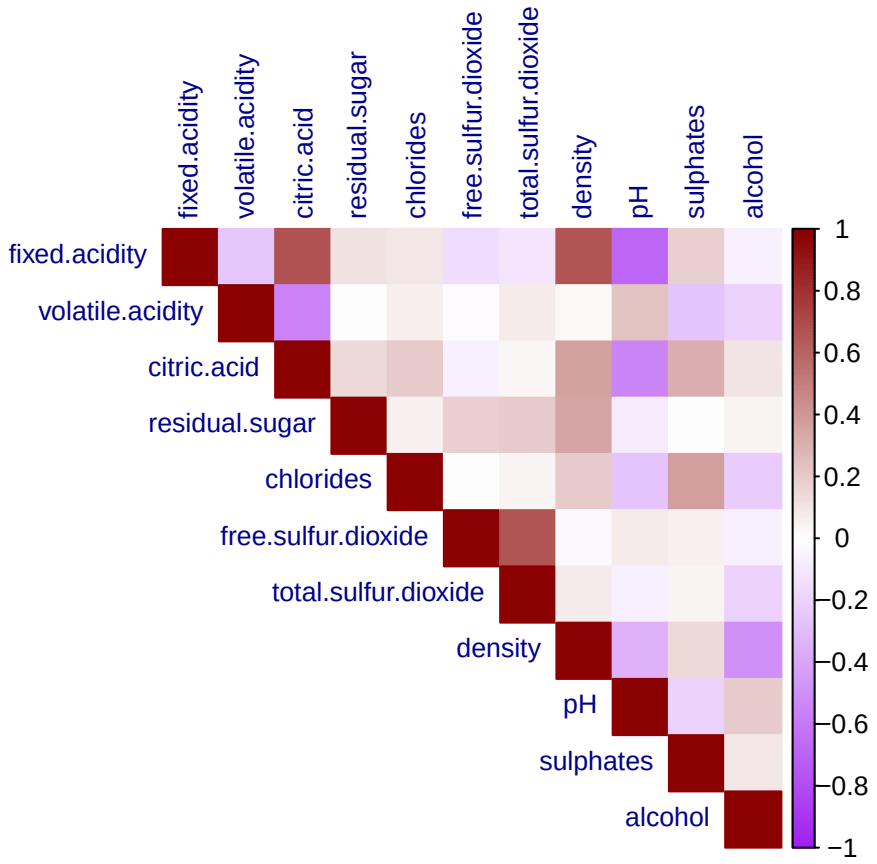


Furthermore, looking at the covariates, we can intuitively hypothesize a high correlation between some of them. In particular we can see that

- There are several variables that try to explain the level of acidity ('***ph***', '***fixed acidity***', '***volatile acidity***', '***citric acid***').
- A high quantity of ***alcohol*** is associated with a lower ***density***.
- The variable '***free sulfur dioxide***' explains partially the variable '***total sulfur dioxide***'.

We now look at the effective correlation between variables

```
X = subset(data, select = -quality)
corrplot(cor(X), method="color", type='upper', tl.col = "darkblue", tl.srt = 90,
         tl.cex = 0.8, col=colorRampPalette(c("purple","white","darkred"))(200))
```

