Tinkering

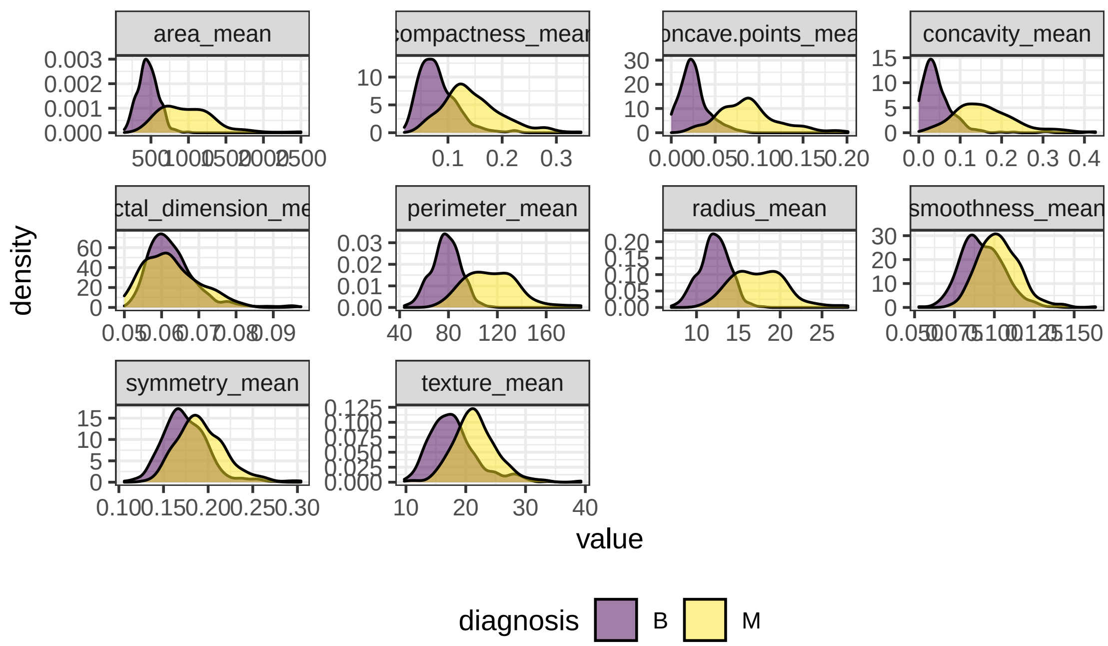
Waveley Qiu (wq2162)

2022-03-17

EDA

Let's import and take a look at the data.

Let's take a look at the distributions of other variables.




```
tbl_summary(bc, by = diagnosis)
```

```
## Table printed with `knitr::kable()`, not {gt}. Learn why at
## https://www.danieldsjoberg.com/gtsummary/articles/rmarkdown.html
## To suppress this message, include `message = FALSE` in code chunk header.
```

Characteristic	**B**, N = 357	**M**, N = 212
id	908,916 (874,662, 8,812,816)	895,366 (861,345, 8,911,290)
radius_mean	12.2 (11.1, 13.4)	17.3 (15.1, 19.6)
texture_mean	17.4 (15.2, 19.8)	21.5 (19.3, 23.8)
perimeter_mean	78 (71, 86)	114 (99, 130)
area_mean	458 (378, 551)	932 (705, 1,204)
smoothness_mean	0.091 (0.083, 0.101)	0.102 (0.094, 0.111)
compactness_mean	0.08 (0.06, 0.10)	0.13 (0.11, 0.17)
concavity_mean	0.04 (0.02, 0.06)	0.15 (0.11, 0.20)
concave.points_mean	0.02 (0.02, 0.03)	0.09 (0.06, 0.10)

Characteristic	**B**, N = 357	**M**, N = 212
symmetry_mean	0.171 (0.158, 0.189)	0.190 (0.174, 0.210)
fractal_dimension_mean	0.062 (0.059, 0.066)	0.062 (0.057, 0.067)
radius_se	0.26 (0.21, 0.34)	0.55 (0.39, 0.76)
texture_se	1.11 (0.80, 1.49)	1.10 (0.89, 1.43)
perimeter_se	1.85 (1.45, 2.39)	3.68 (2.72, 5.21)
area_se	20 (15, 25)	58 (36, 94)
smoothness_se	0.0065 (0.0052, 0.0085)	0.0062 (0.0051, 0.0080)
compactness_se	0.016 (0.011, 0.026)	0.029 (0.020, 0.039)
concavity_se	0.018 (0.011, 0.031)	0.037 (0.027, 0.050)
concave.points_se	0.009 (0.006, 0.012)	0.014 (0.011, 0.017)
symmetry_se	0.019 (0.016, 0.024)	0.018 (0.015, 0.022)
fractal_dimension_se	0.0028 (0.0021, 0.0042)	0.0037 (0.0027, 0.0049)
radius_worst	13.3 (12.1, 14.8)	20.6 (17.7, 23.8)
texture_worst	22.8 (19.6, 26.5)	28.9 (25.8, 32.7)
perimeter_worst	87 (78, 97)	138 (119, 160)
area_worst	547 (447, 670)	1,303 (970, 1,713)
smoothness_worst	0.125 (0.110, 0.138)	0.143 (0.130, 0.156)
compactness_worst	0.17 (0.11, 0.23)	0.36 (0.24, 0.45)
concavity_worst	0.14 (0.08, 0.22)	0.40 (0.33, 0.56)
concave.points_worst	0.07 (0.05, 0.10)	0.18 (0.15, 0.21)
symmetry_worst	0.27 (0.24, 0.30)	0.31 (0.28, 0.36)
fractal_dimension_worst	0.077 (0.070, 0.085)	0.088 (0.076, 0.103)

We want our outcome variable to be binary. Let's create a new outcome variable, `bin_out`, which will be 1 if `diagnosis == 'M'` and 0 if 'diagnosis == 'B'.

```
bc <- bc %>% mutate(bin_out = ifelse(diagnosis == "M", 1, 0)) %>% relocate(bin_out)
```

Full Model

First, we want to establish logistic model using all variables in the dataset. We will do this by performing a Newton Raphson optimization in order to find the MLEs of the beta coefficients.

