Exercise B.6 for Predictive Modeling

Pablo Vidal Fernández 100483812
José Ignacio Díez Ruiz 100487766
Carlos Roldán Piñero 100484904

2023-01-08

We are asked to replicate Figure 3.5 from the course notes but with some modifications: 10 predictors instead of 5, and only considering AIC and BIC, no LOOCV. Let us detail the followed steps.

First, we define a function to generate a data frame with our response variable and features. This are generated as follows. X is a $n \times p$ matrix (with n the number of observations) filled with normally distributed random numbers $\sim \mathcal{N}(0, 1)$. Then, $\beta = (\beta_0, \vec{\beta})^T$ a $(p+1)$-element vector with values $\beta_0 = 0.5$, $\beta_1 = \beta_2 = 1$, $\beta_{i>2} = 0$. Finally, the response variable is computed,

$$Y = X\vec{\beta} + \vec{\varepsilon} + \beta_0 \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} , \tag{1}$$

with $\vec{\varepsilon}$ a p-element vector $\sim \mathcal{N}(0, 1)$ of noise.

```r
gen_data <- function(n, p) {
  x <- matrix(rnorm(n * p), nrow = n, ncol = p)
  b <- c(1, 1, rep(0, p - 2))
  e <- rnorm(n)
  y <- drop(x %*% b) + e + 0.5
  return(data.frame(y = y, x = x))
}
```

Now we define the function for the Monte Carlo estimation of the probability. The idea is to generate the data, perform a bidirectional step AIC and BIC on the model and record whether it did predict the only non-null predictors to be the intercept and the first two or not. Then we repeat this process M times and average.

```r
prob_model <- function(m, ns, p) {
  # Initialize arrays
  true_aic <- rep(0, length(ns))
  true_bic <- rep(0, length(ns))

  for (i in seq_len(length(ns))) {
    n <- ns[i]
    for (j in 1:m){
      data <- gen_data(n, p)

      # Build the intermediate formula, take in account the n < p/2 case
      f_string <- "y ~ "
      for (k in 1:min(p / 2 - 1, n - 3)) {
        f_string <- paste(f_string, "x.", k, " + ", sep = "")
      }
      f_string <- paste(f_string, "x.", min(p / 2, n - 2), sep = "")

      # Limit models
      mod_zero  <- lm(y ~ 1, data = data)
      mod_all   <- lm(y ~ ., data = data)
      mod_inter <- lm(as.formula(f_string), data = data)

      # Step AIC and step BIC
      red_aic <- MASS::stepAIC(mod_inter, direction = "both", trace = 0, k = 2,
                               scope = list(lower = mod_zero, upper = mod_all))
      red_bic <- MASS::stepAIC(mod_inter, direction = "both",
                               trace = 0, k = log(n),
                               scope = list(lower = mod_zero, upper = mod_all))

      # Add one if correct model
      pred_aic     <- names(red_aic$coefficients)
      pred_bic     <- names(red_bic$coefficients)
      target_names <- c("(Intercept)", "x.1", "x.2")
      true_aic[i] <- true_aic[i] +
        ((length(pred_aic) == 3) && all(pred_aic == target_names))
      true_bic[i] <- true_bic[i] +
        ((length(pred_bic) == 3) && all(pred_bic == target_names))
    }
  }

  # Average and build dataframe
  true_aic <- true_aic / m
  true_bic <- true_bic / m
  return(data.frame(n      = c(ns, ns),
                    p      = c(true_aic, true_bic),
                    method = c(rep("AIC", length(ns)), rep("BIC", length(ns)))
                    ))
}
```

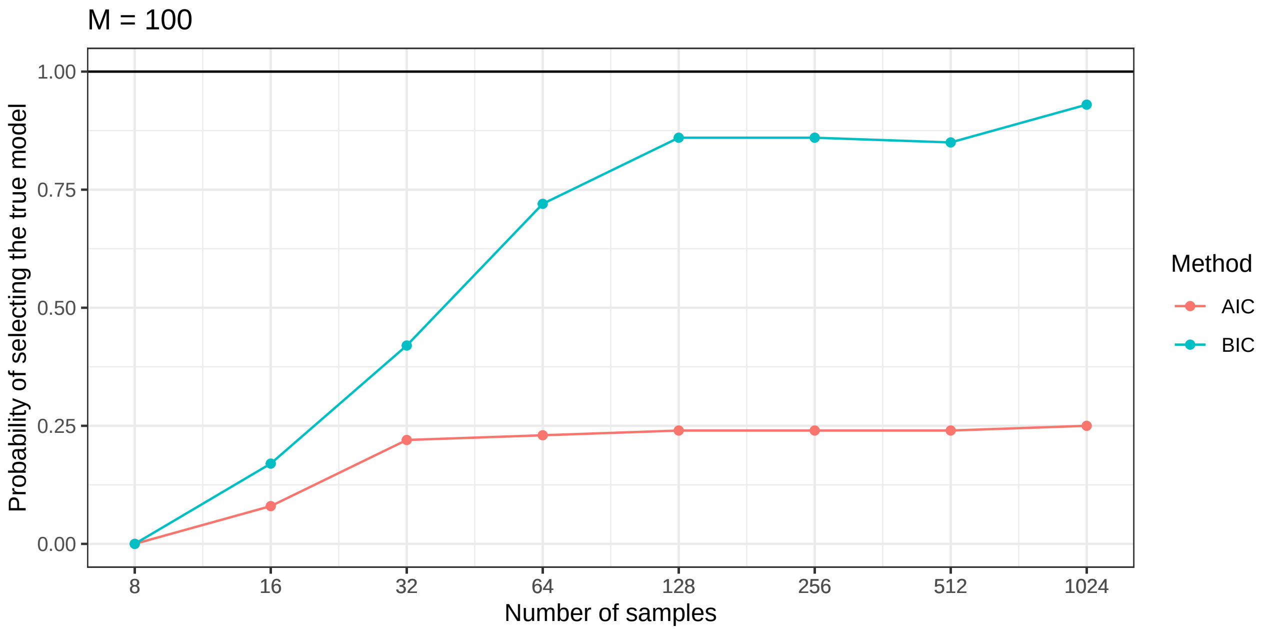
Now we simulate and plot for $M = 100, 200, 500, 1000$.

```
plot_mc <- function(data, m) {
  ggplot(data, aes(x = n, y = p)) +
    geom_line(aes(color = method)) +
    geom_point(aes(color = method)) +
    geom_hline(yintercept = 1) +
    scale_x_continuous(trans = "log2", breaks = data$n) +
    xlab("Number of samples") +
    ylab("Probability of selecting the true model") +
    ggtitle(paste("M =", m)) +
    theme_bw() +
    labs(color = "Method")
}
```