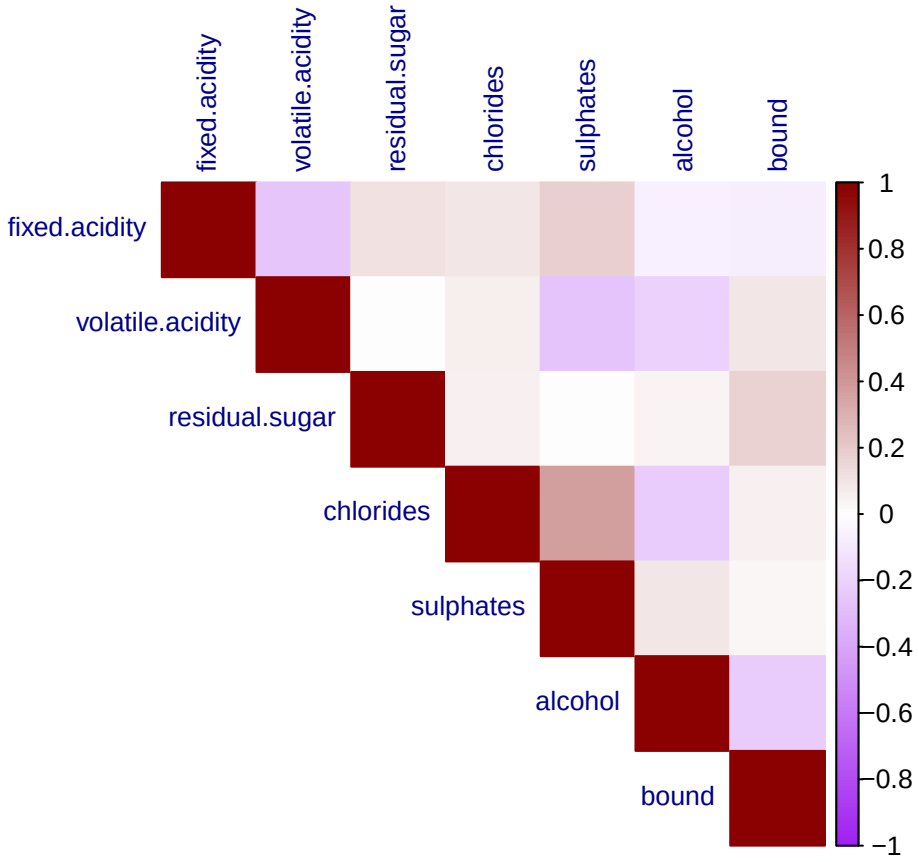


This plot seems to confirm our hypothesis, we therefore decide to remove the *'density'*, *'citric acid'*, and *'ph'* variables and to substitute the *'total sulfur dioxide'* and the *'free sulfur dioxide'* with the ***difference*** of the two.

```
data$bound <- data$total.sulfur.dioxide - data$free.sulfur.dioxide
data1 <- subset(data, select = -c(pH, free.sulfur.dioxide, density, citric.acid,
                                  total.sulfur.dioxide))
```

Looking at the new correlation plot, we can see that the correlation between the variables is now very low.

```
X1 = subset(data1, select = -quality)
p <- ncol(X1)
corrplot(cor(X1), method="color", type='upper', tl.col = "darkblue", tl.srt = 90,
         tl.cex = 0.8,
         col=colorRampPalette(c("purple","white","darkred"))(200))
```



2. THE APPROACHES WE HAVE CHOSEN

Since we are dealing with a response that consists in **ordered categories**, we have decided to implement two different **probit models**.

1. As a first approach, we convert the response variable into a two-level variable and implement a **Binary Probit Model**.
2. As a second approach, we convert the response variable into a three-level variable and implement a **Multivariate Probit Model**.

3. PROBIT MODEL FOR BINARY DATA

Suppose we have N independent random variables $y_i \in \{0, 1\}$ where each y_i comes from a Bernoulli distribution with probability of success p_i.

$$Y_i|p_i \overset{ind}{\sim} Bern(p_i)$$

We also assume that the success probability p_i is related to a vector of covariates $\mathbf{x}_i = (x_{i,1}, ..., x_{i,p})$. Then, we define the **binary regression model** as

$$p_i = g^{-1}(\beta^T \mathbf{x}_i)$$

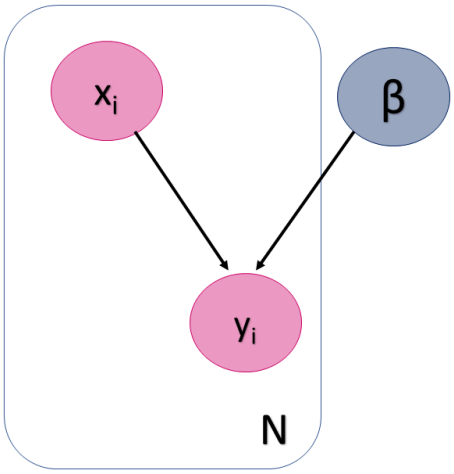
where β represents a $(p \times 1)$ column vector of regression coefficients and $g^{-1}(.)$ is the inverse link function.

The **probit model** is obtained if we define $g^{-1}(.) = \mathbf{\Phi}(.)$, where $\mathbf{\Phi}(.)$ denotes the **cumulative distribution function of the standard normal distribution**.

Our goal is to estimate the model parameters and we can do that by trying to compute their posterior distribution. In order to do that, we need to take a prior distribution for the parameters, $\pi(\beta)$. The main elements for the Bayesian binary probit regression model are now

$$
\begin{aligned}
y_i &\sim Bernoulli(\Phi(\eta_i)) \\
\eta_i &= \beta^T x_i \\
\beta &\sim \pi(\beta)
\end{aligned}
$$

