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

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

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 $\tilde{\mathbf{p}} = (\tilde{p}_1, \ldots, \tilde{p}_n)^\top$ and $\tilde{\mathbf{W}} = \mathrm{diag}(\tilde{w}_1, \ldots, \tilde{w}_n)$ 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:

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

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

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

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

```
# 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(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,] 3.979882e-01  -0.5175439    0.000000   0.00000000        0.00000
##  [2,] 1.896993e-01  -0.5373322    0.000000   0.00000000        0.00000
##  [3,] 9.041927e-02  -0.6654039    0.000000   0.00000000        0.00000
##  [4,] 4.309793e-02  -0.7408071    0.000000   0.00000000        0.00000
##  [5,] 2.054243e-02  -0.7562361    0.000000   0.14437376        0.00000
##  [6,] 9.791455e-03  -0.7126412    0.000000   0.34977650        0.00000
##  [7,] 4.667052e-03  -0.5923240    0.000000   0.58621105        0.00000
##  [8,] 2.224529e-03  -0.4457237    0.000000   0.54203824        0.00000
##  [9,] 1.060311e-03  -0.1993582    0.000000   0.28042449        0.00000
## [10,] 5.053925e-04   0.1348887    0.000000   0.00000000        0.00000
## [11,] 2.408930e-04   0.7396742   -2.277184   0.00000000        0.00000
## [12,] 1.148205e-04   1.5664674   -5.613223   0.00000000        0.00000
## [13,] 5.472868e-05   2.2973504   -8.190091   0.01819084        0.00000
## [14,] 2.608617e-05   3.6495909  -11.044585   0.45518115        0.00000
## [15,] 1.243385e-05   5.3159276  -25.807710   1.23486622        0.00000
## [16,] 5.926539e-06   6.8087761  -67.609909   1.78884681       15.49049
## [17,] 2.824858e-06   8.1521530 -105.629152   2.28091051       22.41391
## [18,] 1.346455e-06   9.2023317 -151.048617   2.48422441       45.50482
## [19,] 6.417818e-07  10.0482042 -207.559965   2.45992249       85.30488
## [20,] 3.059023e-07  10.8028427 -242.096207   2.60320892      101.83832
```

```
##       area_mean smoothness_mean compactness_mean concavity_mean
##  [1,]   0.00000       0.0000000        0.0000000     0.00000000
##  [2,]   0.00000       0.0000000        0.0000000     0.00000000
##  [3,]   0.00000       0.0000000        0.0000000     0.00000000
##  [4,]   0.00000       0.0000000        0.0000000     0.00000000
##  [5,]   0.00000       0.0000000        0.0000000     0.00000000
##  [6,]   0.00000       0.0000000        0.0000000     0.00000000
##  [7,]   0.00000       0.0000000        0.0000000     0.00000000
##  [8,]   0.00000       0.0000000        0.0000000     0.00000000
##  [9,]   0.00000       0.0000000       -0.5178377     0.04612443
## [10,]   0.00000       0.0000000       -1.5946550     0.83837452
## [11,]   0.00000       0.1712791       -3.0004187     2.24273010
## [12,]   0.00000       1.0056591       -4.9252542     4.37170022
## [13,]   0.00000       1.8382951       -6.8824176     6.92165209
## [14,]   0.00000       2.5021254       -8.8880462     8.83327377
## [15,]  12.61793       3.2196336      -11.3775366     9.96191119
## [16,]  36.92705       4.2786797      -16.3643378     8.08418176
## [17,]  64.07145       6.1300625      -21.6963848     7.92535552
## [18,]  83.91697       8.1633512      -26.4718296     8.23183232
## [19,]  98.99004      10.4769511      -30.5230022     8.15571705
## [20,] 115.20441      12.3004569      -34.2789002     8.99112780
##       concave.points_mean symmetry_mean fractal_dimension_mean  radius_se
##  [1,]          0.00000000    0.00000000             0.00000000  0.0000000
##  [2,]          0.00000000    0.00000000             0.00000000  0.0000000
##  [3,]          0.03976418    0.00000000             0.00000000  0.0000000
##  [4,]          0.25111458    0.00000000             0.00000000  0.0000000
##  [5,]          0.42620806    0.00000000             0.00000000  0.1075042
##  [6,]          0.51030193    0.00000000             0.00000000  