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

$$\lambda_{max} = \max_j |\langle \mathbf{x}_{\cdot j}, \mathbf{y} \rangle| , \tag{3}$$

where $\mathbf{x}_{\cdot j}$ is the j-th column of the design matrix $\mathbf{X}$, for $j = 1, \ldots, p$.
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:

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

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

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 convergence criterion of the middle loop is not met: $\left| g(\tilde{\boldsymbol{\beta}}_s(\lambda_r), \lambda_r) - g(\tilde{\boldsymbol{\beta}}_{s-1}(\lambda_r), \lambda_r) \right| > \epsilon$ **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 convergence criterion of the inner loop is not met: $\left| h_s(\tilde{\boldsymbol{\beta}}_s^{(t)}(\lambda_r), \lambda_r) - h_s(\tilde{\boldsymbol{\beta}}_s^{(t-1)}(\lambda_r), \lambda_r) \right| > \epsilon$ **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

```
# function -f/n with penalties (minimize!) used in middle loop's convergence criterion
logitLASSO_func <- function(u, y, betavec, lambda) {
  -sum(y * u - log1pexp(u)) / length(y) + lambda * sum(abs(betavec[-1]))
}

# 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) * (abs(z) - gamma) * (abs(z) - gamma > 0)
}
```

We implement the algorithm in **R**.

```
# outer loop
LogisticLASSO <- function(dat, start, lambda) {
  k <- length(lambda)
  X <- as.matrix(cbind(rep(1, nrow(dat)), dat[, -1])) # design matrix
  y <- dat[, 1] # response vector
  res <- matrix(NA, nrow = k, ncol = ncol(dat) + 1)
  for (i in 1:k) {
    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, tol = 1e-10) {
  prevfunc <- Inf
  betavec <- start
  u <- X %*% betavec
  p_vec <- sigmoid(u) # function `sigmoid` to compute exp(x)/(1 + exp(x))
  w <- p_vec * (1 - p_vec)
  eps <- 1e-5
  # 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
  curfunc <- logitLASSO_func(u = u, y = y, betavec = betavec, lambda = lambda)
  while (abs(curfunc - prevfunc) > tol) {
    prevfunc <- curfunc
    betavec <- InnerLoop(X = X, z = z, w = w, betavec = betavec, lambda = lambda)
    u <- X %*% betavec
```

```
    curfunc <- logitLASSO_func(u = u, y = y, betavec = betavec, lambda = lambda)
  }
  return(betavec)
}

# inner loop
InnerLoop <- function(X, z, w, betavec, lambda, tol = 1e-10) {
  prevfunc <- Inf
  curfunc <- coordinate_func(X = X, z = z, w = w, betavec = betavec, lambda = lambda)
  while (abs(curfunc - prevfunc) > tol) {
    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 / nrow(X) * X[, j] * (z - X[, -j] %*% betavec[-j])), gamma
    }
    curfunc <- coordinate_func(X = X, z = z, w = w, betavec = betavec, lambda = lambda)
  }
  return(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.

```
X <- as.matrix(Training[, -1])
y <- Training[, 1]
lambda_max <- max(t(X) %*% y) / length(y)

lambdas <- exp(seq(log(lambda_max), -15, length = 20))
res <- LogisticLASSO(dat = Training, start = rep(0, ncol(Training)),
                     lambda = lambdas)
res
```

```
##             lambda (Intercept) radius_mean texture_mean perimeter_mean
##  [1,] 3.880308e-01 -0.51754386   0.0000000   0.00000000      0.0000000
##  [2,] 1.851999e-01 -0.53849285   0.0000000   0.00000000      0.1391837
##  [3,] 8.839247e-02 -0.61179266   0.0000000   0.06785343      0.4051150
##  [4,] 4.218809e-02 -0.63812955   0.0000000   0.28470223      0.6307570
##  [5,] 2.013559e-02 -0.65867445   0.8774479   0.53866005      0.0000000
##  [6,] 9.610345e-03 -0.68259237   1.1251315   0.80974652      0.0000000
##  [7,] 4.586839e-03 -0.61510574   0.0000000   1.06575063      0.0000000
##  [8,] 2.189213e-03 -0.60943988   0.0000000   1.28008023      0.0000000
##  [9,] 1.044871e-03 -0.61006830   0.0000000   1.43720595      0.0000000
## [10,] 4.986975e-04 -0.61003848   0.0000000   1.52613866      0.0000000
## [11,] 2.380191e-04 -0.38589619  -2.3182387   1.55298614      0.0000000
## [12,] 1.136021e-04 -0.03417447  -6.0719952   1.55920272      0.0000000
## [13,] 5.422017e-05  0.14680957  -7.9999705   1.57947248      0.0000000
## [14,] 2.587828e-05  0.27811112 -12.9034409   1.58837653      4.0253465
## [15,] 1.235122e-05  0.34649187 -15.7004353   1.59518709      6.4029256
## [16,] 5.895010e-06  0.37922220 -16.9885759   1.59858794      7.4788652
## [17,] 2.813579e-06  0.39491276 -17.6562062   1.60032725      8.0561074
```