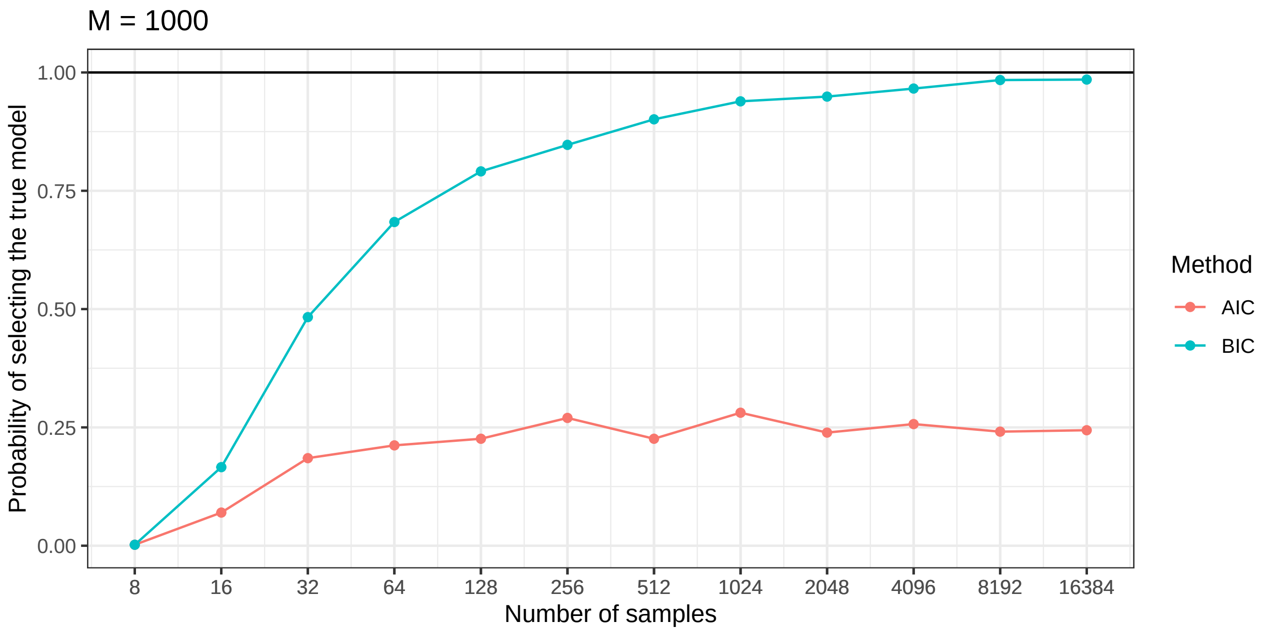
```
## WARNING!!!
## These are time-consuming computations,
## hence we provide the result inside the
## 10.Rdata file
# data100  <- prob_model(100,  2^c(3:10), 10)
# data1000 <- prob_model(1000, 2^c(3:14), 10)
```

```r
load("10.RData")
```

```r
plot_mc(data100,  100)
```



```r
plot_mc(data1000, 1000)
```



We can immediately see that for larger datasets, BIC consistently improves, reaching the 0.9 area for our largest case, with still a positive tendency if we were to consider bigger n's. On the other hand, AIC lacks this consistency and saturates at around or below 0.25, with no hope for further improvement on bigger datasets.

This aforementioned difference comes from the penalization term for extra estimators, which is $\log(n)p$ for BIC and $2p$ for AIC. This does also explain why for small number of samples both of them behave similarly, as $\log(2^l) = l \log(2) \approx 2$, $l = 3, 4$. As we further increment the number of points, the penalization from BIC exceeds that of AIC, favoring less number of predictors, hence improving its chances to predict the true underlying model.