Decision Trees

Ivet A., Laura A., Arnau G.

1 Introduction

File `HFCRD.csv` contains the *Heart Failure Clinical Records Dataset* with information on 299 patients with advanced heart failure in whom the following variables were analyzed:

- `age`: age of the patient (years)
- `anaemia`: decrease of red blood cells or hemoglobin (boolean)
- `creatinine_phosphokinase`: level of the CPK enzyme in the blood (mcg/L)
- `diabetes`: if the patient has diabetes (boolean)
- `ejection_fraction`: ejection fraction: percentage of blood leaving the heart at each contraction
- `high_blood_pressure`: if the patient has hypertension (boolean)
- `platelets`: platelets in the blood (kiloplatelets/mL)
- `serum_creatinine`: level of serum creatinine in the blood (mg/dL)
- `serum_sodium`: level of serum sodium in the blood (mEq/L)
- `sex`: female/male (binary)
- `smoking`: if the patient smokes or not (boolean)
- `time`: follow-up period (days)
- `DEATH_EVENT`: the patient deceased during the follow-up period (boolean)

Let's load the data:

```r
data <- read.csv("HFCRD.csv")
```

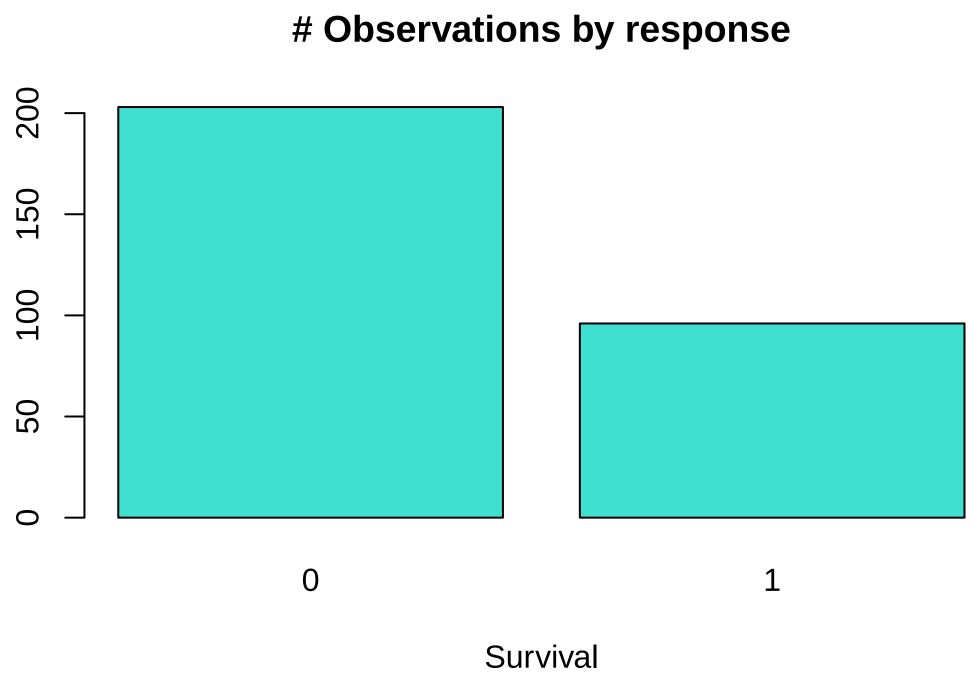
2 Exploratory data analysis

First of all, notice that many variables are described as *boolean*, but they are stored as *numerical* in the data frame. So let's save as factors the variables which should be treated as that:

```r
# Features
hfcrd <- data
hfcrd$anaemia <- as.factor(hfcrd$anaemia)
hfcrd$diabetes <- as.factor(hfcrd$diabetes)
hfcrd$high_blood_pressure <- as.factor(hfcrd$high_blood_pressure)
hfcrd$sex <- as.factor(hfcrd$sex)
hfcrd$smoking <- as.factor(hfcrd$smoking)
# Target
hfcrd$DEATH_EVENT <- as.factor(hfcrd$DEATH_EVENT)
```

Let's see how the distribution of the target is:

```r
survival_table <- table(hfcrd$DEATH_EVENT)
barplot(survival_table,
        main = "# Observations by response",
        xlab = "Survival",
        col = "turquoise")
```



It seems we have available much more data (twice) about individuals who ended up surviving after the study than from the ones who ended up dying. This could difficult the prediction of the latter cases.

The next chunk displays a quite complete description of the features:

```r
library(skimr)
# skim(hfcrd[, -13], .data_name = "HFCRD")
```

We can wee the absence of missing values in the data, which is nice. The other factor variables also seem to be quite unbalanced, but we hope that this fact won't affect negatively the training process.

Numerical variables, in general, do not show a symmetric distribution either.

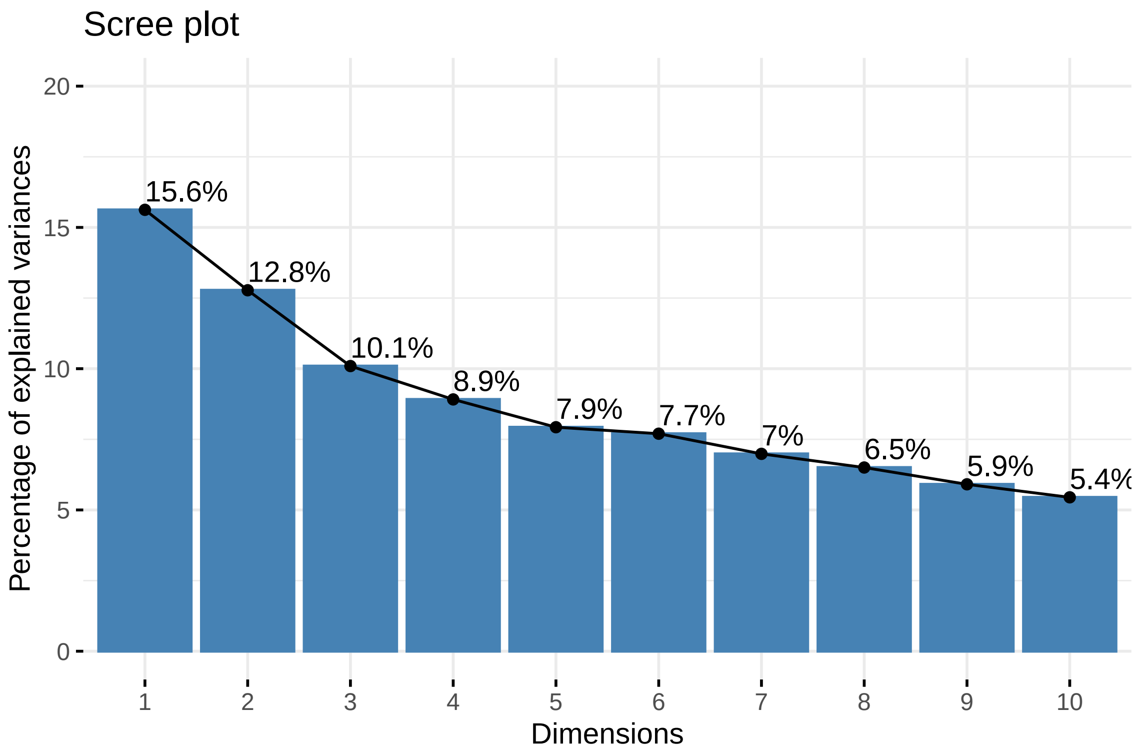
Another analysis that's often considered when working with data sets is a PCA. In this case we do not have a large amount of variables, so it is not a *mandatory* technique. However, it could be interesting to check its results.

Notice that we use the original data set `data` because the `prcomp()` needs the data to be of type numeric (instead of factor).

```r
hfcrd_PCA <- prcomp(data, scale = TRUE)
```

The following function will plot the variance explained by each component:

```r
library(factoextra)
fviz_screeplot(hfcrd_PCA, addlabels = TRUE, ylim = c(0, 20))
```

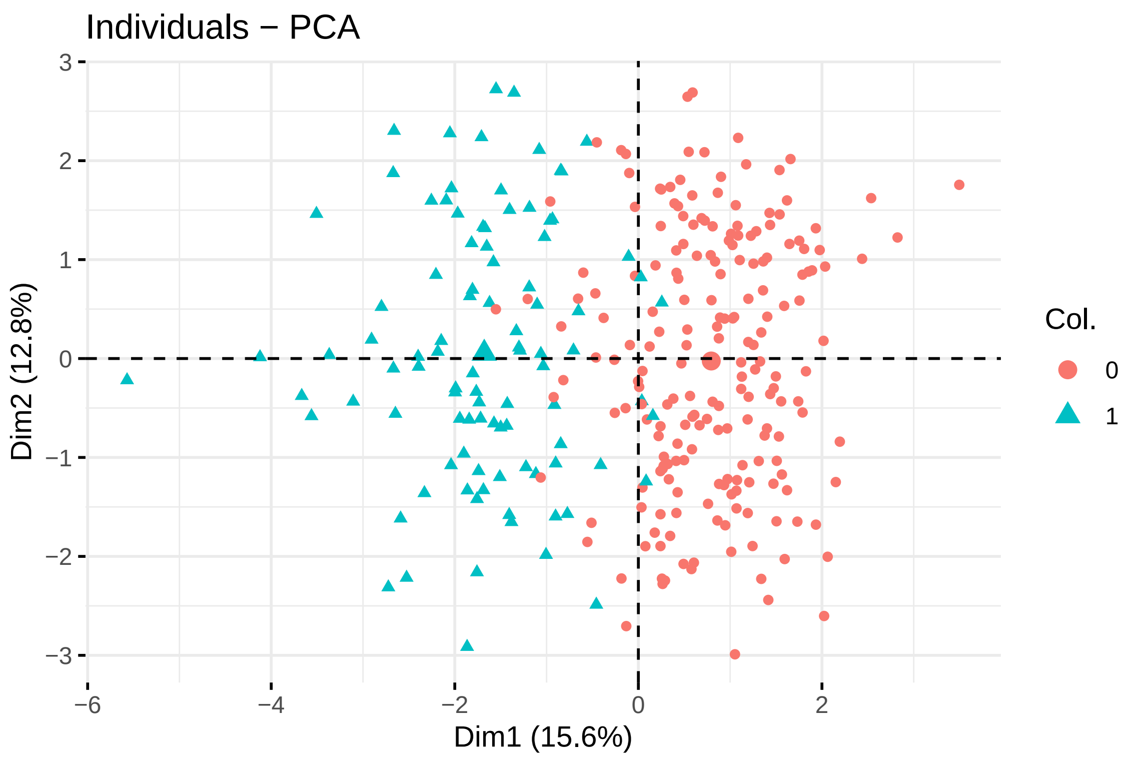


By looking at the previous plot we can notice how important and necessary are all the variables for our analysis, since all of them are needed in order to explain the variability of the data.

