Task 3

Task 3: Build a logistic-LASSO model to select features, and implement a path-wise coordinate-wise optimization algorithm to obtain a path of solutions with a sequence of descending λ's.

Reference: Friedman J, Hastie T, Tibshirani R. Regularization Paths for Generalized Linear Models via Coordinate Descent. J Stat Softw. 2010;33(1):1-22. PMID: 20808728; PMCID: PMC2929880.

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2929880/#FD14

Algorithm

Log-likelihood f in task 1:

$$f(\boldsymbol{\beta}; \mathbf{y}, \mathbf{X}) = \sum_{i=1}^{n} \left[Y_i \mathbf{x}_i^\top \boldsymbol{\beta} - \log\left(1 + e^{\mathbf{x}_i^\top \boldsymbol{\beta}}\right) \right]. \tag{1}$$

LASSO estimates the logistic model parameters $\boldsymbol{\beta}$ by optimizing a penalized loss function:

$$\min_{\boldsymbol{\beta}} \ -\frac{1}{n} f(\boldsymbol{\beta}) + \lambda \sum_{k=1}^{p} |\beta_k|. \tag{2}$$

where $\lambda \geq 0$ is the tuning parameter. Note that the intercept is not penalized and all predictors are standardized.

Algorithm Structure

OUTER LOOP: Decrement λ.
MIDDLE LOOP: Update $\tilde{w}_i$, $\tilde{p}_i$, and thus the quadratic approximation ℓ using the current parameters $\tilde{\boldsymbol{\beta}}$.
INNER LOOP: Run the coordinate descent algorithm on the penalized weighted-least-squares problem.

OUTER LOOP In the outer loop, we compute the solutions of the optimization problem (2) for a decreasing sequence of values for λ: $\{\lambda_1, \ldots, \lambda_m\}$, starting at the smallest value $\lambda_1 = \lambda_{max}$ for which the estimates of all coefficients $\hat{\beta}_j = 0, \ j = 1, 2, \ldots, p$, which is

$$\lambda_{max} = \max_{j \in \{1,\ldots,p\}} \left| \frac{1}{n} \sum_{i=1}^{n} X_{ij}(Y_i - \bar{Y}) \right|, \tag{3}$$

where $\bar{Y} = \frac{1}{n}\sum_{i=1}^{n} Y_i$. For tuning parameter value λ_{k+1}, we initialize coordinate descent algorithm at the computed solution for λ_k (warm start). Apart from giving us a path of solutions, this scheme exploits warm starts, and leads to a more stable algorithm.

MIDDLE LOOP In the middle loop, we find the estimates of $\boldsymbol{\beta}$ by solving the optimization problem (2) for a fixed λ. For each iteration of the middle loop, based on the current parameter estimates $\tilde{\boldsymbol{\beta}}$, we form a quadratic approximation to the log-likelihood f using a Taylor expansion:

