# HW3 Data-driven statistical modelling

Fanny Bergström

28/2/2021

## Programming assignment

In this assignment we will do our own implemention of logistic regression with Lasso penalty using coordinate descent. For the coordinate-wise optimisation step I have chosen to use the quadratic approximations discussed in the book, this means that I will not need a numberic search for the minimum as the approximation is a convex function. We will use the Ionosphere dataset (https://se.mathworks.com/help/stats/regularize-logistic-regression.html).

Figure 1 is showing the beta coefficients as a function of lambda when using the function glmlasso.m from matlab, which we will use as a reference to see if our own implementation below is correct.

```r
library(tidyverse)
library(KernelKnn)
set.seed(123)

ionosphere_beta <- read_table2("ionosphere_beta.txt",
  col_names = FALSE
)
ionosphere_lambda <- read_table2("ionosphere_lambda.txt",
  col_names = FALSE
)
ionosphere_beta %>%
  t() %>%
  as.data.frame() %>%
  mutate(lambda = ionosphere_lambda$X1) %>%
  pivot_longer(V1:V32) %>%
  ggplot(aes(x = lambda, y = value, color = name)) +
  geom_line() +
  scale_x_log10() +
  theme_minimal() +
  theme(legend.position = "none")
```

We load the dataset and standardize the covariates (zero mean and unit variance). Column 1 and 2 is removed (as is done in the matlab example: https://se.mathworks.com/help/stats/regularize-logistic-regression.html) because of their undesirable statistical properties. I've choose to add a column consiting of 1's in the design matrix $X$ which corresponds to the intercept $\beta_0$ for easier code.

```r
data(ionosphere)
df <- ionosphere %>%
  dplyr::select(-V1, -V2, -class) %>%
  scale() %>%
  as.data.frame() %>%
  mutate(class = c(0:1)[match(
    ionosphere$class,
```

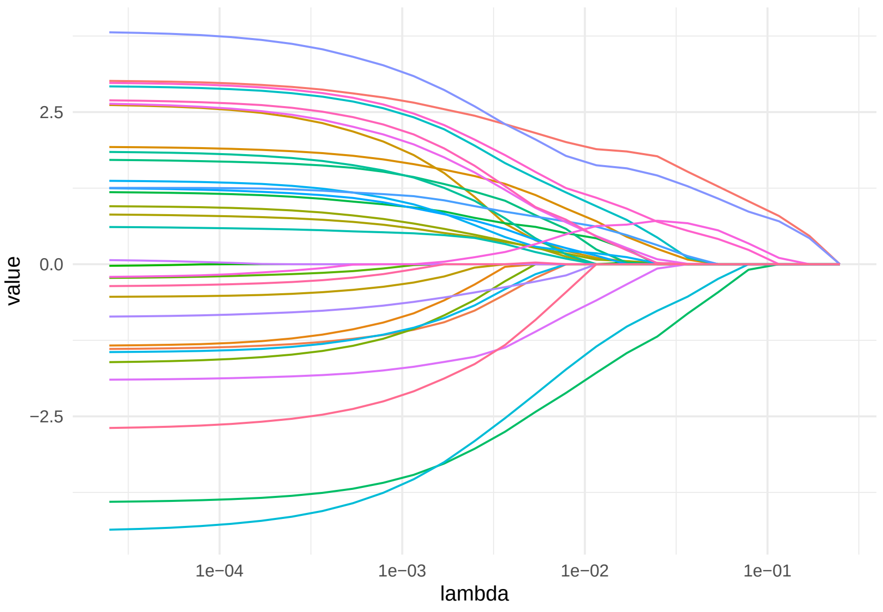

Figure 1: Beta coefficients as a function of Lambda from the glmlasso.m implementation.

```
    sort(unique(ionosphere$class))
  )])

y <- df %>%
  dplyr::select(class) %>%
  as.matrix()

X <- df %>%
  dplyr::select(-class) %>%
  mutate(V0 = 1) %>%
  dplyr::select(V0, everything()) %>%
  as.matrix()
```

The code in the below chunk are some help functions. I have choosen to use the quadratic approximation of the loglikelihood function described in the book 'Statistical Learing with Sparsity' Section 3.3.2. This means that each outer loop then amounts to a weighted lasso regression. The weighted beta updates are then given by

$$\tilde{\beta}_j \leftarrow \frac{S_\lambda(\sum_{i=1}^N w_i(x_{ij}(y_i - \tilde{y}_i^{(j)})))}{\sum_{i=1}^N w_i x_{ij}^2},$$

which is stated in Eq. (10) the paper by Friedman, Hastie and Tibshiranias (2009).

```
# Soft threshold function
soft_threshold <- function(x, lambda) {
  if (isTRUE(abs(x) - lambda > 0)) {
    sign(x) * (abs(x) - lambda)
  }
  else {
    0
  }
}
```

```r
# Approximate the log-likelihood by a quadratic function
q_loglik <- function(betas, w, z, X, lambda) {
  q_l <- 1 / (2 * nrow(X)) * sum(w * (z - (X %*% betas))^2)
  q_l + lambda * sum(abs(betas))
}

# Current state of the coordinate logistic regression
cur_state <- function(X, y, beta) {
  u <- X %*% beta
  p <- exp(u) / (1 + exp(u))
  w <- p * (1 - p)
  z <- (X %*% beta) + ((y - p) / (p * (1 - p)))
  list(p = p, w = w, z = z)
}
```