and the graphical model can be represented in the following way



The posterior distribution would be the following

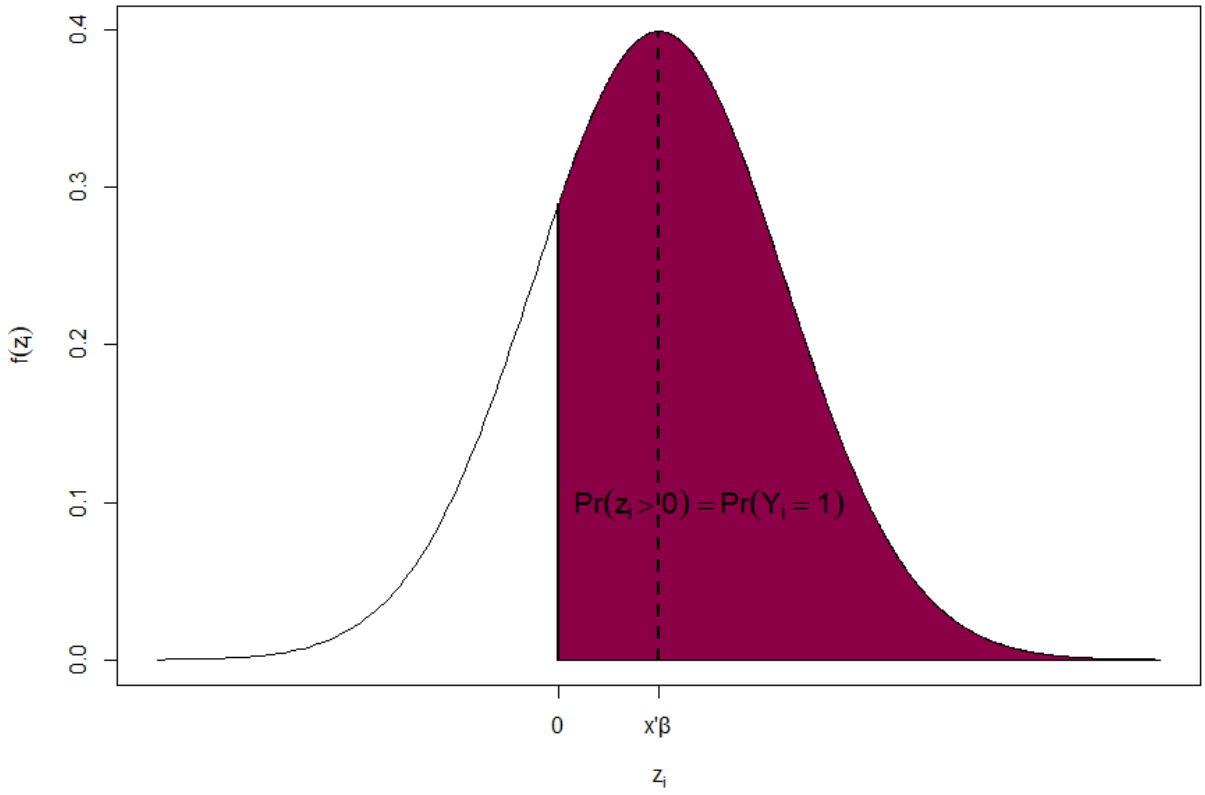
$$p(\beta|y, X) \propto p(\beta)p(\mathbf{y}|\beta, \mathbf{X}) = \pi(\beta)\prod_{i=1}^{N} p(y_i|\theta, \mathbf{x_i}) = \pi(\beta)\prod_{i=1}^{N} \Phi(\beta^T\mathbf{x_i})^{y_i}(1 - \Phi(\beta^T\mathbf{x_i}))^{(1-y_i)}$$

However notice how, in a Bayesian framework, performing inference on this model is complicated by the fact that **no conjugate prior $\pi(\beta)$ exists for the parameters** of the probit regression.

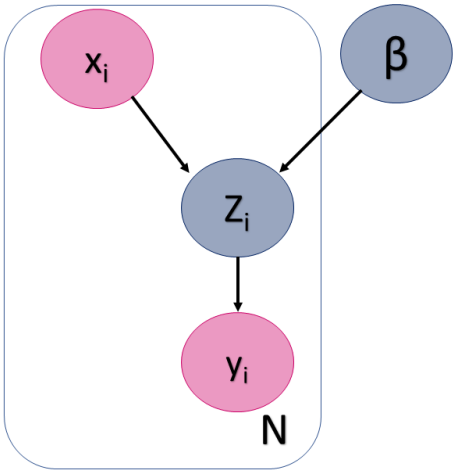
To overcome this issue, Albert and Chib (1993) ***augmented*** the original model with an additional auxiliary variable (***latent variable***) that renders the conditional distributions of the model parameters similar to those under a Bayesian normal linear regression model with Gaussian noise. This way it has been possible to derive an **efficient Gibbs sampling scheme** for computing the posterior distribution.

3.1 AUGMENTED MODEL

Let us introduce N independent latent variables z_i, where each z_i follows a Normal distribution.



Then, the augmented probit model has the following structure:



where

$$
\begin{aligned}
y_i &= \begin{cases} 1 & if\ z_i > 0 \\ 0 & if\ z_i \leq 0 \end{cases} \\
z_i &= \beta^T \mathbf{x_i} + \epsilon_i \\
\epsilon_i &\sim N(0,1) \\
\beta &\sim \pi(\beta)
\end{aligned}
$$

where y_i is now deterministic conditional on the sign of the stochastic auxiliary variable z_i. Therefore, we formulate the original problem as a missing data problem where we have a Normal regression model on the latent data z_i and the observed responses y_i are incomplete in that we only observe whether $z_i > 0$ or $z_i \leq 0$.

We are now interested in computing the **joint posterior distribution** of the latent variables $\mathbf{z}$ and the model parameters β given the data $\mathbf{y}$ and $\mathbf{X}$

$$
p(\mathbf{z}, \beta | \mathbf{y}, \mathbf{X}) \propto p(\beta) p(\mathbf{z} | \beta, \mathbf{X}) p(\mathbf{y} | \mathbf{z}) = \pi(\beta) \prod_{i=1}^{N} p(z_i | \beta, \mathbf{x_i}) p(y_i | z_i) = prior\ on\ \beta\ \times\ augmented\ likelihood
$$

where we have that

$$
\begin{aligned}
p(z_i | \beta, \mathbf{x_i}) &= N(z_i | \beta^T x_i, 1) \\
p(y_i | z_i) &= \mathcal{I}(y_i = 1)\mathcal{I}(z_i > 0) + \mathcal{I}(y_i = 0)\mathcal{I}(z_i \leq 0)
\end{aligned}
$$

3.2 GIBBS SAMPLING SCHEME

This joint posterior is difficult to normalize and sample from directly. However, computation of the **marginal posterior** of β and $\mathbf{z}$ using the Gibbs sampling only requires computing the **full conditional distributions**, $p(\beta | \mathbf{z}, \mathbf{y}, \mathbf{X})$ and $p(\mathbf{z} | \beta, \mathbf{y}, \mathbf{X})$, which are of standard form.