With 2 components, less than a 30% of the data is explained... So we won't discard any feature.

One last interesting plot is the following, which shows the distribution of the first and second components by each target level:

```r
fviz_pca_ind(hfcrd_PCA, geom.ind = "point",
             col.ind = hfcrd$DEATH_EVENT,
             axes = c(1, 2),
             pointsize = 1.5)
```



The first dimension seems responsible of capturing the trend of the target variable. This fact sounds good to us because it could mean that `DEATH_EVENT` is an important value related to the rest of variables.

3 Data preparation

Before moving into the fitting of any model, data should be prepared. As scaling is not necessary when working with trees, the only step we are going to consider is splitting the data into 2 sets: a training set (2/3) and a test set (1/3).

```r
set.seed(1234)
n <- nrow(hfcrd)
train_prop <- 2/3
train_ind <- sample(1:n, train_prop * n)
hfcrd_train <- hfcrd[train_ind, ]
hfcrd_test <- hfcrd[-train_ind, ]
survival_test <- hfcrd[-train_ind, "DEATH_EVENT"]
```

4 Model fitting

In this section we are going to fit the following statistical models:

1. A classification tree to predict survival.
2. A logistic classifiers with the same goals and assessment.
3. A third predictive model of our choice: a Bayesian Logistic Classifier.

For each one of them we'll do the required tuning, if needed, train the model and do a proper test-based evaluation.

4.1 Classification tree

The first model we are going to fit is a *Classification Tree*. We will use the training set, `hfcrd_train`, to try to predict the `DEATH_EVENT` using all the rest of variables.

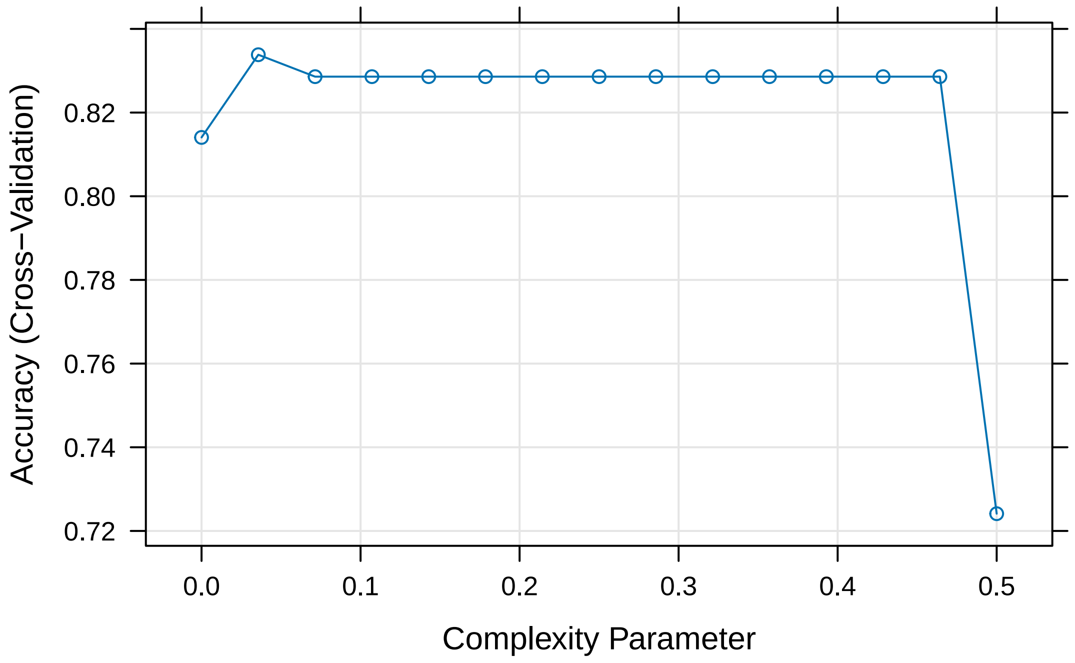
We choose to use the `caret` library, which can be used to fit this type of models by specifying the argument `method = "rpart"` in the `train` method. By default, this uses the bootstrap to choose the optimal value for the hyperparameter α (which controls the the trade-off between tree complexity and accuracy). We prefer to use *Cross-Validation* instead, with 10 folds. In additions, the argument `tuneLength = 15` is added in order to choose between 15 different values of α.

```r
set.seed(12345)
library(caret)
tree <- train(DEATH_EVENT ~ .,
              data = hfcrd_train,
              method = "rpart",
              tuneLength = 15,
              trControl = trainControl(method = "cv", number = 10))
tree
```

```
## CART
##
## 199 samples
##  12 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 179, 178, 178, 179, 179, 179, ...
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.00000000  0.8140602  0.5694693
##   0.03571429  0.8338221  0.6130375
##   0.07142857  0.8285840  0.5681356
##   0.10714286  0.8285840  0.5681356
##   0.14285714  0.8285840  0.5681356
##   0.17857143  0.8285840  0.5681356
##   0.21428571  0.8285840  0.5681356
##   0.25000000  0.8285840  0.5681356
##   0.28571429  0.8285840  0.5681356
##   0.32142857  0.8285840  0.5681356
##   0.35714286  0.8285840  0.5681356
##   0.39285714  0.8285840  0.5681356
##   0.42857143  0.8285840  0.5681356
##   0.46428571  0.8285840  0.5681356
##   0.50000000  0.7241228  0.2109943
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.03571429.
```

Let's see the accuracy obtained for each value of α:

```
plot(tree)
```



Most of the values considered result in the same accuracy. We can see, however, the presence of an absolute maximum near 0:

```
max_acc_ind <- which.max(tree$results[, "Accuracy"])
(alpha_opt <- tree$results[max_acc_ind, "cp"])
```

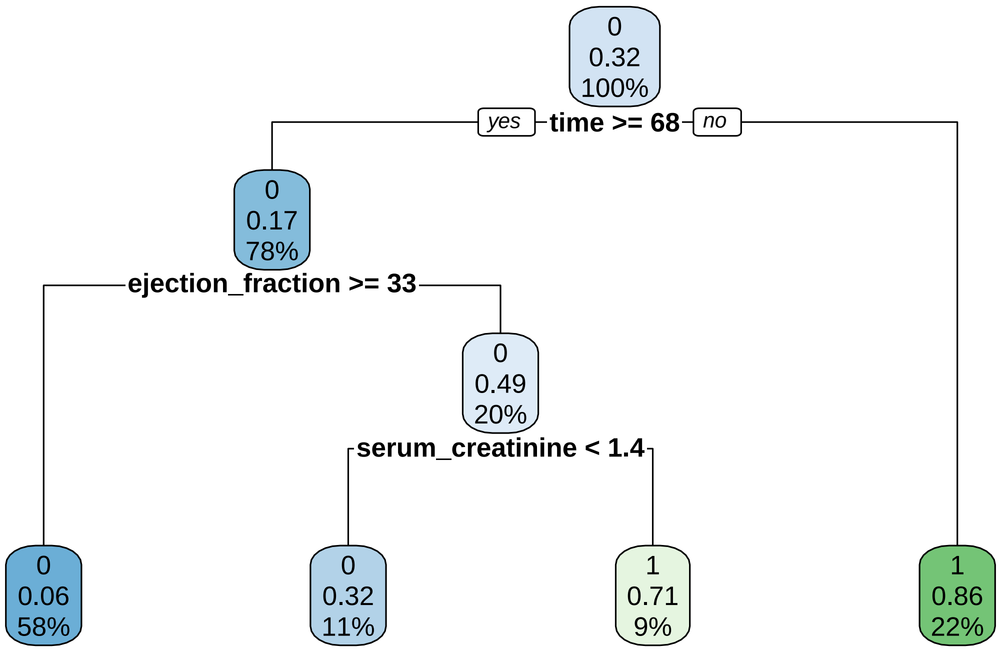
```
## [1] 0.03571429
```

```
(acc_opt <- tree$results[max_acc_ind, "Accuracy"])
```

```
## [1] 0.8338221
```