$$
\begin{aligned}
f(\boldsymbol{\beta}) \approx \ell(\boldsymbol{\beta}) &= f(\tilde{\boldsymbol{\beta}}) + (\boldsymbol{\beta} - \tilde{\boldsymbol{\beta}})^\top \nabla f(\tilde{\boldsymbol{\beta}}) + \frac{1}{2}(\boldsymbol{\beta} - \tilde{\boldsymbol{\beta}})^\top \nabla^2 f(\tilde{\boldsymbol{\beta}})(\boldsymbol{\beta} - \tilde{\boldsymbol{\beta}}) \\
&= f(\tilde{\boldsymbol{\beta}}) + [\mathbf{X}(\boldsymbol{\beta} - \tilde{\boldsymbol{\beta}})]^\top (\mathbf{y} - \tilde{\mathbf{p}}) - \frac{1}{2}[\mathbf{X}(\boldsymbol{\beta} - \tilde{\boldsymbol{\beta}})]^\top \tilde{\mathbf{W}} \mathbf{X}(\boldsymbol{\beta} - \tilde{\boldsymbol{\beta}}) \\
&= f(\tilde{\boldsymbol{\beta}}) + \sum_{i=1}^n (Y_i - \tilde{p}_i)\mathbf{x}_i^\top (\boldsymbol{\beta} - \tilde{\boldsymbol{\beta}}) - \frac{1}{2}\sum_{i=1}^n \tilde{w}_i \left[\mathbf{x}_i^\top (\boldsymbol{\beta} - \tilde{\boldsymbol{\beta}})\right]^2 \\
&= -\frac{1}{2}\sum_{i=1}^n \tilde{w}_i \left\{ \left[\mathbf{x}_i^\top (\tilde{\boldsymbol{\beta}} - \boldsymbol{\beta})\right]^2 + 2\frac{Y_i - \tilde{p}_i}{\tilde{w}_i}\left[\mathbf{x}_i^\top (\tilde{\boldsymbol{\beta}} - \boldsymbol{\beta})\right] \right\} + f(\tilde{\boldsymbol{\beta}}) \\
&= -\frac{1}{2}\sum_{i=1}^n \tilde{w}_i \left[\mathbf{x}_i^\top (\tilde{\boldsymbol{\beta}} - \boldsymbol{\beta}) + \frac{Y_i - \tilde{p}_i}{\tilde{w}_i}\right] + \frac{1}{2}\sum_{i=1}^n \tilde{w}_i \left(\frac{Y_i - \tilde{p}_i}{\tilde{w}_i}\right)^2 + f(\tilde{\boldsymbol{\beta}}),
\end{aligned}
$$

where $\tilde{\mathbf{p}} = (\tilde{p}_1, \dots, \tilde{p}_n)^\top$ and $\tilde{\mathbf{W}} = \text{diag}(\tilde{w}_1, \dots, \tilde{w}_n)$ are the estimates of $\mathbf{p}$ and $\mathbf{W}$ based on $\tilde{\boldsymbol{\beta}}$.
We rewrite the function $\ell(\boldsymbol{\beta})$ as follows:

$$
\ell(\boldsymbol{\beta}) = -\frac{1}{2}\sum_{i=1}^n \tilde{w}_i (\tilde{z}_i - \mathbf{x}_i^\top \boldsymbol{\beta})^2 + C(\tilde{\boldsymbol{\beta}}), \tag{4}
$$

where

$$
\tilde{z}_i = \mathbf{x}_i^\top \tilde{\boldsymbol{\beta}} + \frac{Y_i - \tilde{p}_i}{\tilde{w}_i}
$$

is the working response, $\tilde{w}_i$ is the working weight, and C is a function that does not depend on $\boldsymbol{\beta}$.

INNER LOOP. In the inner loop, we find the estimates of $\boldsymbol{\beta}$ by solving a modified optimization problem of (2). With fixed $\tilde{w}_i$'s, $\tilde{z}_i$'s, and a fixed form of ℓ based on the estimates of $\boldsymbol{\beta}$ in the previous iteration of the middle loop, we use coordinate descent to solve the penalized weighted least-squares problem

$$
\min_{\boldsymbol{\beta}} \; -\frac{1}{n}\ell(\boldsymbol{\beta}) + \lambda \sum_{k=1}^p |\beta_k|, \tag{5}
$$

and update the estimates of $\boldsymbol{\beta}$. For each iteration of the inner loop, suppose we have the current estimates $\tilde{\beta}_k$ for $k \neq j$ and we wish to partially optimize with respect to β_j:

$$
\min_{\beta_j} \; \frac{1}{2n}\sum_{i=1}^n \tilde{w}_i \left(\tilde{z}_i - X_{ij}\beta_j - \sum_{k \neq j} X_{ik}\tilde{\beta}_k \right)^2 + \lambda|\beta_j| + \lambda \sum_{k \neq j} |\tilde{\beta}_k|.
$$