Furthermore, it should be underlined that $p(\beta | \mathbf{z}, \mathbf{y}, \mathbf{X}) = p(\beta | \mathbf{z}, \mathbf{X})$, since β is conditionally independent of $\mathbf{y}$ given $\mathbf{z}$.

The full conditional of β is written as:

$$
p(\beta | z, X) \propto \pi(\beta) \prod_{i=0}^{N} N(z_i | \beta^{\mathbf{T}} \mathbf{x_i}, \mathbf{1})
$$

if we assign the following proper conjugate prior to β

$$
\beta \sim N_p(\beta_{\mathbf{0}}, \mathbf{V^{-1}})
$$

we can write the **full conditional of** β as:

$$
\beta | z, X \sim N_p(A^{-1}\mathbf{b}, A^{-1})
$$

where $A^{-1} = (X^T X + V)^{-1}$ and $A^{-1}\mathbf{b} = (X^T X + V)^{-1}(\mathbf{z}^T X + \beta_0^T V)$

Moreover, we can write the **full conditional of z** as a truncated normal distribution,

$$
z_i | \beta, y_i, \mathbf{x_i} \sim \begin{cases} tN(\beta^{\mathbf{T}} \mathbf{x_i}, 1, 0, \inf) & if\ y_i = 1 \\ tN(\beta^{\mathbf{T}} \mathbf{x_i}, 1, -\inf, 0) & if\ y_i = 0 \end{cases}
$$

Gibbs sampling is an MCMC algorithm that simulates draws that are dependent and these draws are approximately from the probability distribution of interest, i.e. the target distribution $p(\mathbf{z}, \beta|\mathbf{y}, \mathbf{X})$.

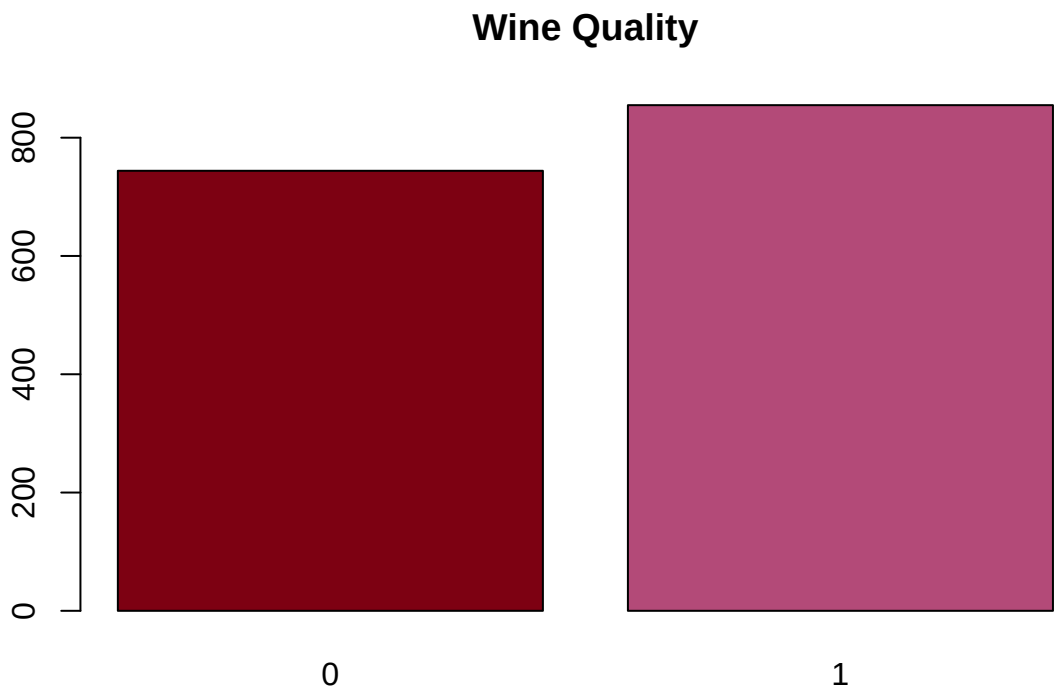
In each simulation step of the Gibbs algorithm we have a sample $\beta|\mathbf{z}, \mathbf{X}$ and $z_i|\beta, y_i, \mathbf{x_i}$ in turn.

The following code implements the Gibbs sampling algorithm for the Bayesian binary probit on the ***winequality-red*** dataset.

First we convert the response variable into a binary response. The classification is now as follows:

- Quality: **Bad - 0**
  - categories included: 3,4,5
- Quality: **Good - 1**
  - categories included: 6,7,8

```
data1$quality = as.factor(data1$quality)
levels(data1$quality) <- c(0,0,0,1,1,1)
barplot(table(data1$quality), col=hcl.colors(5, palette = 'Red-Purple'),
        main = 'Wine Quality')
```