0.3890802
##  [7,]          0.70942328    0.00000000            -0.05850471  1.0824618
##  [8,]          0.98607296    0.00000000            -0.19718217  1.8340419
##  [9,]          1.51728455    0.00000000            -0.10601278  2.6715817
## [10,]          1.90175761    0.00000000             0.00000000  3.5398685
## [11,]          2.19177657    0.00000000             0.01981274  3.0327661
## [12,]          1.78058277   -0.08079907             0.12314962  0.4379303
## [13,]          1.31181611   -0.27593354             0.22150393  0.0000000
## [14,]          1.64429739   -0.50169244             0.05941753  0.1528700
## [15,]          2.93670109   -0.67975234            -0.22692339  4.5955991
## [16,]          7.08012422   -0.89926447            -0.54582792 21.0836302
## [17,]         11.16383767   -1.29875723            -0.70229904 33.0260593
## [18,]         13.44871223   -1.88783263            -0.96868980 43.9740608
## [19,]         14.93661279   -2.91920611            -1.61500066 55.7786624
## [20,]         16.35378227   -3.64173049            -1.79064042 64.4923295
##       texture_se perimeter_se   area_se smoothness_se compactness_se
##  [1,] 0.00000000    0.0000000  0.000000     0.0000000     0.00000000
##  [2,] 0.00000000    0.0000000  0.000000     0.0000000     0.00000000
##  [3,] 0.00000000    0.0000000  0.000000     0.0000000     0.00000000
##  [4,] 0.00000000    0.0000000  0.000000     0.0000000     0.00000000
##  [5,] 0.00000000    0.0000000  0.000000     0.0000000     0.00000000
##  [6,] 0.00000000    0.0000000  0.000000     0.0000000     0.00000000
##  [7,] 0.00000000    0.0000000  0.000000     0.0000000    -0.04564155
##  [8,] -0.08839658   0.0000000  0.000000     0.1984570    -0.41775123
##  [9,] -0.33682875   0.0000000  0.000000     0.4383099    -0.67801338
## [10,] -0.61866868   0.0000000  0.000000     0.5856049    -0.70242549
## [11,] -0.66649754  -0.5283341  2.499965     0.5753226    -0.42138343
```

```
## [12,] -0.75101017   -1.2787922  9.089037     0.8847592     0.46169113
## [13,] -0.96419122   -3.8973222 15.932230     1.1356654     2.41472024
## [14,] -1.21141290   -9.4162040 28.207013     0.9483941     4.94832408
## [15,] -1.64245057  -19.2689396 39.886217     0.1951945     8.23410204
## [16,] -2.29007517  -39.7510710 45.528140    -1.6557396    13.07090418
## [17,] -3.08305776  -55.4919004 51.778074    -2.8125701    17.54567589
## [18,] -3.71526522  -68.9883064 56.613378    -3.5167138    22.44000405
## [19,] -4.13260530  -81.7911236 57.683203    -3.9079902    28.29843595
## [20,] -4.57274743  -91.4521587 60.026234    -4.2544868    32.10837247
##       concavity_se concave.points_se  symmetry_se fractal_dimension_se
##  [1,]   0.00000000         0.0000000   0.00000000           0.00000000
##  [2,]   0.00000000         0.0000000   0.00000000           0.00000000
##  [3,]   0.00000000         0.0000000   0.00000000           0.00000000
##  [4,]   0.00000000         0.0000000   0.00000000           0.00000000
##  [5,]   0.00000000         0.0000000   0.00000000           0.00000000
##  [6,]   0.00000000         0.0000000   0.00000000          -0.01090972
##  [7,]   0.00000000         0.0000000  -0.07484541          -0.17323658
##  [8,]  -0.00807579         0.0000000  -0.17800716          -0.16333405
##  [9,]  -0.16166446         0.0000000  -0.21568581          -0.17743174
## [10,]  -0.49011657         0.2504996  -0.23779142          -0.48518578
## [11,]  -1.16307391         1.3664593  -0.37393054          -1.45613057
## [12,]  -2.09921518         2.8689782  -0.50789356          -4.12887619
## [13,]  -3.41953181         4.3873815  -1.02333389          -7.53478770
## [14,]  -5.01879157         6.3812671  -1.98908482         -11.10037913
## [15,]  -7.80779535         9.5032086  -3.38506836         -14.34528701
## [16,] -12.38291124        14.8746462  -5.40249983         -16.70062717
## [17,] -17.08954297        20.6753756  -7.51947093         -20.61837068
## [18,] -21.63410622        25.5691333  -9.29678946         -25.26999828
## [19,] -26.38146149        30.0527779 -11.13016643         -30.61724341