Updates:

$$
\begin{aligned}
\tilde{\beta}_0 &\leftarrow \frac{\sum_{i=1}^n \tilde{w}_i (\tilde{z}_i - \sum_{k=1}^p X_{ik}\tilde{\beta}_k)}{\sum_{i=1}^n \tilde{w}_i}, \\
\tilde{\beta}_j &\leftarrow \frac{S\left(\frac{1}{n}\sum_{i=1}^n \tilde{w}_i X_{ij}(\tilde{z}_i - \sum_{k \neq j} x_{ik}\tilde{\beta}_k), \lambda\right)}{\frac{1}{n}\sum_{i=1}^n \tilde{w}_i X_{ij}^2}, \; j = 1, \dots, p
\end{aligned}
$$

where $S(z, \gamma)$ is the soft-thresholding operator with value

$$
S(z, \gamma) = \text{sign}(z)(|z| - \gamma)_+ = \begin{cases} z - \gamma, & \text{if } z > 0 \text{ and } \gamma < |z| \\ z + \gamma, & \text{if } z < 0 \text{ and } \gamma < |z| \\ 0, & \text{if } \gamma \geq |z| \end{cases}
$$

We can then update estimates of β_j's repeatedly for $j = 0, 1, 2, ..., p, 0, 1, 2, ...$ until convergence.

Note: Care is taken to avoid coefficients diverging in order to achieve fitted probabilities of 0 or 1. When a probability is within $\epsilon = 10^{-5}$ of 1, we set it to 1, and set the weights to ϵ. 0 is treated similarly.

Algorithm 1 Path-wise coordinate-wise optimization algorithm

Require: $g(\boldsymbol{\beta}, \lambda) = -\frac{1}{n} f(\boldsymbol{\beta}) + \lambda \sum_{k=1}^{p} |\beta_k|$ - target function, where $f(\boldsymbol{\beta})$ is given in (1); $\boldsymbol{\beta}_0$ - starting value; $\{\lambda_1, \ldots, \lambda_m\}$ - a sequence of descending λ's, where $\lambda_1 = \lambda_{max}$ is given in (3); ϵ - tolerance; N_s, N_t - maximum number of iterations of the middle and inner loops

Ensure: $\widehat{\boldsymbol{\beta}}(\lambda_r)$ such that $\widehat{\boldsymbol{\beta}}(\lambda_r) \approx \arg\min_{\boldsymbol{\beta}}\ g(\boldsymbol{\beta}, \lambda_r),\ r = 1, \ldots, m$

$\tilde{\boldsymbol{\beta}}_0(\lambda_1) \leftarrow \boldsymbol{\beta}_0$
OUTER LOOP
for $r \in \{1, \ldots, m\}$, where r is the current number of iterations of the outer loop, **do**
    $s \leftarrow 0$, where s is the current number of iterations of the middle loop
    $g(\tilde{\boldsymbol{\beta}}_{-1}(\lambda_r), \lambda_r) \leftarrow \infty$
    MIDDLE LOOP
    while $t \geq 2$ and $s < N_s$ **do**
        $s \leftarrow s + 1$
        Update $\tilde{w}_i^{(s)}$, $\tilde{z}_i^{(s)}$ $(i = 1, \ldots, n)$, and thus $\ell_s(\boldsymbol{\beta})$ as given in (4) based on $\tilde{\boldsymbol{\beta}}_{s-1}(\lambda_r)$
        $t \leftarrow 0$, where t is the current number of iterations of the inner loop
        $\tilde{\boldsymbol{\beta}}_s^{(0)}(\lambda_r) \leftarrow \tilde{\boldsymbol{\beta}}_{s-1}(\lambda_r)$
        $h_s(\tilde{\boldsymbol{\beta}}_s^{(-1)}(\lambda_r), \lambda_r) \leftarrow \infty$, where $h_s(\boldsymbol{\beta}, \lambda) = -\frac{1}{n}\ell_s(\boldsymbol{\beta}) + \lambda \sum_{k=1}^{p} |\beta_k|$
        INNER LOOP
        while $\left| h_s(\tilde{\boldsymbol{\beta}}_s^{(t)}(\lambda_r), \lambda_r) - h_s(\tilde{\boldsymbol{\beta}}_s^{(t-1)}(\lambda_r), \lambda_r) \right| > \epsilon$ and $t < N_t$ **do**
            $t \leftarrow t + 1$
            $\tilde{\beta}_0^{(t)}(\lambda_r) \leftarrow \sum_{i=1}^{n} \tilde{w}_i^{(s)} \left(\tilde{z}_i^{(s)} - \sum_{k=1}^{p} X_{ik} \tilde{\beta}_k^{(t-1)}(\lambda_r) \right) \Big/ \sum_{i=1}^{n} \tilde{w}_i^{(s)}$
            for $j \in \{1, \ldots, p\}$ **do**
                $\tilde{\beta}_j^{(t)}(\lambda_r) \leftarrow S\left(\frac{1}{n} \sum_{i=1}^{n} \tilde{w}_i^{(s)} X_{ij} \left(\tilde{z}_i^{(s)} - \sum_{k<j} X_{ik} \tilde{\beta}_k^{(t)}(\lambda_r) - \sum_{k>j} X_{ik} \tilde{\beta}_k^{(t-1)}(\lambda_r) \right), \lambda_r \right) \Big/ \frac{1}{n} \sum_{i=1}^{n} \tilde{w}_i^{(s)} X_{ij}^2$
            end for
        end while
        $\tilde{\boldsymbol{\beta}}_s(\lambda_r) \leftarrow \tilde{\boldsymbol{\beta}}_s^{(t)}(\lambda_r)$
    end while
    $\widehat{\boldsymbol{\beta}}(\lambda_r) \leftarrow \tilde{\boldsymbol{\beta}}_s(\lambda_r)$
    $\tilde{\boldsymbol{\beta}}_0(\lambda_{r+1}) \leftarrow \widehat{\boldsymbol{\beta}}(\lambda_r)$
end for

