

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:

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

LASSO estimates the logistic model parameters β by optimizing a penalized loss function:

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

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 $\tilde{\beta}$.

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 : $\{\lambda_1, \dots, \lambda_m\}$, starting at the smallest value $\lambda_1 = \lambda_{max}$ for which the estimates of all coefficients $\hat{\beta}_j = 0$, $j = 1, 2, \dots, p$, which is

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

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 β by solving the optimization problem (2) for a fixed λ . For each iteration of the middle loop, based on the current parameter estimates $\tilde{\beta}$, we form a

quadratic approximation to the log-likelihood f using a Taylor expansion:

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

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 $\tilde{\beta}$. We rewrite the function $\ell(\beta)$ as follows:

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

where

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

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

INNER LOOP. In the inner loop, we find the estimates of β 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 β in the previous iteration of the middle loop, we use coordinate descent to solve the penalized weighted least-squares problem

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

and update the estimates of β . 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:

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

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 $j = 0, 1, 2, \dots, p, 0, 1, 2, \dots$ 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: $g(\beta, \lambda) = -\frac{1}{n}f(\beta) + \lambda \sum_{k=1}^p |\beta_k|$ - target function, where $f(\beta)$ is given in (1); β_0 - starting value; $\{\lambda_1, \dots, \lambda_m\}$ - 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: $\hat{\beta}(\lambda_r)$ such that $\hat{\beta}(\lambda_r) \approx \arg \min_{\beta} g(\beta, \lambda_r)$, $r = 1, \dots, m$

```

1:  $\tilde{\beta}_0(\lambda_1) \leftarrow \beta_0$ 
2: OUTER LOOP
3: for  $r \in \{1, \dots, m\}$ , where  $r$  is the current number of iterations of the outer loop, do
4:    $s \leftarrow 0$ , where  $s$  is the current number of iterations of the middle loop
5:    $g(\tilde{\beta}_{-1}(\lambda_r), \lambda_r) \leftarrow \infty$ 
6:   MIDDLE LOOP
7:   while  $t \geq 2$  and  $s < N_s$  do
8:      $s \leftarrow s + 1$ 
9:     Update  $\tilde{w}_i^{(s)}, \tilde{z}_i^{(s)}$  ( $i = 1, \dots, n$ ), and thus  $\ell_s(\beta)$  as given in (4) based on  $\tilde{\beta}_{s-1}(\lambda_r)$ 
10:     $t \leftarrow 0$ , where  $t$  is the current number of iterations of the inner loop
11:     $\tilde{\beta}_s^{(0)}(\lambda_r) \leftarrow \tilde{\beta}_{s-1}(\lambda_r)$ 
12:     $h_s(\tilde{\beta}_s^{(-1)}(\lambda_r), \lambda_r) \leftarrow \infty$ , where  $h_s(\beta, \lambda) = -\frac{1}{n}\ell_s(\beta) + \lambda \sum_{k=1}^p |\beta_k|$ 
13:    INNER LOOP
14:    while  $|h_s(\tilde{\beta}_s^{(t)}(\lambda_r), \lambda_r) - h_s(\tilde{\beta}_s^{(t-1)}(\lambda_r), \lambda_r)| > \epsilon$  and  $t < N_t$  do
15:       $t \leftarrow t + 1$ 
16:       $\tilde{\beta}_0^{(t)}(\lambda_r) \leftarrow \sum_{i=1}^n \tilde{w}_i^{(s)} \left( \tilde{z}_i^{(s)} - \sum_{k=1}^p X_{ik} \tilde{\beta}_k^{(t-1)}(\lambda_r) \right) / \sum_{i=1}^n \tilde{w}_i^{(s)}$ 
17:      for  $j \in \{1, \dots, p\}$  do
18:         $\tilde{\beta}_j^{(t)}(\lambda_r) \leftarrow S \left( \frac{1}{n} \sum_{i=1}^n \tilde{w}_i^{(s)} X_{ij} \left( \tilde{z}_i^{(s)} - \sum_{k < j} X_{ik} \tilde{\beta}_k^{(t)}(\lambda_r) - \sum_{k > j} X_{ik} \tilde{\beta}_k^{(t-1)}(\lambda_r) \right), \lambda_r \right) / \frac{1}{n} \sum_{i=1}^n \tilde{w}_i^{(s)} X_{ij}^2$ 
19:      end for
20:    end while
21:     $\tilde{\beta}_s(\lambda_r) \leftarrow \tilde{\beta}_s^{(t)}(\lambda_r)$ 
22:  end while
23:   $\hat{\beta}(\lambda_r) \leftarrow \tilde{\beta}_s(\lambda_r)$ 
24:   $\tilde{\beta}_0(\lambda_{r+1}) \leftarrow \hat{\beta}(\lambda_r)$ 
25: end for

```

Implementation in R

target functions needed to be optimized and soft-threshold operator

```
# 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.