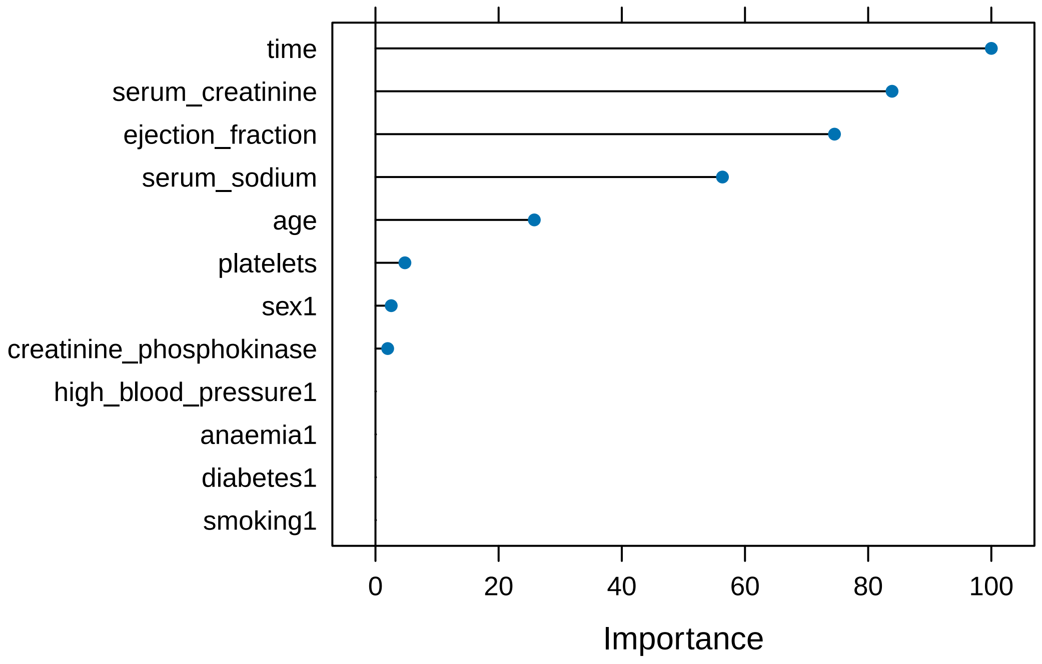
A value of $\alpha = 0.036$ gives the maximum accuracy achieved by the tree, which is **0.834**. The `finalModel` attribute saved in the `tree` object stored the tree corresponding to these optimal values:

```
library(rpart.plot)
rpart.plot(tree$finalModel, cex = 0.8)
```



By looking at the tree we can realize how only 3 variables seem to be relevant in order to determine survival. Moreover, the `varImp()` function can be used to see with more detailed the importance of each feature:

```r
plot(varImp(tree))
```



Actually, more variables are relevant for the response target. However, they might contain too little information and so considering them might lead the model to overfit.

Let's conclude the analysis of this model by checking its performance with the test set:

```r
pred_test <- predict(tree, newdata = hfcrd_test)
(confMat <- confusionMatrix(pred_test, survival_test))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 61  7
##          1  7 25
##
##                Accuracy : 0.86
##                  95% CI : (0.7763, 0.9213)
##     No Information Rate : 0.68
##     P-Value [Acc > NIR] : 3.065e-05
##
##                   Kappa : 0.6783
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.8971
##             Specificity : 0.7812
##          Pos Pred Value : 0.8971
##          Neg Pred Value : 0.7812
##              Prevalence : 0.6800
##          Detection Rate : 0.6100
##    Detection Prevalence : 0.6800
##       Balanced Accuracy : 0.8392
##
##        'Positive' Class : 0
##
```

The model achieves an accuracy of **0.86**, which is slightly higher than the one obtained for the training set! These are very good results, because it implies that the tree generalizes well for new data.

It is also nice to notice that these results are reached by taking into account the data of only 3 variables. This implies that applying this model in the future might be easy because only few information of the individuals will be needed in order to predict whether they are going to survive or not.

4.2 Logistic classifier

Now we are going to fit a logistic regression model. To identify a good subset of predictors for this problem we are going to run best subsets with the AIC criteria. This can be easily done using the library `glmulti`.

```r
library(glmulti)
mod_aic <- glmulti(DEATH_EVENT ~ .,
                   data = hfcrd_train,
                   family = binomial,
                   level = 1,
                   crit = "aic",
                   plotty = FALSE,
                   report = FALSE)
summary(mod_aic)$bestmodel
```

```
## [1] "DEATH_EVENT ~ 1 + diabetes + age + ejection_fraction + serum_creatinine + "
## [2] "    serum_sodium + time"
```

Let us notice that the variables that have been chosen as useful are `diabetes`, `age`, `ejection_fraction`. `serum_creatinine`, `serum_sodium` and `time`.

The coefficients of this model are:

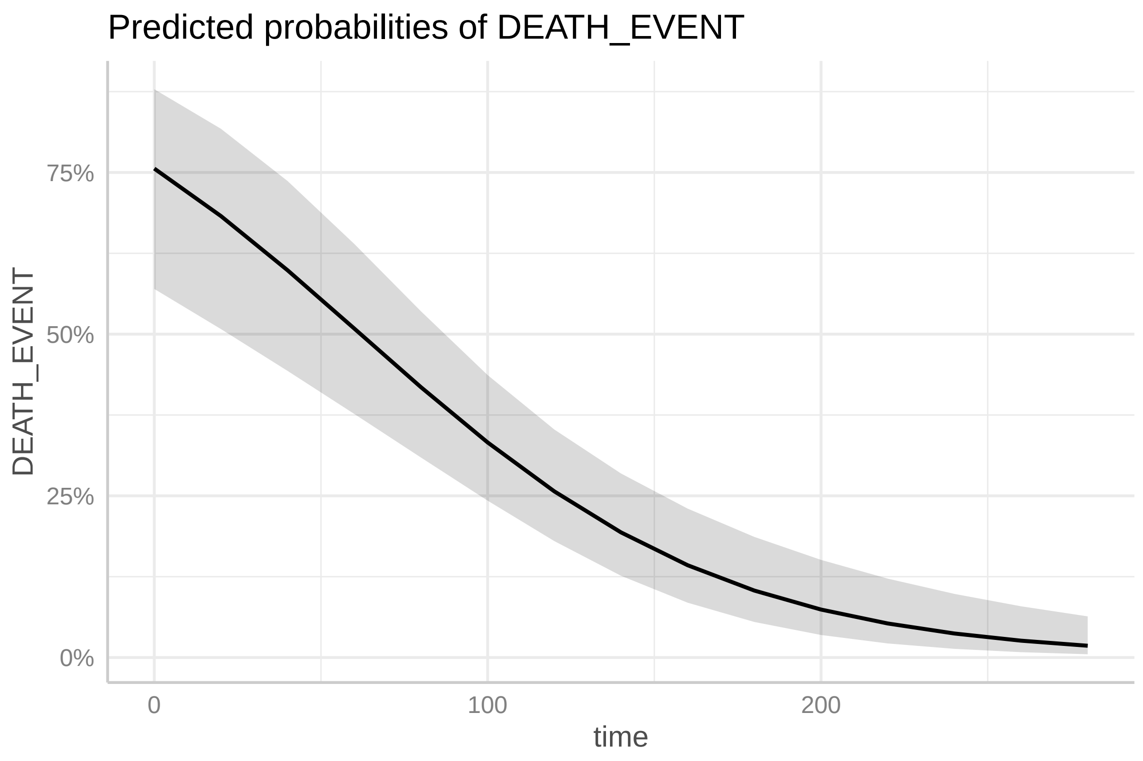
```
mod_log <- mod_aic@objects[[1]]
summary(mod_log)
```

```
##
## Call:
## fitfunc(formula = as.formula(x), family = ..1, data = data)
##
## Coefficients:
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)       25.790092   8.089558   3.188 0.001432 **
## diabetes1          0.631873   0.436598   1.447 0.147823
## age                0.038148   0.019574   1.949 0.051304 .
## ejection_fraction -0.076952   0.020255  -3.799 0.000145 ***
## serum_creatinine   0.775714   0.219038   3.541 0.000398 ***
## serum_sodium      -0.185929   0.056941  -3.265 0.001093 **
## time              -0.018277   0.003547  -5.153 2.56e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 249.97  on 198  degrees of freedom
## Residual deviance: 140.48  on 192  degrees of freedom
## AIC: 154.48
##
## Number of Fisher Scoring iterations: 6
```