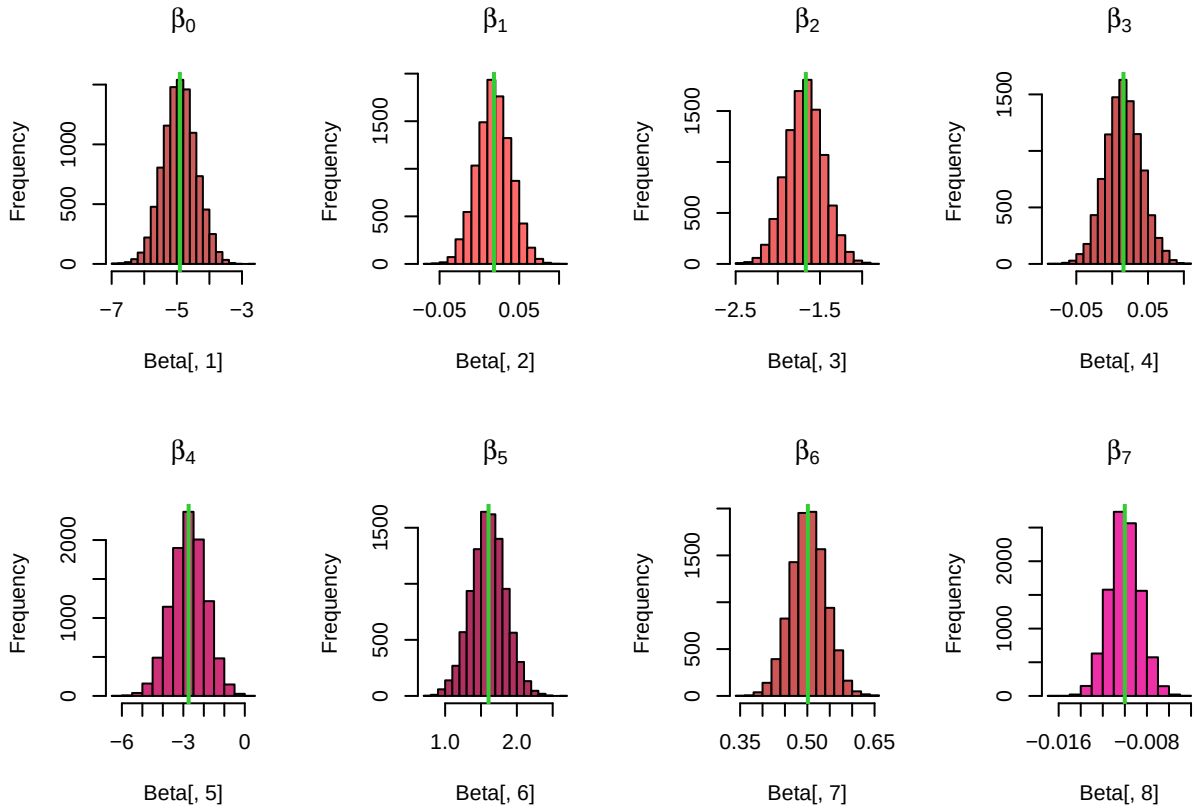


We now proceed with the implementation of the algorithm

```
y <- data1$quality
X1 <- model.matrix(y~., X1)

n = nrow(X1)
p = ncol(X1)
ns <- table(y)
```

```r
# Priors
beta.0 <-  rep(0, p)                          # Prior Mean for Beta
V <- diag(0.01, p)                          # Prior Precision of Beta (vague)

# Initial Values
beta<-glm(y ~ X1-1, family = binomial(link = probit))$coefficients # MLE
z<-rep(0,n) # Latent Normal Variable
Y_hat_s <- rep(0,n)
Z <- rep(0,n)

# Create matrices to store results
nsim<-50000           # Number of MCMC Iterations
thin<-5             # Thinning interval
burn<-100                 # Burnin
lastit<-(nsim-burn)/thin      # Last stored value
Beta <- matrix(NA, lastit, p)
Y_hat <- matrix(NA, lastit, n)

# mode function for prediction
mode <- function(v) {
  uniqv <- unique(v)
  uniqv[which.max(tabulate(match(v, uniqv)))]
}

####################
# Gibbs Sampler   #
####################
tmp<-proc.time()
for (i in 1:nsim) {

  muz<-X1%*%beta # mean of Z
  z[y == 0] <- rtruncnorm(ns[1], mean = muz[y == 0], sd = 1, a = -Inf, b = 0)
  z[y == 1] <- rtruncnorm(ns[2], mean = muz[y == 1], sd = 1, a = 0, b = Inf)

  #################
  # Prediction    #
  #################
  Z <- rnorm(n, muz, 1)
  for (k in 1:n){
    if (Z[k] < 0) {
      Y_hat_s[k] = 0
      }else{
        Y_hat_s[k] = 1
    }
  }


  # Update beta
  V.n    = V + t(X1)%*%X1
  beta.n = solve(V + t(X1)%*%X1)%*%(t(X1)%*%z + V%*%beta.0)
  beta = c(rmvnorm(1, beta.n, solve(V.n)))

  #################
```

```
  # Store Results #
  ##################

  if (i> burn & i%%thin==0) {
    j<-(i-burn)/thin
    Beta[j,] <- beta
    Y_hat[j,] <- Y_hat_s
}

  if(i%%100==0) print(i)
}

proc.time()-tmp
```