Following chunk is the Logistic Regression function with Lasso using coordinate decent.

```r
# Logistic LASSO implementation
lasso_cd <- function(X, y, beta, lambda, tol = 1e-5) {

  # Initial values
  names(beta) <- names(X)
  N <- nrow(X)
  state <- cur_state(X, y, beta)
  diff_beta <- Inf

  # Outer loop
  while (diff_beta > tol) {
    beta_old <- beta
    state <- cur_state(X, y, beta)
    beta[1] <- sum(state$w * (state$z - X %*% beta)) / (sum(state$w) * N)

    # Coordinate descent thorugh the betas (excluding intercept)
    for (k in 2:length(beta)) {
      val <- sum(state$w * X[, k] * (state$z - X[, -k] %*% beta[-k]) )
      beta[k] <- (1 / sum(state$w * X[, k]^2)) * soft_threshold(val, lambda * N)
    }


    # Convergence check
    diff_beta <- norm(as.matrix(beta - beta_old), "F")
  }
  beta
}
```

We can now make a plot of the coefficients as a function over different values of $\lambda$. Before plotting the coefficients we will transform them back to the original scale. The regression is given by

$$\hat{Y} = \tilde{\beta}_0 + \tilde{\beta}^T[(x-m)/s] = \tilde{\beta}_0 - \tilde{\beta}^T(m/s) + (\tilde{\beta}/s)^T x = \beta_0 + \beta^T x,$$

which means that the beta coefficients in the original scale are given by

$$\beta_j = \frac{\tilde{\beta}_j}{s_j}.$$

```r
# Visualizations
lambdas <- ionosphere_lambda$X1
f <- formula(class ~ .)
glm_fit <- glm(f, family = binomial(link = "logit"), data = df)

beta_lasso <- NULL
for (l in 1:25) {
  lasso <- lasso_cd(
    X = X, y = y, beta = glm_fit$coefficients,
    lambda = lambdas[l], tol = 1e-4
  )
  beta_lasso <- rbind(beta_lasso, lasso)
}

sds <- c(1,apply(ionosphere, 2, sd)[3:34])

df_betas <- beta_lasso %*% diag(1 / sds) %>% # transformation of the betas to original scale
  as.data.frame() %>%
  mutate(lambda = lambdas) %>%
  tidyr::gather(., key = "coeff", value = "coeff_est", V1:V33) %>%
  mutate(log.lambda = log(lambda))

ggplot(data = df_betas, aes(x = lambda, y = coeff_est, color = coeff, group = coeff)) +
  geom_line() +
  labs(title = "Logistic Regression Lasso Coefficients",
    x = "log(lambda)", y = "value") +
  theme_minimal() +
  scale_x_log10() +
  theme(legend.position = "none")
```

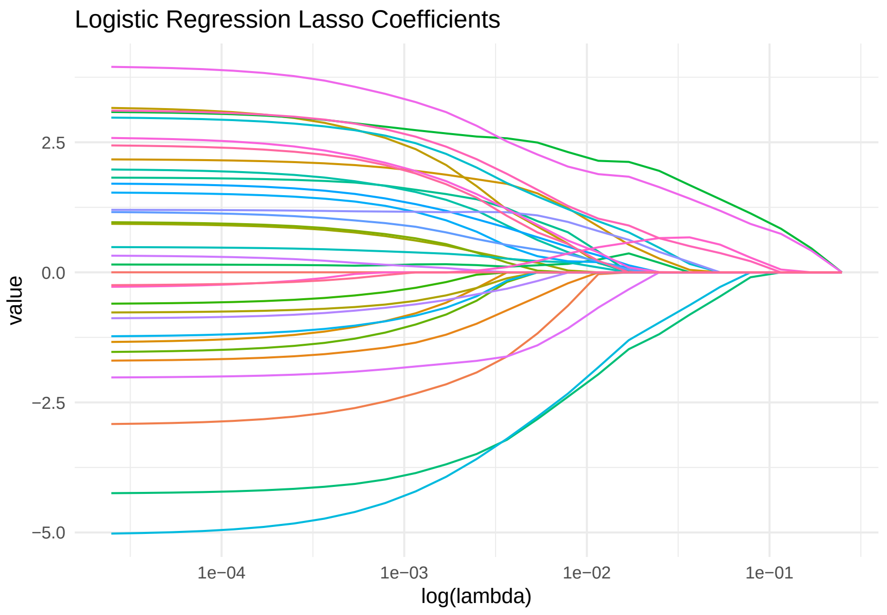

Figure 2: Beta coefficients as a function of Lambda.

When comparing the above results to the ones from glmlasso.m (Fig. 1), we see that the results correspond rather well.