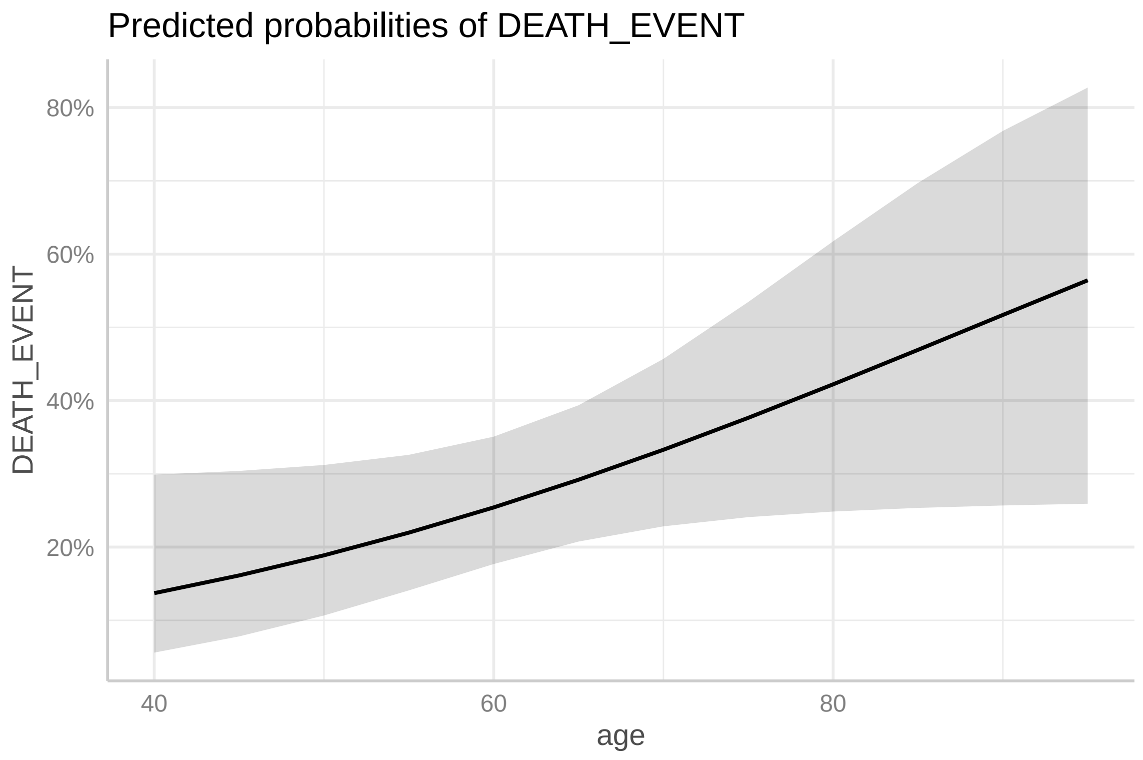
Let us observe that as the variables `age` and `serum_creatinine` increase, the outcome increases. In this case, this means that the prediction gets closer to 1, that is the death of the patient. In return, as the variables with negative coefficients (`ejection_fraction`, `serum_sodium` and `time`) increase, the prediction decreases (survival of the patient).

To see more clearly the effect that some of these variables have on the outcome, we can use an interaction plot. For example, we are going to see how `age` and `time` affect the prediction.

```
library(ggeffects)
par(mfrow=c(1,2))
plot1 <- ggemmeans(mod_log, terms = "time")
plot(plot1)
```



```r
plot2 <- ggemmeans(mod_log, terms = "age")
plot(plot2)
```



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

As we have already said, time has a negative effect on the response variable and age has a positive one.

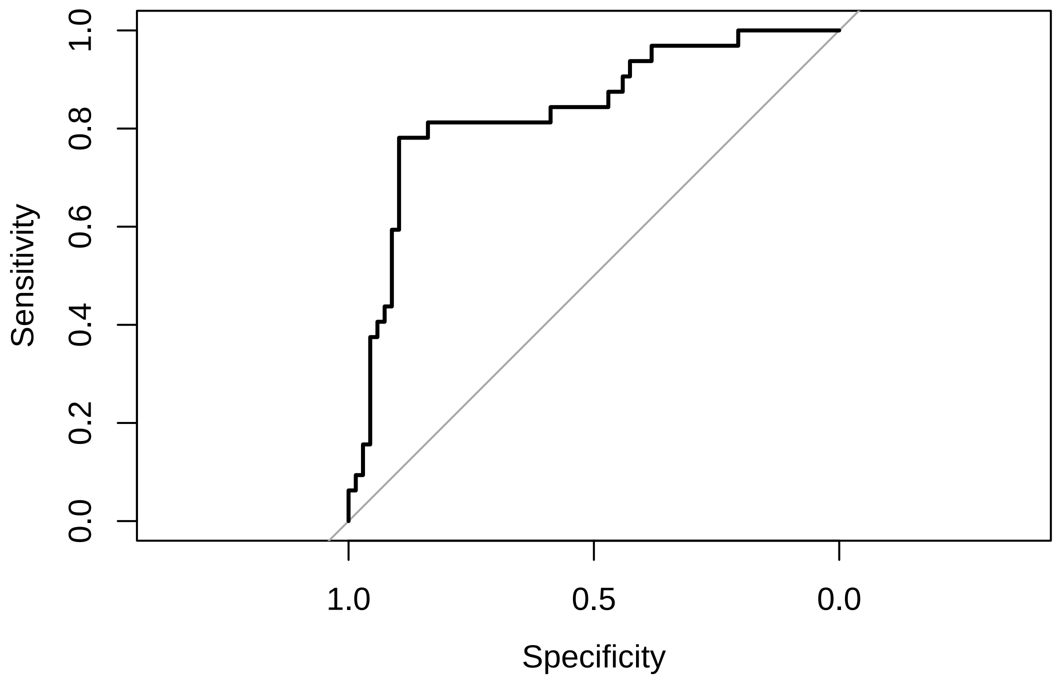
We can use this model to try to predict the target variable for the test set. Since the predictions are continuous and not binary, we will need to define a threshold to partition which predictions are going to be considered 1 or 0. For now, we will use 0.5 as the threshold. With these predictions we can compute the confusion matrix.

```r
pred_prob <- predict(mod_log, type = "response", newdata = hfcrd_test)
pred05 <- as.factor(1 * (pred_prob > 0.5))
(confMat_log <- confusionMatrix(pred05, survival_test))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 61 11
##          1  7 21
##
##                Accuracy : 0.82
##                  95% CI : (0.7305, 0.8897)
##     No Information Rate : 0.68
##     P-Value [Acc > NIR] : 0.001247
##
##                   Kappa : 0.5722
##
##  Mcnemar's Test P-Value : 0.479500
##
##             Sensitivity : 0.8971
##             Specificity : 0.6562
##          Pos Pred Value : 0.8472
##          Neg Pred Value : 0.7500
##              Prevalence : 0.6800
##          Detection Rate : 0.6100
##    Detection Prevalence : 0.7200
##       Balanced Accuracy : 0.7767
##
##        'Positive' Class : 0
##
```

The accuracy of the model using this threshold is **0.82**, so the model does a pretty good job with new data. However, 0.5 may not be the best threshold we can define for this problem. In order to find it, we can use the ROC curve. Let us plot it.

```r
library(pROC)
roc_aic <- roc(DEATH_EVENT ~ pred_prob, data = hfcrd_test)
plot(roc_aic)
```



Now, we can easily find the best threshold as follows.

```
best_aic <- coords(roc_aic,
                   x = "best",
                   best.method = "closest.topleft")
best_aic[[1]]
```

```
## [1] 0.4621655
```

Finally, using 0.4621655 as our new threshold for the predictions, we obtain a confusion matrix with an improved accuracy.

```
pred_threshold <- as.factor(1 * (pred_prob > best_aic[[1]]))
(confMat_log2 <- confusionMatrix(pred_threshold, survival_test))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 61  7
##          1  7 25
##
##                Accuracy : 0.86
##                  95% CI : (0.7763, 0.9213)
##     No Information Rate : 0.68
##     P-Value [Acc > NIR] : 3.065e-05
##
##                   Kappa : 0.6783
##
##  Mcnemar's Test P-Value : 1
```

```
##
##             Sensitivity : 0.8971
##             Specificity : 0.7812
##          Pos Pred Value : 0.8971
##          Neg Pred Value : 0.7812
##              Prevalence : 0.6800
##          Detection Rate : 0.6100
##    Detection Prevalence : 0.6800
##       Balanced Accuracy : 0.8392
##
##        'Positive' Class : 0
##
```

Thus, this model makes predictions with a **0.86** accuracy on the test set, which is a pretty good performance. Still, we should take into account that it needs 6 variables in order to do so.

4.3 Bayesian logistic classifier

In this section we will be fitting a Bayesian logistic regression model. We have been following the chapters 21 and 22 of the Kruschke's book (which can be downloaded in (https://nyu-cdsc.github.io/learningr/assets/kruschke_bayesian_in_R.pdf)). Moreover, we have been following the github blog (https://avehtari.github.io/modelselection/diabetes.html), where there is an example on how to fit a Bayesian logistic regression model using the `rstanarm` package.

We will be using a Bayesian logistic regression model. It is, we assume $y_i \sim Bern(\mu)$ where $\mu = logistic(\beta_0 + \sum_{i=1}^{p} \beta_i X_i)$ being X_i the covariates. And we will be taking non-informative prior distributions.

We charge the required libraries in this section:

```r
library(rstan)
library(bayesplot)
library(rstanarm)
library(ggplot2)
```

We will be using the `rstanarm` package which is an `R` package that emulates other `R` model-fitting functions but uses Stan via the `rstan` package. See information about the `rstanarm` package on (https://mc-stan.org/rstanarm/).

The next Bayesian logistic regression model is fitted via Stan by means of the MCMC method. By default the software use non-informative flat priors, we will be using these priors.

```r
post <- stan_glm(DEATH_EVENT~., data = hfcrd_train,
                 family = binomial(link = "logit"),
                 seed = 1234)
```

```
##
## SAMPLING FOR MODEL 'bernoulli' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.000524 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 5.24 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
```

```
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.061 seconds (Warm-up)
## Chain 1:                0.069 seconds (Sampling)
## Chain 1:                0.13 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'bernoulli' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 9e-06 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.09 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.063 seconds (Warm-up)
## Chain 2:                0.059 seconds (Sampling)
## Chain 2:                0.122 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'bernoulli' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 8e-06 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.08 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.059 seconds (Warm-up)
## Chain 3:                0.063 seconds (Sampling)
## Chain 3:                0.122 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'bernoulli' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 8e-06 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.08 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.062 seconds (Warm-up)
## Chain 4:                0.075 seconds (Sampling)
## Chain 4:                0.137 seconds (Total)
## Chain 4:
```

We should check for the well-convergence of the MCMC method. The reader can run the following code if they are interested on seeing a nice shiny app where the different features related with the convergence of the MCMC method are greatly displayed. Here we can see that the convergence has been reached.

```
# launch_shinystan(post, ppd = FALSE)
```

In the previous shiny app there are also features related with the fitted model, the posterior distributions obtained, etc. It is really nice.