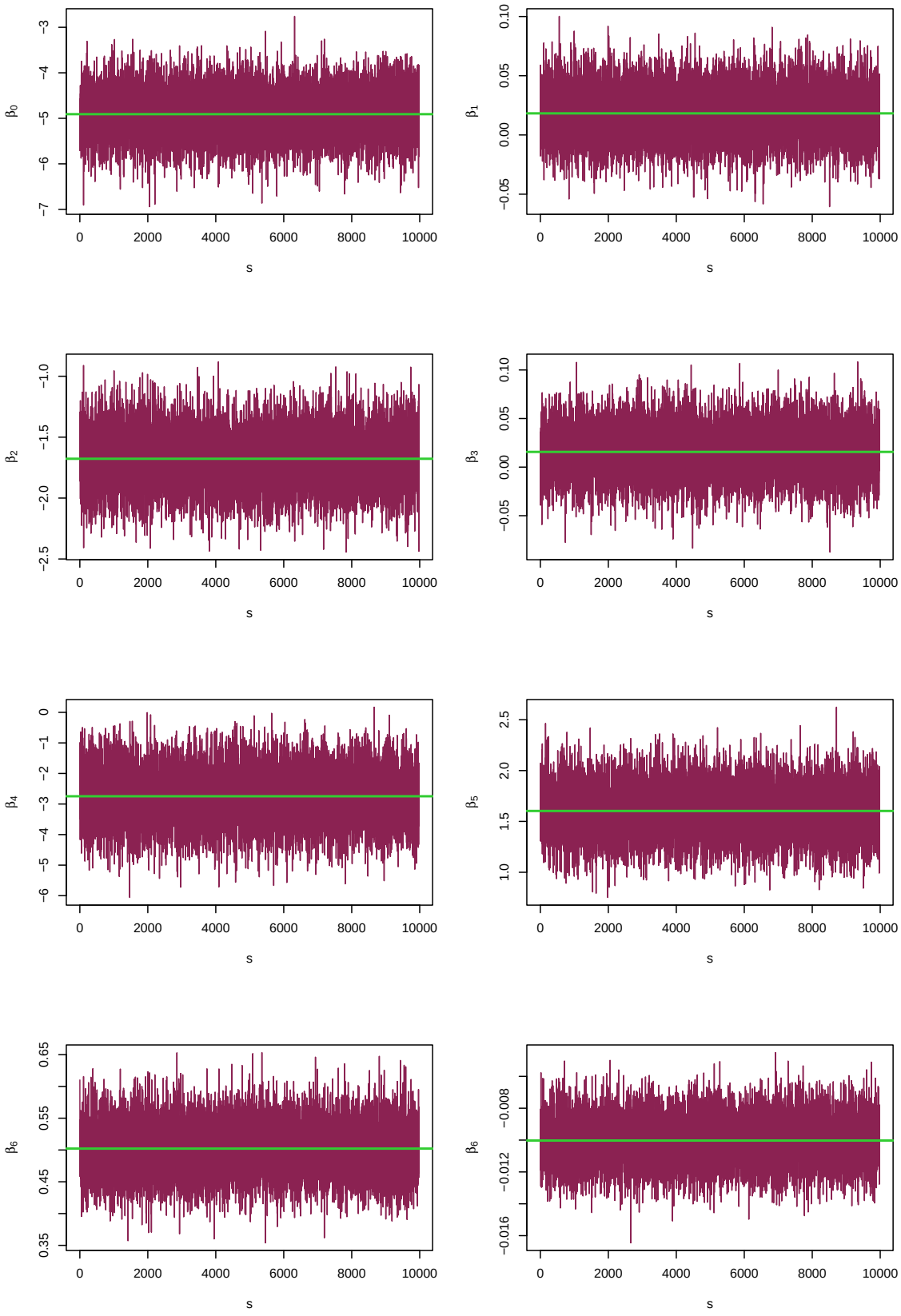


3.2.1 MCMC DIAGNOSTIC

Since MCMC is a numerical technique, it is also subject to approximation error. For this reason, we decide to check if the chains produced by the algorithm provide a good approximation of the true posterior distribution.