```
## [18,] 1.342869e-06  0.40240102 -17.9747857   1.60116189      8.3315520
## [19,] 6.409265e-07  0.40597522 -18.1268385   1.60155916      8.4630144
## [20,] 3.059023e-07  0.40768145 -18.1993055   1.60174855      8.5256339
##        area_mean smoothness_mean compactness_mean concavity_mean
##  [1,]  0.0000000       0.0000000       0.00000000     0.00000000
##  [2,]  0.0000000       0.0000000       0.00000000     0.00000000
##  [3,]  0.0000000       0.0000000       0.00000000     0.00000000
##  [4,]  0.0000000       0.0000000       0.00000000     0.00000000
##  [5,]  0.0000000       0.0000000       0.00000000     0.00000000
##  [6,]  0.2795794       0.1435188       0.00000000     0.00000000
##  [7,]  2.1376206       0.3684955       0.00000000     0.04820929
##  [8,]  2.7843541       0.6472058       0.00000000     0.31147960
##  [9,]  3.2448662       0.8619461       0.00000000     0.54217523
## [10,]  3.5024713       0.9797358      -0.01424478     0.67282407
## [11,]  6.1688009       1.0001853       0.00000000     0.61660547
## [12,] 10.3394063       0.9813061       0.20738636     0.47718010
## [13,] 12.5227142       0.9728306       0.32811127     0.41163247
## [14,] 13.8196049       1.0174558       0.13750788     0.36961971
## [15,] 14.4788574       1.0464764       0.02118120     0.35128546
## [16,] 14.8015606       1.0598091      -0.02713414     0.34275984
## [17,] 14.9495627       1.0669428      -0.05738191     0.33870598
## [18,] 15.0201947       1.0703528      -0.07180743     0.33678205
## [19,] 15.0539067       1.0719794      -0.07869267     0.33586247
## [20,] 15.0700090       1.0727537      -0.08197024     0.33542268
##       concave.points_mean symmetry_mean fractal_dimension_mean
##  [1,]            0.000000    0.00000000             0.00000000
##  [2,]            0.701543    0.00000000             0.00000000
##  [3,]            1.106430    0.00000000             0.00000000
##  [4,]            1.587912    0.00000000             0.00000000
##  [5,]            2.092273    0.08709797             0.00000000
##  [6,]            2.283327    0.24158389             0.00000000
##  [7,]            2.331009    0.37935211            -0.02228411
##  [8,]            2.141156    0.48782549            -0.13487894
##  [9,]            1.998785    0.55727625            -0.24378730
## [10,]            1.937169    0.59529249            -0.30062201
## [11,]            2.052213    0.59964338            -0.33003107
## [12,]            2.202131    0.59477283            -0.46261370
## [13,]            2.309321    0.60334453            -0.54844597
## [14,]            2.265973    0.61626639            -0.56477385
## [15,]            2.235381    0.62531058            -0.57324051
## [16,]            2.221311    0.62958199            -0.58004534
## [17,]            2.214085    0.63183003            -0.58088417
## [18,]            2.210630    0.63290427            -0.58129277
## [19,]            2.208980    0.63341628            -0.58148686
## [20,]            2.208197    0.63366020            -0.58158059
```

Compare the results of logistic-LASSO model with $\lambda = e^{-15}$ using our function and logistic model (i.e., $\lambda = 0$) using the `glm` function:

```
tibble(
  predictor = c("(Intercept)", names(Training)[-1]),
  ours_lambda.exp.neg.15 = res[nrow(res), -1],
  glm_lambda.0 = glm(diagnosis ~ ., family = binomial(link = "logit"), data = Training)$coefficients
) %>%
```

```
knitr::kable()
```

predictor	ours_lambda.exp.neg.15	glm_lambda.0
(Intercept)	0.4076814	0.4117085
radius_mean	-18.1993055	-18.3611026
texture_mean	1.6017485	1.6021463
perimeter_mean	8.5256339	8.6625595
area_mean	15.0700090	15.1087870
smoothness_mean	1.0727537	1.0743641
compactness_mean	-0.0819702	-0.0887978
concavity_mean	0.3354227	0.3342689
concave.points_mean	2.2081969	2.2066530
symmetry_mean	0.6336602	0.6341794
fractal_dimension_mean	-0.5815806	-0.5819608