Let us print the `rstanarm` object firstly. We use the `summary` method. In the summary the reader can see some summary statistics of the posterior distributions and some quantities indicating wether the MCMC method has converged or not.

```r
summary(post)
```

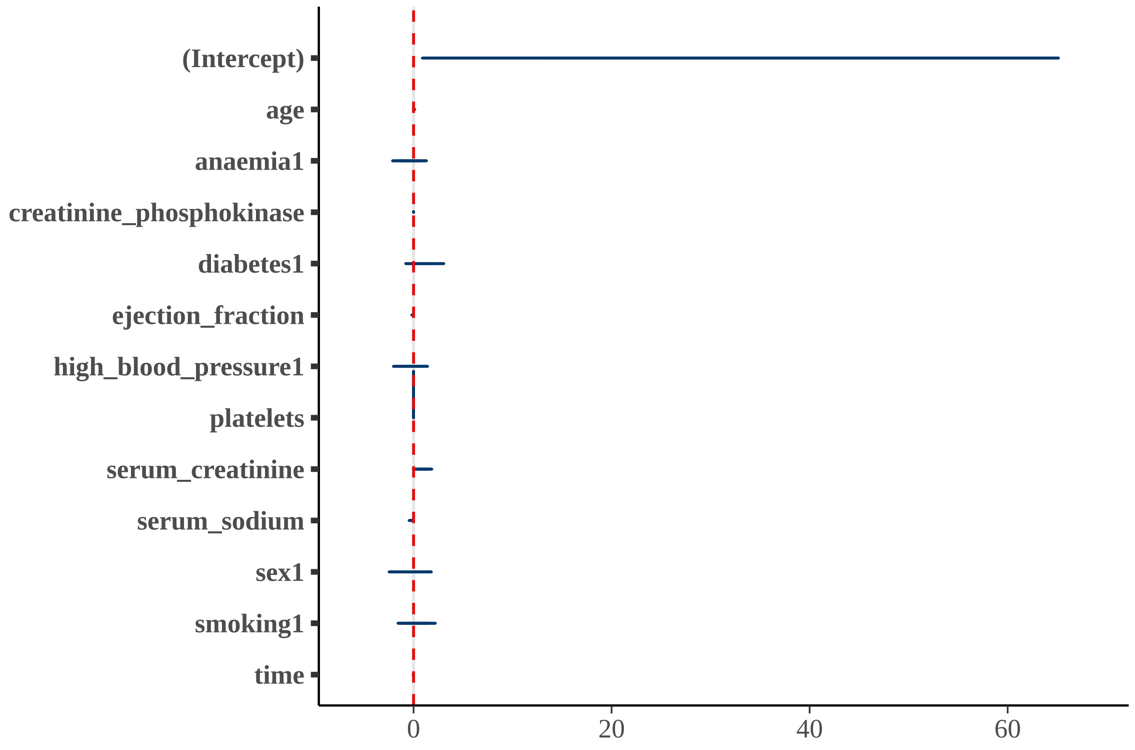
```
##
## Model Info:
##  function:     stan_glm
##  family:       binomial [logit]
##  formula:      DEATH_EVENT ~ .
##  algorithm:    sampling
##  sample:       4000 (posterior sample size)
##  priors:       see help('prior_summary')
##  observations: 199
##  predictors:   13
##
## Estimates:
##                            mean   sd   10%   50%   90%
## (Intercept)              29.7    8.7 18.6  29.5  40.6
## age                       0.0    0.0  0.0   0.0   0.1
## anaemia1                 -0.4    0.5 -1.1  -0.4   0.2
## creatinine_phosphokinase  0.0    0.0  0.0   0.0   0.0
## diabetes1                 0.8    0.5  0.2   0.8   1.4
## ejection_fraction        -0.1    0.0 -0.1  -0.1  -0.1
## high_blood_pressure1     -0.2    0.5 -0.8  -0.2   0.4
## platelets                 0.0    0.0  0.0   0.0   0.0
## serum_creatinine          0.9    0.3  0.6   0.9   1.2
## serum_sodium             -0.2    0.1 -0.3  -0.2  -0.1
## sex1                     -0.4    0.6 -1.1  -0.4   0.4
## smoking1                  0.3    0.6 -0.4   0.3   1.0
## time                      0.0    0.0  0.0   0.0   0.0
##
## Fit Diagnostics:
##            mean   sd   10%   50%   90%
## mean_PPD 0.3    0.0  0.3   0.3   0.4
##
## The mean_ppd is the sample average posterior predictive distribution of the outcome variable (for de
##
## MCMC diagnostics
##                          mcse Rhat n_eff
## (Intercept)              0.1  1.0  4206
## age                      0.0  1.0  3786
## anaemia1                 0.0  1.0  4386
## creatinine_phosphokinase 0.0  1.0  4220
## diabetes1                0.0  1.0  4442
## ejection_fraction        0.0  1.0  3310
## high_blood_pressure1     0.0  1.0  4574
## platelets                0.0  1.0  5163
## serum_creatinine         0.0  1.0  3921
## serum_sodium             0.0  1.0  4410
## sex1                     0.0  1.0  3517
## smoking1                 0.0  1.0  4133
## time                     0.0  1.0  3543
## mean_PPD                 0.0  1.0  4811
## log-posterior            0.1  1.0  1740
##
```

```
## For each parameter, mcse is Monte Carlo standard error, n_eff is a crude measure of effective sample
```

As we can see, all the Rhat quantities are 1, which indicates that the convergence has been reached. Moreover, the n_eff obtained indicates good convergence as well, because they are big enough (as a rule of thumb, we expect these numbers to be greater to 600 to have good convergence).

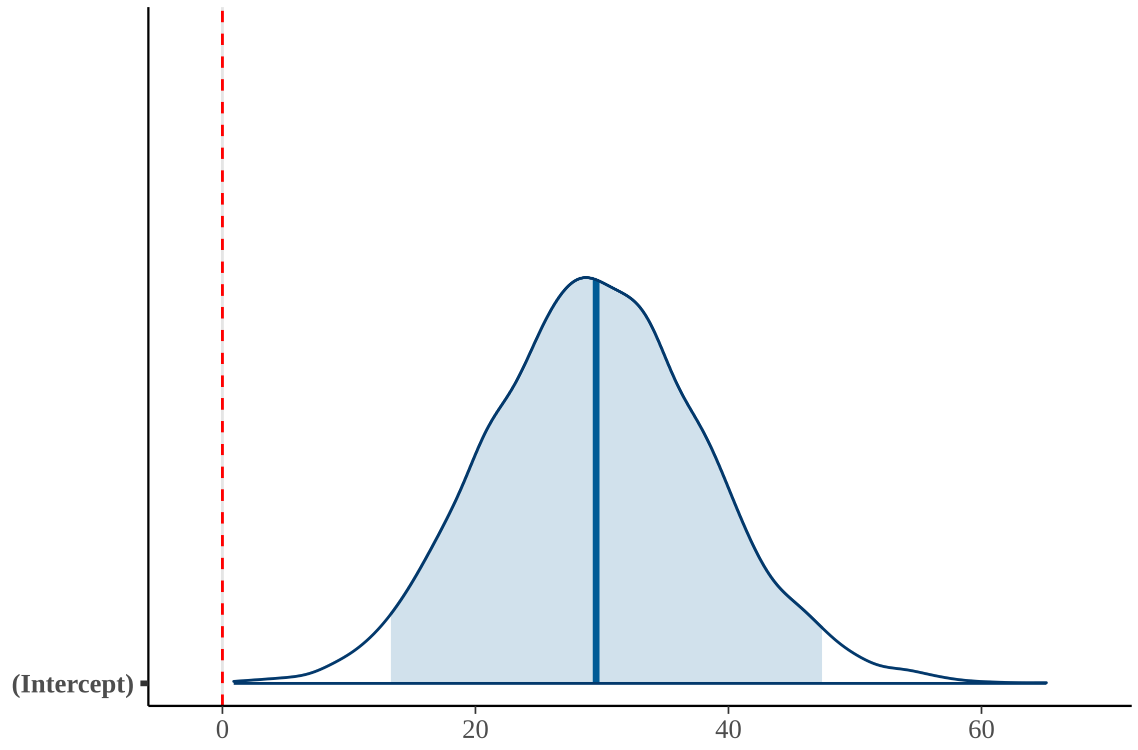
Let us plot the posterior distributions of the parameters:

```r
plot(post, "areas", prob = 0.95, prob_outer = 1) +
  geom_vline(xintercept = 0, color = "red", lty = 2)
```