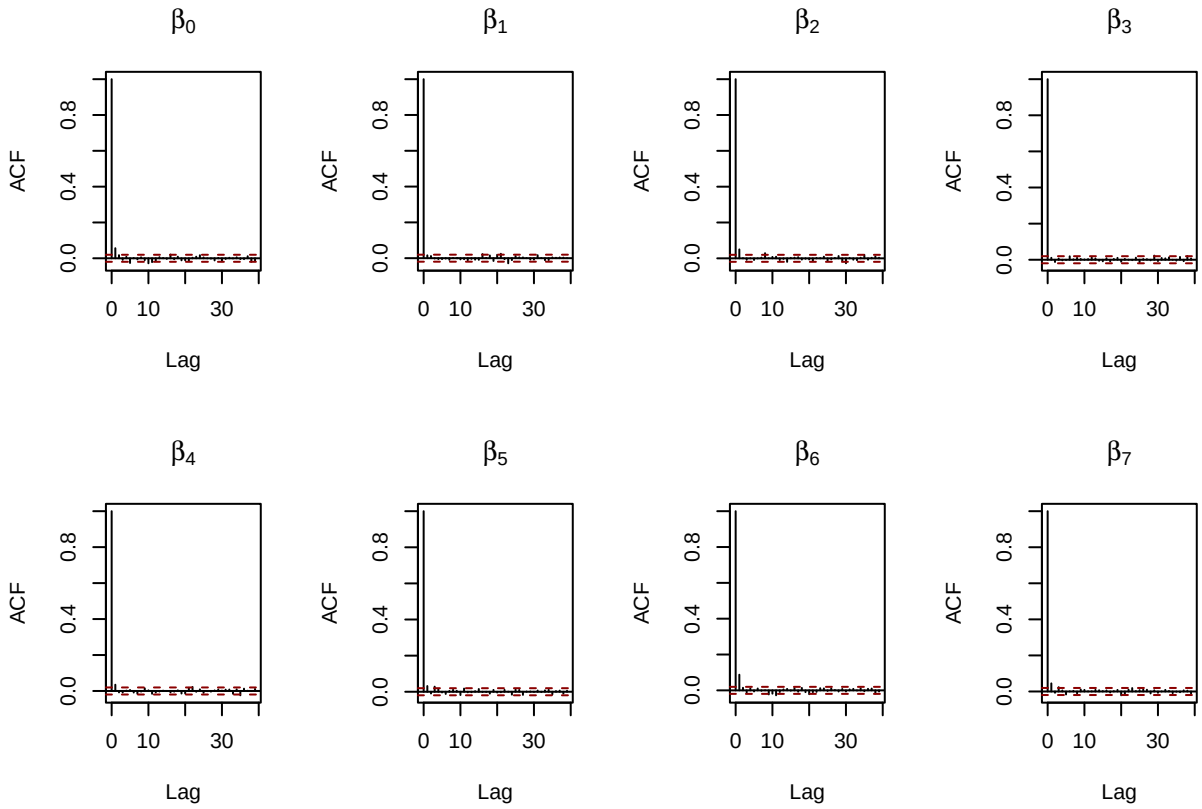
- GRAPHICAL CHECKS

First of all we use the ***trace plots*** of the sampled values $\beta_{\mathbf{i}}^{(\mathbf{s})}$ across the iterations $s = 1, ...S$. The chains should be concentrated around a region of high posterior probability and **no particular trends** should be observed in the chains.


β0
−3
−4
−5
−6
−7
0
2000
4000
6000
8000
10000
s
β1
0.10
0.05
0.00
−0.05
β2
−1.0
−1.5
−2.0
−2.5
β3
β4
0
−1
−2
−3
−4
−5
−6
β5
2.5
2.0
1.5
1.0
β6
0.65
0.55
0.45
0.35
−0.008
−0.012
−0.016

We can't see any particular pattern in these trace plots.

We now use **acf** plots to check for autocorrelations in the chains. In fact, although the MCMC output is a dependent sequence from the posterior, its aim is to simulate random (and independent) draws from the posterior itself.



In this case, we don't see dependencies in any of the chains.

- FORMAL CONVERGENCE DIAGNOSTIC

We are now going to perform some formal tests.

The first one is the **Geweke Test**. The idea behind this test is the following: if the chain has reached convergence, then statistics computed from different portions of the chain should be "similar".

Let us consider two portions of the chain:

- the initial $x\%$: $\beta_{i,I}$, with size $n_{i,I}$
- the final $y\%$: $\beta_{i,L}$, with size $n_{i,L}$

If the chain is stationary, the two corresponding sample means $\bar{\beta}_{i,I}$ and $\bar{\beta}_{i,L}$ should be similar.

The Geweke statistics is:

$$Z_n = \frac{\bar{\beta}_{i,I} - \bar{\beta}_{i,L}}{\sqrt{\hat{s}^2_{i,I} + \hat{s}^2_{i,L}}} \longrightarrow N(0,1) \qquad as \;\; n \to \infty$$

with $n = n_{i,I} + n_{i,L}$ and $\hat{s}^2_{i,I}$, $\hat{s}^2_{i,L}$ the sample variances of $\beta_{i,L}$, $\beta_{i,I}$.

If $|Z_n|$ is "large" the (null) hypothesis of stationarity (equal means) is rejected. Therefore, if we fix $\alpha = 0.05$, with a value of $|Z_n| > 1.96$ we are in the rejection region of the null hypothesis.

```
G_test <- geweke.diag(Beta)
```

	beta0	beta1	beta2	beta3	beta4	beta5	beta6	beta7
Z score	-0.93597	0.36613	-0.79139	-0.18602	0.73866	0.79085	1.03421	-0.98619

Looking at these results, we can state that the Geweke Test indicates that the chains produced by our algorithm actually reached convergence.

We now evaluate the **Effective Sample Size (ESS)**

The idea is to have a sort of ***"exchange rate"*** between dependent and independent samples. You might want to say, for example, that 1,000 samples from a certain Markov chain are worth about as much as 80 independent samples because the MCMC samples are highly correlated. Or you might want to say that 1,000 samples from a different Markov chain are worth about as much as 800 independent samples because although the MCMC samples are dependent, they're weakly correlated.

$$ESS = \frac{G}{1 + 2\sum_{g=1}^{G} acf_g}$$

with G the number of post-burn-in MCMC samples. Notice how, if all acf's are zero (no dependence), then $ESS = G$.

```
## in our case G is equal to:  9980
```

```
ESS <- effectiveSize(Beta)
```

	ESS
beta0	10163.653
beta1	9666.987
beta2	9031.771
beta3	9980.000
beta4	9305.506
beta5	9030.594
beta6	8375.462
beta7	8838.065

From this test we can see that, for every coefficient, the MCMC samples are weakly correlated.