Implementation in R

target functions needed to be optimized and soft-threshold operator

```r
# function -ell/n (without C) with penalties (minimize!) used in inner loop's convergence criterion
coordinate_func <- function(X, z, w, betavec, lambda) {
  0.5 * sum(w * (z - X %*% betavec)^2) / nrow(X) + lambda * sum(abs(betavec[-1]))
}

# soft-threshold operator used in inner loop
soft.threshold <- function(z, gamma) {
  sign(z) * max(abs(z) - gamma, 0)
}
```

We implement the algorithm in **R**.

```r
# outer loop
LogisticLASSO <- function(dat, start, lambda) {
  r <- length(lambda)
  X <- as.matrix(cbind(rep(1, nrow(dat)), dat[, -1])) # design matrix
  y <- dat[, 1] # response vector
  res <- matrix(NA, nrow = r, ncol = ncol(dat) + 1)
  for (i in 1:r) {
    betavec <- MiddleLoop(X = X, y = y, start = start, lambda = lambda[i])
    res[i, ] <- c(lambda[i], betavec)
    start <- betavec
  }
  colnames(res) <- c("lambda", "(Intercept)", names(dat)[-1])
  return(res)
}

# middle loop

MiddleLoop <- function(X, y, start, lambda, maxiter = 100) {
  betavec <- start
  s <- 0
  eps <- 1e-5
  repeat {
    s <- s + 1
    u <- X %*% betavec
    p_vec <- sigmoid(u) # function `sigmoid` to compute exp(x)/(1 + exp(x))
    w <- p_vec * (1 - p_vec)
    # see note
    p_vec[p_vec < eps] <- 0
    p_vec[p_vec > 1 - eps] <- 1
    w[p_vec == 1 | p_vec == 0] <- eps
    z <- u + (y - p_vec) / w
    betavec <- InnerLoop(X = X, z = z, w = w, betavec = betavec, lambda = lambda)
    t <- betavec[1]
    betavec <- betavec[-1]
    if (t == 1 || s >= maxiter) { # if number of iterations of inner loop = 1, converge.
      break
    }
  }
```

```
  return(betavec)
}

# inner loop
InnerLoop <- function(X, z, w, betavec, lambda, tol = 1e-10, maxiter = 1000) {
  prevfunc <- Inf
  curfunc <- coordinate_func(X = X, z = z, w = w, betavec = betavec, lambda = lambda)
  t <- 0
  while (abs(curfunc - prevfunc) > tol && t < maxiter) {
    t <- t + 1
    prevfunc <- curfunc
    betavec[1] <- sum(w * (z - X[, -1] %*% betavec[-1])) / sum(w)
    for (j in 2:length(betavec)) {
      betavec[j] <- soft.threshold(z = sum(w * X[, j] * (z - X[, -j] %*% betavec[-j])) / nrow(X), gamma
    }
    curfunc <- coordinate_func(X = X, z = z, w = w, betavec = betavec, lambda = lambda)
  }
  return(c(t, betavec))
}
```