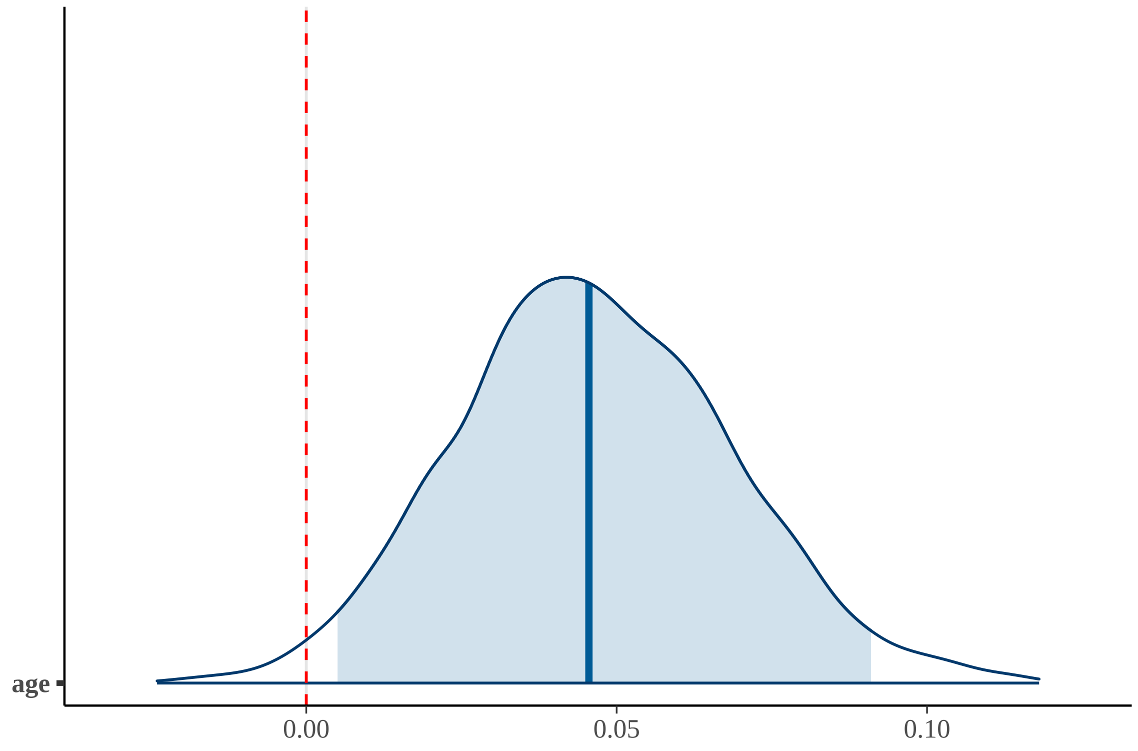


Here we can not visualize as we want the curve of the posterior distributions, so let us plot apart some of these curves. We choose to plot those curves which posterior distribution seems to put few probability on the 0. Notice that, in order to check the significance of the parameters in our model, we should check whether there is a lot of probability around the 0 in the posterior distributions obtained.

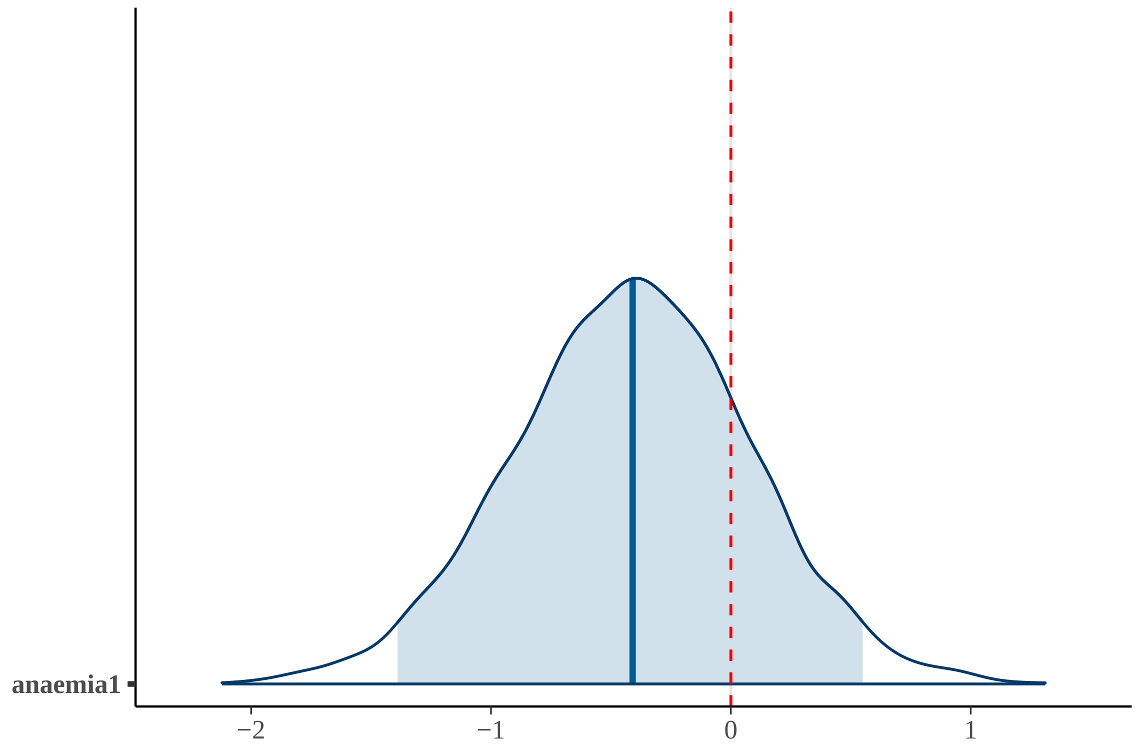
```r
param_names <- names(coef(post))
plot(post, "areas", prob = 0.95, prob_outer = 1, pars = param_names[1]) +
  geom_vline(xintercept = 0, color = "red", lty = 2)
```



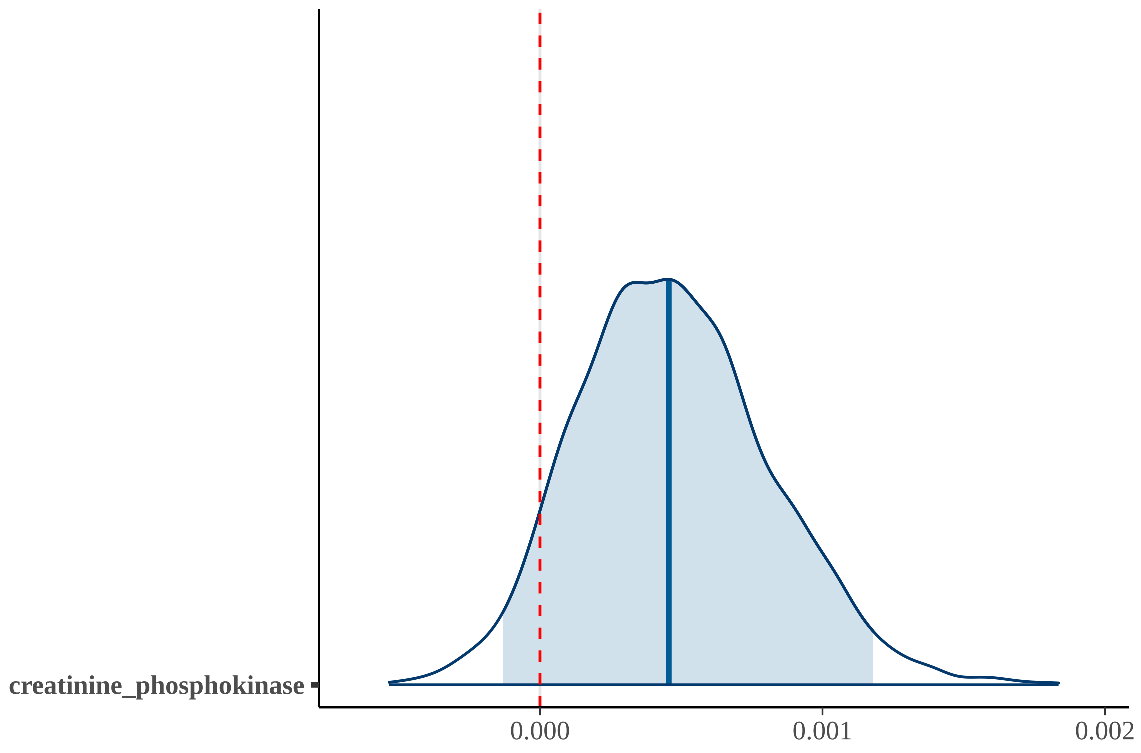
```r
plot(post, "areas", prob = 0.95, prob_outer = 1, pars = param_names[2]) +
  geom_vline(xintercept = 0, color = "red", lty = 2)
```



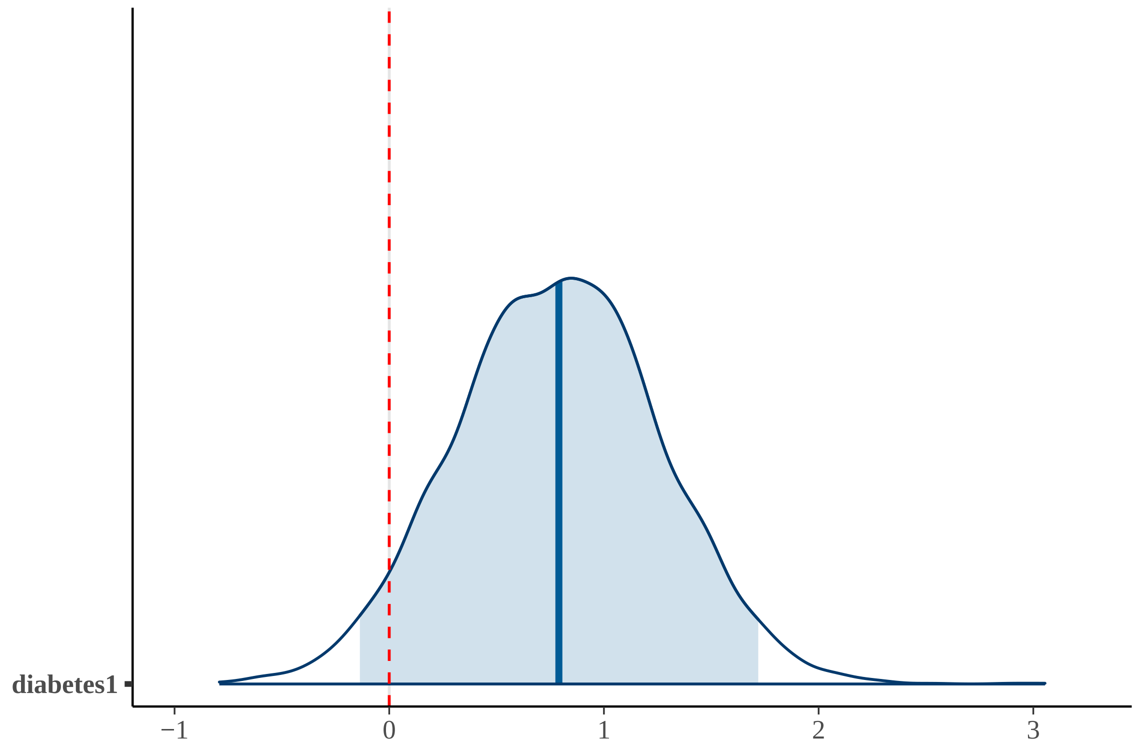
```r
plot(post, "areas", prob = 0.95, prob_outer = 1, pars = param_names[3]) +
  geom_vline(xintercept = 0, color = "red", lty = 2)
```