## [20,] -30.02477558        33.8559779 -12.55900136         -34.66616201
##       radius_worst texture_worst perimeter_worst area_worst smoothness_worst
##  [1,]    0.0000000    0.00000000        0.000000   0.000000       0.00000000
##  [2,]    0.3322387    0.00000000        0.000000   0.000000       0.00000000
##  [3,]    0.6345494    0.02927027        0.000000   0.000000       0.00000000
##  [4,]    1.0000412    0.21208717        0.000000   0.000000       0.03560325
##  [5,]    1.4922570    0.29172604        0.000000   0.000000       0.16958852
##  [6,]    2.0759271    0.32634542        0.000000   0.000000       0.35484113
##  [7,]    2.3866011    0.32517637        0.000000   0.000000       0.52822887
##  [8,]    2.7471912    0.63640402        0.000000   0.000000       0.56913836
##  [9,]    3.2755646    1.24042636        0.000000   0.000000       0.56464705
## [10,]    3.9495483    1.87507184        0.000000   0.000000       0.67782468
## [11,]    6.9737449    2.09438087        0.000000   0.000000       0.67649915
## [12,]   10.8112097    2.41828009        0.000000   0.000000       0.00000000
## [13,]   11.0181128    2.95555932        2.966463   0.000000      -0.56062735
## [14,]    5.0309363    3.26349501       11.821649   0.000000      -0.75098244
## [15,]    0.0000000    3.65544386       25.583509  -5.000182      -0.66151991
## [16,]  -13.1768207    4.85998840       55.974870 -12.478858      -0.06859582
## [17,]  -14.8079223    6.50476454       80.679095 -24.376244      -0.37651682
## [18,]  -16.1623247    8.04808630       98.030143 -30.804857      -1.17071222
## [19,]  -22.1483340    9.37105802      112.698391 -29.425009      -2.59714883
## [20,]  -22.4994996   10.61444715      125.618828 -34.389817      -3.46802125
##       compactness_worst concavity_worst concave.points_worst symmetry_worst
##  [1,]          0.000000      0.00000000            0.0000000     0.00000000
##  [2,]          0.000000      0.00000000            0.5788365     0.00000000
```

```
##  [3,]          0.000000      0.00000000            0.9343097     0.00000000
##  [4,]          0.000000      0.00000000            1.0313024     0.05947929
##  [5,]          0.000000      0.00000000            0.9933919     0.19221250
##  [6,]          0.000000      0.06951697            0.9370008     0.32112945
##  [7,]          0.000000      0.35177696            0.8757576     0.45669910
##  [8,]          0.000000      0.74876872            0.9838449     0.65875115
##  [9,]          0.000000      1.21299983            1.1241085     0.80173601
## [10,]          0.000000      1.28939242            1.1183336     0.92856939
## [11,]          0.000000      1.21383601            0.4551907     1.12001122
## [12,]          0.000000      0.94222656            0.0000000     1.41193545
## [13,]         -1.536309      0.92062037            0.0000000     2.15302520
## [14,]         -4.490199      1.61661491           -0.2274300     3.30916961
## [15,]         -8.292990      3.84868562           -0.6999014     4.64191139
## [16,]        -13.870783      9.45251059           -3.1303028     6.49282271
## [17,]        -18.027017     14.72100399           -6.5129162     8.87253910
## [18,]        -23.142552     19.64858254           -8.4132960    11.01798191
## [19,]        -30.025706     24.89895117           -9.8529279    13.54659666
## [20,]        -33.818421     28.56782722          -11.0125327    15.48841638
##       fractal_dimension_worst
##  [1,]               0.0000000
##  [2,]               0.0000000
##  [3,]               0.0000000
##  [4,]               0.0000000
##  [5,]               0.0000000
##  [6,]               0.0000000
##  [7,]               0.0000000
##  [8,]               0.0000000
##  [9,]               0.1062373
## [10,]               0.6464210
## [11,]               1.6744345
## [12,]               3.4190278
## [13,]               5.6602792
## [14,]               8.5099491
## [15,]              11.3910219
## [16,]              14.1601527
## [17,]              17.1304925
## [18,]              20.6489240
## [19,]              24.9073453
## [20,]              27.7525670
```