Model fit on training data

We fit a logistic-LASSO model on the training data using our function `LogisticLASSO` with a sequence of descending λ's.

```
lambda_max <- max(abs(t(x) %*% (y - mean(y)))) / length(y) + 1e-10 # avoid computational error

lambdas <- exp(seq(log(lambda_max), log(lambda_max) - 4, length = 15))
res <- LogisticLASSO(dat = Training, start = rep(0, ncol(Training)),
                     lambda = lambdas)
res
```

```
##            lambda (Intercept) radius_mean texture_mean perimeter_mean area_mean
##  [1,] 0.395134947  -0.5295835           0    0.0000000              0         0
##  [2,] 0.296934940  -0.5524566           0    0.0000000              0         0
##  [3,] 0.223139865  -0.5910713           0    0.0000000              0         0
##  [4,] 0.167684542  -0.6318088           0    0.0000000              0         0
##  [5,] 0.126011126  -0.6701188           0    0.0000000              0         0
##  [6,] 0.094694500  -0.7025876           0    0.0000000              0         0
##  [7,] 0.071160766  -0.7272230           0    0.0000000              0         0
##  [8,] 0.053475700  -0.7460048           0    0.0000000              0         0
##  [9,] 0.040185774  -0.7499579           0    0.0000000              0         0
## [10,] 0.030198697  -0.7444002           0    0.0447684              0         0
## [11,] 0.022693635  -0.7233096           0    0.1545311              0         0
## [12,] 0.017053751  -0.6932370           0    0.2505568              0         0
## [13,] 0.012815507  -0.6538671           0    0.3306242              0         0
## [14,] 0.009630562  -0.5998805           0    0.4043298              0         0
## [15,] 0.007237149  -0.5372183           0    0.4708138              0         0
##       smoothness_mean compactness_mean concavity_mean concave.points_mean
##  [1,]               0                0              0          0.00000000
##  [2,]               0                0              0          0.00000000
##  [3,]               0                0              0          0.00000000
```

```
##  [4,]              0              0            0         0.00000000
##  [5,]              0              0            0         0.00000000
##  [6,]              0              0            0         0.05439274
##  [7,]              0              0            0         0.17198760
##  [8,]              0              0            0         0.30216261
##  [9,]              0              0            0         0.44702109
## [10,]              0              0            0         0.55167107
## [11,]              0              0            0         0.54744077
## [12,]              0              0            0         0.52852121
## [13,]              0              0            0         0.49336236
## [14,]              0              0            0         0.48457215
## [15,]              0              0            0         0.52735694
##       symmetry_mean fractal_dimension_mean  radius_se texture_se perimeter_se
##  [1,]             0                      0 0.00000000          0            0
##  [2,]             0                      0 0.00000000          0            0
##  [3,]             0                      0 0.00000000          0            0
##  [4,]             0                      0 0.00000000          0            0
##  [5,]             0                      0 0.00000000          0            0
##  [6,]             0                      0 0.00000000          0            0
##  [7,]             0                      0 0.00000000          0            0
##  [8,]             0                      0 0.00000000          0            0
##  [9,]             0                      0 0.00000000          0            0
## [10,]             0                      0 0.02939158          0            0
## [11,]             0                      0 0.18251978          0            0
## [12,]             0                      0 0.35993143          0            0
## [13,]             0                      0 0.56091818          0            0
## [14,]             0                      0 0.83094267          0            0
## [15,]             0                      0 1.14904111          0            0
##       area_se smoothness_se compactness_se concavity_se concave.points_se
##  [1,]       0             0              0            0                 0
##  [2,]       0             0              0            0                 0
##  [3,]       0             0              0            0                 0
##  [4,]       0             0              0            