```
# 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
  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
  s <- 0
  t <- 2
  while (t > 1 && s < maxiter) { # if number of iterations of inner loop = 1, converge.
    s <- s + 1
    betavec <- InnerLoop(X = X, z = z, w = w, betavec = betavec, lambda = lambda)
    t <- betavec[1]
    betavec <- betavec[-1]
    u <- X %*% betavec
  }
  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 -0.5175439      0      0.00000000      0      0
## [2,] 0.296934940 -0.5332517      0      0.00000000      0      0
## [3,] 0.223139865 -0.5811345      0      0.00000000      0      0
## [4,] 0.167684542 -0.6248298      0      0.00000000      0      0
## [5,] 0.126011126 -0.6651202      0      0.00000000      0      0
## [6,] 0.094694500 -0.6997942      0      0.00000000      0      0
## [7,] 0.071160766 -0.7261112      0      0.00000000      0      0
## [8,] 0.053475700 -0.7458971      0      0.00000000      0      0
## [9,] 0.040185774 -0.7529327      0      0.00000000      0      0
## [10,] 0.030198697 -0.7491925      0      0.04210221      0      0
## [11,] 0.022693635 -0.7319824      0      0.14375195      0      0
## [12,] 0.017053751 -0.7049084      0      0.24242756      0      0
## [13,] 0.012815507 -0.6674354      0      0.32654248      0      0
## [14,] 0.009630562 -0.6145893      0      0.40441986      0      0
## [15,] 0.007237149 -0.5476007      0      0.47397920      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 0.04380883
## [7,] 0 0 0 0.14896696
## [8,] 0 0 0 0.27457842
## [9,] 0 0 0 0.41580703
## [10,] 0 0 0 0.52796315
## [11,] 0 0 0 0.54981249
## [12,] 0 0 0 0.54102904
## [13,] 0 0 0 0.51052136
## [14,] 0 0 0 0.49609011
## [15,] 0 0 0 0.52501264
## 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 0.02349977 0 0
## [11,] 0 0 0.14647972 0 0
## [12,] 0 0 0.31040920 0 0
## [13,] 0 0 0.50573579 0 0
## [14,] 0 0 0.77052586 0 0
## [15,] 0 0 1.10645170 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 0.00000000
## [2,] 0 0.00000000 0.0000000 0.00000000
## [3,] 0 0.00000000 0.1405159 0.00000000
## [4,] 0 0.00000000 0.4035540 0.00000000
## [5,] 0 0.00000000 0.5735091 0.00000000
## [6,] 0 0.00000000 0.7443102 0.003652313
## [7,] 0 0.00000000 0.8916556 0.104960549
## [8,] 0 0.00000000 1.0584742 0.217862348
## [9,] 0 0.00000000 1.2872840 0.323021293
## [10,] 0 0.00000000 1.5422766 0.386454248

```

##	[11,]	0	0.00000000	1.7812263	0.390419292
##	[12,]	0	0.00000000	2.0380085	0.393812809
##	[13,]	0	0.00000000	2.3247578	0.409394032
##	[14,]	0	-0.04696304	2.5697707	0.431067222
##	[15,]	0	-0.15828748	2.7536193	0.461233496
##		perimeter_worst	area_worst	smoothness_worst	compactness_worst
##	[1,]	0.00000000	0	0.00000000	0
##	[2,]	0.1122777	0	0.00000000	0
##	[3,]	0.1142284	0	0.00000000	0
##	[4,]	0.00000000	0	0.00000000	0
##	[5,]	0.00000000	0	0.00000000	0
##	[6,]	0.00000000	0	0.00000000	0
##	[7,]	0.00000000	0	0.00000000	0
##	[8,]	0.00000000	0	0.00000000	0
##	[9,]	0.00000000	0	0.04096284	0
##	[10,]	0.00000000	0	0.11407151	0
##	[11,]	0.00000000	0	0.20136541	0
##	[12,]	0.00000000	0	0.29542037	0
##	[13,]	0.00000000	0	0.39791678	0
##	[14,]	0.00000000	0	0.49499762	0
##	[15,]	0.00000000	0	0.57906812	0
##		concavity_worst	concave.points_worst	symmetry_worst	
##	[1,]	0.00000000	0.00000000	0.00000000	
##	[2,]	0.00000000	0.3103703	0.00000000	
##	[3,]	0.00000000	0.5167995	0.00000000	
##	[4,]	0.00000000	0.7136204	0.00000000	
##	[5,]	0.00000000	0.8911671	0.00000000	
##	[6,]	0.00000000	1.0326396	0.00000000	
##	[7,]	0.00000000	1.1009757	0.00000000	
##	[8,]	0.00000000	1.1353403	0.01531455	
##	[9,]	0.00000000	1.0722904	0.08031214	
##	[10,]	0.00000000	1.0054745	0.14640343	
##	[11,]	0.00000000	0.9992111	0.21307821	
##	[12,]	0.01895635	1.0019598	0.27213031	
##	[13,]	0.06841130	0.9950544	0.32166477	
##	[14,]	0.15287364	1.0001782	0.36226161	
##	[15,]	0.27638290	1.0211611	0.39494122	
##		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			