3.3 PREDICTION

In order to evaluate the prediction capabilities of our model, we have proceeded as follows:

At each iteration (s):

- we evaluated $Z_i^{(s)} \sim N(\beta^{\mathbf{T(s)}}\mathbf{x_i},\ 1)$
  - we assigned to the elements of the vector $\hat{Y}^{(s)}$ the value 0 whenever $z_i^{(s)} > 0$
  - we assigned to the elements of the vector $\hat{Y}^{(s)}$ the value 1 whenever $z_i^{(s)} < 0$

At the end of this process, we obtained a matrix containing n vectors of *Lastit* elements, where each vector represents the possible values for each predicted response $\hat{Y}_i$.

We decided to take the mode of each vector as a value for $\hat{Y}_i$.

We then compare the predicted values with the observed values by looking at the percentage of 0's and 1's with respect to the total in both cases.

```r
mode_ <- apply(Y_hat, 2, mode)
length(mode_)
```

```
## [1] 1599
```

```r
Pr_0 <- 1/n*length(which(data1$quality==0))
Pr_0_hat <- 1/n*length(which(mode_==0))

Pr_1 <- 1/n*length(which(data1$quality==1))
Pr_1_hat <- 1/n*length(which(mode_==1))
```

```
## the percentage of observed responses with value 1 is  0.46529
```

```
## the percentage of predicted responses with value 1 is  0.4828
```

```
## the percentage of observed responses with value 1 is  0.53471
```

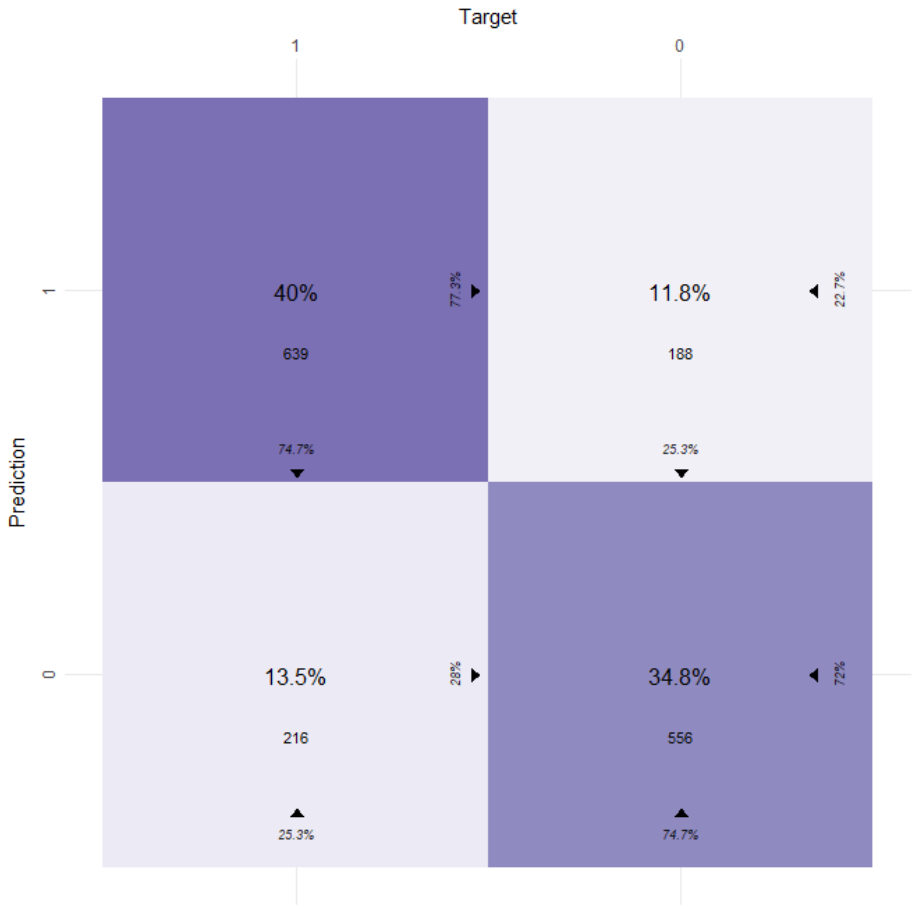
```
## the percentage of predicted responses with value 1 is  0.5172
```

When inspecting a classification model's performance, a **confusion matrix** tells you the distribution of the predictions and targets.

If we have two classes (0, 1), we have these 4 possible combinations of predictions and targets:

Target	Prediction	Called
0	0	True Negative
0	1	False Positive
1	0	False Negative
1	1	True Positive

For each combination, we can count how many times the model made that prediction for an observation with that target.



In the middle of each tile, we have the normalized count (overall percentage) and, beneath it, the count.

At the bottom, we have the column percentage. Of all the observations where Target is 1, 74.7% of them were predicted to be 1 and 25.3% 0.

At the right side of each tile, we have the row percentage. Of all the observations where Prediction is 1, 77.3% of them were actually 1, while 22.7% were 0.

Note that the color intensity is based on the counts

```
d_binomial <- tibble("target" = y,
                     "prediction" = mode_)

eval <- evaluate(d_binomial,
                 target_col = "target",
                 prediction_cols = "prediction",
                 type = "binomial")
```

Balanced Accuracy	Accuracy	F1	Sensitivity	Specificity
0.748597	0.7485929	0.7609988	0.748538	0.7486559

Pos Pred Value	Neg Pred Value	AUC	Lower CI	Upper CI
0.7738815	0.7215026	0.748597	0.7272688	0.7699252