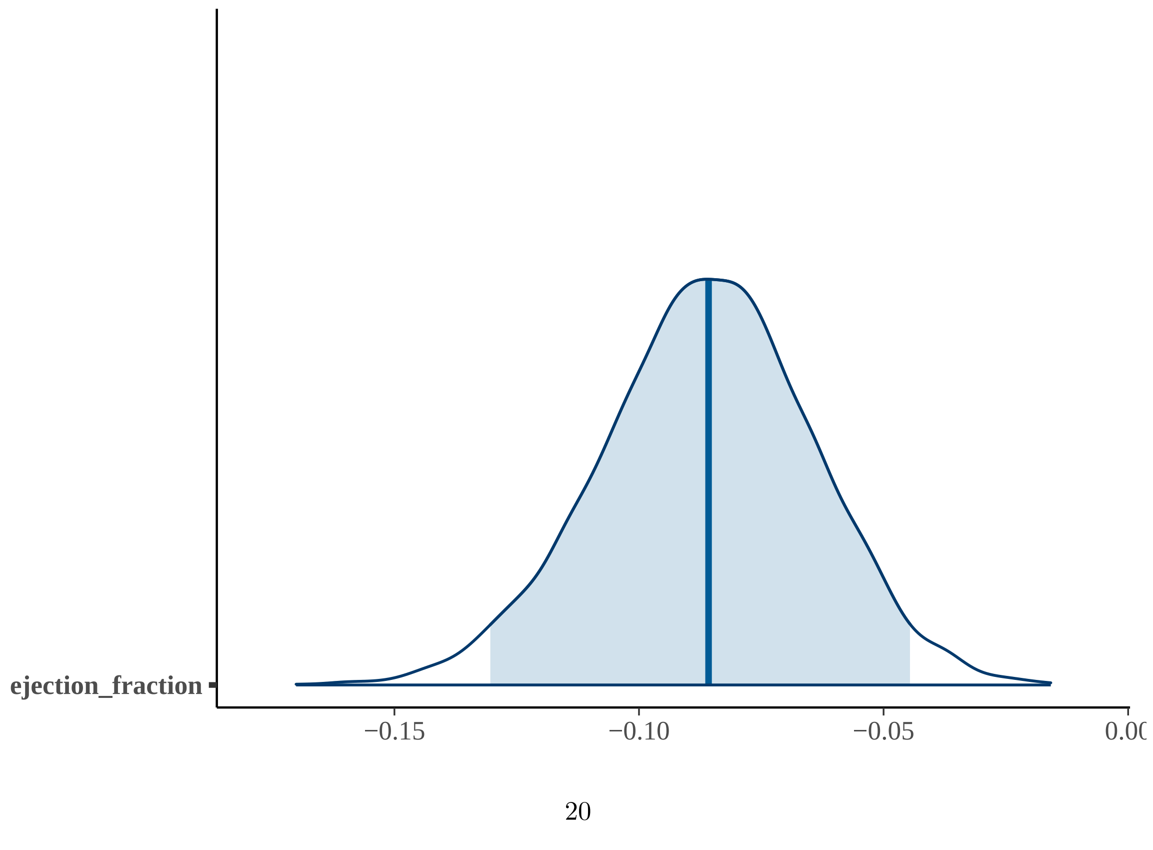
```r
plot(post, "areas", prob = 0.95, prob_outer = 1, pars = param_names[4]) +
  geom_vline(xintercept = 0, color = "red", lty = 2)
```



```r
plot(post, "areas", prob = 0.95, prob_outer = 1, pars = param_names[5]) +
  geom_vline(xintercept = 0, color = "red", lty = 2)
```



```r
plot(post, "areas", prob = 0.95, prob_outer = 1, pars = param_names[6]) +
  geom_vline(xintercept = 0, color = "red", lty = 2)
```



And, as we have painted with the blue zone the 95% credibility interval, we have that those posteriors not including the 0 within the blue zone are significant at a 95% level (there is a 95% of that the 0 is not the parameter value). We have seen this graphically, let us see it now in a more compact way by means of 95% credibility intervals:

```
round(posterior_interval(post, prob = 0.95), 3)
```

```
##                              2.5%  97.5%
## (Intercept)                13.314 47.397
## age                         0.005  0.091
## anaemia1                   -1.389  0.550
## creatinine_phosphokinase    0.000  0.001
## diabetes1                  -0.137  1.719
## ejection_fraction          -0.130 -0.045
## high_blood_pressure1       -1.171  0.684
## platelets                   0.000  0.000
## serum_creatinine            0.412  1.458
## serum_sodium               -0.332 -0.097
## sex1                       -1.535  0.712
## smoking1                   -0.767  1.401
## time                       -0.029 -0.013
```

As we can see, the 0 is included in the 95% credibility interval of `anaemia1`, `creatinine_phosphokinase`, `diabetes1`, `high_blood_pressure1`, `platelets`, `sex1`, `smoking1`. Thus, these are not significant covariates. However, the rest of the covariates they are.

Now, it is time to see how good is our Bayesian model classifying and predicting. We compute posterior predictive probabilities and use them to compute classification error. We first compute the posterior predictive probabilities using the `posterior_epred()` function, then we compute an estimate of this posterior predictive distributions, we have chosen the mean, and then we have the probabilities for each observation in the training sample to be death event or not. We use a 0.5 threshold to classify these probabilities. And finally we compare the obtained predictions with the real one to compute the accuracy.

```
preds <- posterior_epred(post)
pred <- colMeans(preds)
pr <- as.integer(pred >= 0.5)
# posterior classification accuracy
round(mean(xor(pr,as.integer(hfcrd_train$DEATH_EVENT == 0))), 2)
```

```
## [1] 0.82
```

And then the accuracy is 0.82. Notwithstanding, we know that this accuracy computed with the sample used for build the model could be overestimating. Thus, let us do predictions with the test data set and compute again the accuracy.

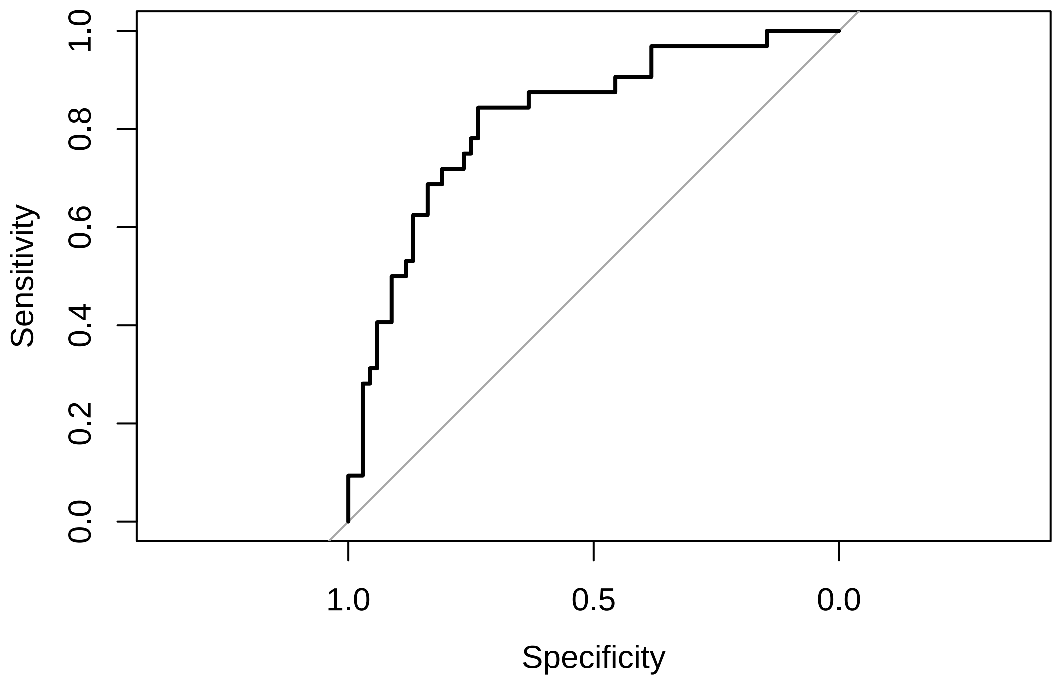
```
preds_test <- posterior_epred(post, newdata = hfcrd_test)
pred_test <- colMeans(preds_test)
pr_test <- as.integer(pred_test >= 0.5)
# posterior classification accuracy
round(mean(xor(pr_test,as.integer(hfcrd_test$DEATH_EVENT == 0))), 2)
```

```
## [1] 0.77
```

Thus, we have obtained an accuracy of 0.77 with the test data set.