0                 0
##  [5,]       0             0              0            0                 0
##  [6,]       0             0              0            0                 0
##  [7,]       0             0              0            0                 0
##  [8,]       0             0              0            0                 0
##  [9,]       0             0              0            0                 0
## [10,]       0             0              0            0                 0
## [11,]       0             0              0            0                 0
## [12,]       0             0              0            0                 0
## [13,]       0             0              0            0                 0
## [14,]       0             0              0            0                 0
## [15,]       0             0              0            0                 0
##       symmetry_se fractal_dimension_se radius_worst texture_worst
##  [1,]           0           0.00000000    0.0000000   0.000000000
##  [2,]           0           0.00000000    0.0000000   0.000000000
##  [3,]           0           0.00000000    0.1206156   0.000000000
##  [4,]           0           0.00000000    0.4297921   0.000000000
##  [5,]           0           0.00000000    0.6085099   0.000000000
##  [6,]           0           0.00000000    0.7826490   0.001346751
##  [7,]           0           0.00000000    0.9300058   0.121096185
##  [8,]           0           0.00000000    1.1000975   0.238337349
##  [9,]           0           0.00000000    1.3387832   0.343497992
```

```
## [10,]          0          0.00000000    1.6009253   0.403611300
## [11,]          0          0.00000000    1.8367239   0.398189345
## [12,]          0          0.00000000    2.0987475   0.404090305
## [13,]          0          0.00000000    2.3938984   0.424786650
## [14,]          0         -0.05238337    2.6381938   0.451675214
## [15,]          0         -0.16690242    2.8315938   0.487159914
##       perimeter_worst area_worst smoothness_worst compactness_worst
##  [1,]       0.0000000          0       0.00000000                 0
##  [2,]       0.1051523          0       0.00000000                 0
##  [3,]       0.1500945          0       0.00000000                 0
##  [4,]       0.0000000          0       0.00000000                 0
##  [5,]       0.0000000          0       0.00000000                 0
##  [6,]       0.0000000          0       0.00000000                 0
##  [7,]       0.0000000          0       0.00000000                 0
##  [8,]       0.0000000          0       0.00000000                 0
##  [9,]       0.0000000          0       0.04691526                 0
## [10,]       0.0000000          0       0.12680700                 0
## [11,]       0.0000000          0       0.21826754                 0
## [12,]       0.0000000          0       0.31530738                 0
## [13,]       0.0000000          0       0.42083435                 0
## [14,]       0.0000000          0       0.51734612                 0
## [15,]       0.0000000          0       0.60122136                 0
##       concavity_worst concave.points_worst symmetry_worst
##  [1,]      0.00000000            0.0000000     0.00000000
##  [2,]      0.00000000            0.3195118     0.00000000
##  [3,]      0.00000000            0.5322324     0.00000000
##  [4,]      0.00000000            0.7359560     0.00000000
##  [5,]      0.00000000            0.9157618     0.00000000
##  [6,]      0.00000000            1.0522435     0.00000000
##  [7,]      0.00000000            1.1076377     0.00000000
##  [8,]      0.00000000            1.1412645     0.01450593
##  [9,]      0.00000000            1.0643104     0.08829545
## [10,]      0.00000000            0.9974356     0.15469660
## [11,]      0.00000000            1.0095475     0.22109589
## [12,]      0.02287587            1.0163350     0.27882331
## [13,]      0.07606781            1.0097876     0.32768971
## [14,]      0.16307403            1.0185758     0.36826630
## [15,]      0.28610709            1.0381665     0.40163889
##       fractal_dimension_worst
##  [1,]                       0
##  [2,]                       0
##  [3,]                       0
##  [4,]                       0
##  [5,]                       0
##  [6,]                       0
##  [7,]                       0
##  [8,]                       0
##  [9,]                       0
## [10,]                       0
## [11,]                       0
## [12,]                       0
## [13,]                       0
## [14,]                       0
## [15,]                       0
```