We finally find the optimal threshold by means of the ROC curve.

```
roc_default <- roc(DEATH_EVENT ~ pred_test, data = hfcrd_test)
plot(roc_default)
```



```
best_thresh <- coords(roc_default,
                      x = "best",
                      best.method = "closest.topleft")
best_thresh
```

```
##   threshold specificity sensitivity
## 1 0.3137333   0.7352941     0.84375
```

We can see that the best threshold is 0.31, which is pretty far away from the 0.5 chosen in the beginning. Let us compute the accuracy now with this threshold:

```
pr_test_tr <- as.integer(pred_test >= 0.3137333)
# accuracy with new threshold
round(mean(xor(pr_test_tr,as.integer(hfcrd_test$DEATH_EVENT == 0))), 2)
```

```
## [1] 0.77
```

We have obtained the same accuracy, but better specificity and sensitivity.

Let us try to improve the model fitting the model only with those covariates which were significant:

```
post2 <- stan_glm(DEATH_EVENT~ age + ejection_fraction + serum_creatinine  +
                   serum_sodium + time, data = hfcrd_train,
                 family = binomial(link = "logit"),
                 seed = 1234)
```

```
##
## SAMPLING FOR MODEL 'bernoulli' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 1.3e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.13 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.041 seconds (Warm-up)
## Chain 1:                0.038 seconds (Sampling)
## Chain 1:                0.079 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'bernoulli' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 8e-06 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.08 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.039 seconds (Warm-up)
## Chain 2:                0.041 seconds (Sampling)
## Chain 2:                0.08 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'bernoulli' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 7e-06 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.07 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.039 seconds (Warm-up)
## Chain 3:                0.041 seconds (Sampling)
## Chain 3:                0.08 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'bernoulli' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 5e-06 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.05 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.041 seconds (Warm-up)
## Chain 4:                0.041 seconds (Sampling)
## Chain 4:                0.082 seconds (Total)
## Chain 4:
```

The credibility intervals for the new model are:

```r
round(posterior_interval(post2, prob = 0.95), 3)
```

```
##                     2.5%  97.5%
## (Intercept)       11.862 44.016
## age                -0.001  0.072
## ejection_fraction -0.119 -0.040
## serum_creatinine   0.345  1.325
## serum_sodium      -0.312 -0.083
## time              -0.027 -0.013
```

It seems that age is now including the 0 in the 95% credibility interval.

Now, let us see how is the accuracy when predicting in the training sample:

```r
preds2 <- posterior_epred(post2)
pred2 <- colMeans(preds2)
pr2 <- as.integer(pred2 >= 0.5)
# posterior classification accuracy
round(mean(xor(pr2,as.integer(hfcrd_train$DEATH_EVENT == 0))), 2)
```

```
## [1] 0.83
```

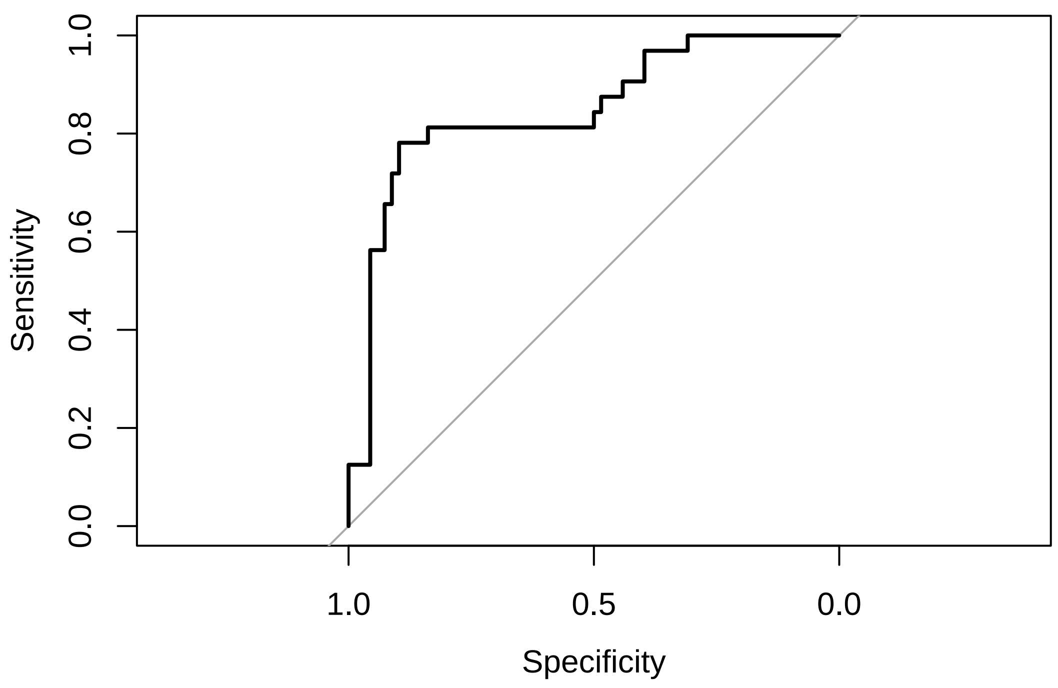
And now we compute the accuracy when predicting in the test sample:

```r
preds_test2 <- posterior_epred(post2, newdata = hfcrd_test)
pred_test2 <- colMeans(preds_test2)
pr_test2 <- as.integer(pred_test2 >= 0.5)
# posterior classification accuracy
round(mean(xor(pr_test2,as.integer(hfcrd_test$DEATH_EVENT == 0))), 2)
```

```
## [1] 0.85
```

So, we have obtained an accuracy of 0.85, which, indeed, improves the accuracy of the previous, which was 0.77. The reader can observe that we have obtained better accuracy when predicting in the test sample than when predicting for the training sample. This is not what happens usually, whoreas it is a possible scenario. Let us now find the best threshold as we did before, by means of the ROC curve:

```r
roc_default2 <- roc(DEATH_EVENT ~ pred_test2, data = hfcrd_test)
plot(roc_default2)
```



We fin the best threshold:

```
best_thresh2 <- coords(roc_default2,
                      x = "best",
                      best.method = "closest.topleft")
best_thresh2
```

```
##   threshold specificity sensitivity
## 1   0.48325   0.8970588     0.78125
```

Hence, the best threshold is 0.48325. Let us see for this quantity how is the accuracy:

```
pr_test2_newt <- as.integer(pred_test2 >= 0.48325)
round(mean(xor(pr_test2_newt,as.integer(hfcrd_test$DEATH_EVENT == 0))), 2)
```

```
## [1] 0.86
```

And so, we are improving the accuracy! In our final Bayesian model we are obtaining an accuracy of 0.86 when predicting the values in the test sample. Let us see other relevant quantities by means of the confusion matrix:

```
(confMat_bayes2 <- confusionMatrix(as.factor(pr_test2_newt), survival_test))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 61  7
##          1  7 25
```

```
##
##                Accuracy : 0.86
##                  95% CI : (0.7763, 0.9213)
##     No Information Rate : 0.68
##     P-Value [Acc > NIR] : 3.065e-05
##
##                   Kappa : 0.6783
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.8971
##             Specificity : 0.7812
##          Pos Pred Value : 0.8971
##          Neg Pred Value : 0.7812
##              Prevalence : 0.6800
##          Detection Rate : 0.6100
##    Detection Prevalence : 0.6800
##       Balanced Accuracy : 0.8392
##
##        'Positive' Class : 0
##
```

5 Conclusions

In this final section we will be comparing the three predictors using the adequate metrics. We mainly focus on the accuracy of the models and the number of variables used.

```r
results <- data.frame(
  Model = c("Classification Tree", "Logistic classifier", "Bayesian logistic classifier"),
  Accuracy = c(0.86, 0.86, 0.86),
  Specificity = c(0.7812,0.7812,0.7812),
  Sensitivity = c(0.8971,0.8971,0.8971),
  NumVariables = c(3,6,5)
)
print(results)
```

```
##                          Model Accuracy Specificity Sensitivity NumVariables
## 1          Classification Tree     0.86      0.7812      0.8971            3
## 2          Logistic classifier     0.86      0.7812      0.8971            6
## 3 Bayesian logistic classifier     0.86      0.7812      0.8971            5
```

As we can see, all three models have the same accuracy, specificity and sensitivity. This means that, at the end they are doing exactly the same classifications on the test data set. Notice that the results exposed in the table of accuracy, specificity and sensitivity are the ones computed when classifying with the test data set. Nonetheless, the classification tree is the model using less variables, and then the simpler one. The following predictor using less variables is the Bayesian logistic classifier, using five variables. Finally we have the logistic classifier, which uses six variables.

In the Classification tree we have analyzed which variables are the most important. Although are more than three variables which seems to be informative, only three of them are used in the classifier tree. The reader can observe that the variables chosen by the models are very similar (in all the cases `age`, `ejection_fraction`, `serum_creatinine`, `serum_sodium` and `time` are the most informative). This is a good sign that our models are selecting informative variables. We expose which variables are used for each model:

- Classification tree: `serum_creatinine`, `ejection_fraction` and `time`.
- Logistic classifier: `diabetes`, `age`, `ejection_fraction`, `serum_creatinine`, `serum_sodium` and `time`.
- Bayesian logistic classifier: `age`, `ejection_fraction`, `serum_creatinine`, `serum_sodium` and `time`.

In conclusion, despite the fact that the three models are performing remarkably similar in terms of classification, we would chose the classification tree because is the one using less variables and hence is the most simple model.