The likelihood function for a logistic model is defined as follows:

$$f(\beta_0, \beta_1, ..., \beta_{30}) = \sum_{i=1}^{n} \left(Y_i \left(\beta_0 + \sum_{j=1}^{30} \beta_j x_{ij} \right) - \log(1 + e^{\left(\beta_0 + \sum_{j=1}^{30} \beta_j x_{ij}\right)} \right)$$

Let $\pi_i = \frac{e^{\beta_0 + \sum_{j=1}^{30} \beta_j x_{ij}}}{1+e^{\beta_0 + \sum_{j=1}^{30} \beta_j x_{ij}}}$. Then, the gradient of this function is defined as follows:

$$\nabla f(\beta_0, \beta_1, ..., \beta_{30}) = \begin{pmatrix} \sum_{i=1}^{n} Y_i - \pi_i \\ \sum_{i=1}^{n} x_{i1}(Y_i - \pi_i) \\ \sum_{i=1}^{n} x_{i2}(Y_i - \pi_i) \\ \vdots \\ \sum_{i=1}^{n} x_{i30}(Y_i - \pi_i) \end{pmatrix}$$

Finally, we define the Hessian of this function as follows:

$$\nabla^2 f(\beta_0, \beta_1, ..., \beta_{30}) = -\sum_{i=1}^{n} \begin{pmatrix} 1 \\ x_{i1} \\ x_{i2} \\ \vdots \\ x_{i30} \end{pmatrix} (1 \; x_{i1} \; x_{i2} \; \dots \; x_{i30}) \, \pi_i(1-\pi_i)$$

$$= -\begin{pmatrix} \sum_{i=1}^n \pi_i(1-\pi_i) & \sum_{i=1}^n x_{i1}\pi_i(1-\pi_i) & \dots & \sum_{i=1}^n x_{i30}\pi_i(1-\pi_i) \\ \sum_{i=1}^n x_{i1}\pi_i(1-\pi_i) & \sum_{i=1}^n x_{i1}^2\pi_i(1-\pi_i) & \dots & \sum_{i=1}^n x_{i30}x_{i1}\pi_i(1-\pi_i) \\ \sum_{i=1}^n x_{i2}\pi_i(1-\pi_i) & \sum_{i=1}^n x_{i1}x_{i2}\pi_i(1-\pi_i) & \dots & \sum_{i=1}^n x_{i30}x_{i2}\pi_i(1-\pi_i) \\ \vdots & \ddots & \ddots & \vdots \\ \sum_{i=1}^n x_{i30}\pi_i(1-\pi_i) & \sum_{i=1}^n x_{i1}x_{i30}\pi_i(1-\pi_i) & \dots & \sum_{i=1}^n x_{i30}^2\pi_i(1-\pi_i) \end{pmatrix}$$

$$= (1 \; x_{i1} \; x_{i2} \; \dots \; x_{i30}) \, I(\pi_i(1-\pi_i)) \begin{pmatrix} 1 \\ x_{i1} \\ x_{i2} \\ \vdots \\ x_{i30} \end{pmatrix}$$

Let's create a function that produces the log-likelihood, gradient vector, and hessian matrix, given a dataset and beta vector:

```
rep_col <- function(x, n){
  matrix(rep(x, each = n), ncol = n, byrow = TRUE)
}

logistic_stuff <- function(dat, beta){

  x <- dat[[1]] %>% unname() %>% as.matrix()
  y <- dat[[2]] %>% unname() %>% as.matrix()

  x_with_1 <- cbind(1, x)

  u <- x_with_1 %*% beta
 # return(u)

  expu <- exp(u)

  loglik <- sum(y*u - log(1 + expu))

  p <- expu/(1 + expu)
  #  return(p)
  # return(p)
  grad <- t(x_with_1) %*% (y - p)

  i_mat <- diag(nrow(p))
  diag(i_mat) <- p*(1 - p)

  hess <- -(t(x_with_1) %*% i_mat %*% x_with_1)
  return(list(
    loglik = loglik,
    grad = grad,
    hess = hess
```

```
  ))
}
```

Newton-Raphson Algorithm

Now, let's write a Newton-Raphson algorithm to find the beta coefficients that maximize this function's likelihood.

The unmodified estimate of $\theta_i = \begin{bmatrix} \beta_0 \\ \vdots \\ \beta_3 0 \end{bmatrix}$ at each step i of the Newton Raphson algorithm is:

$$\theta_i = \theta_{i-1} - [\nabla^2 f(\theta_{i-1})]^{-1} \nabla f(\theta_{i-1})$$

In this modified Newton Raphson algorithm, we want to first ensure that the $\nabla^2 f(\theta_{i-1})$ is either negative definite or replaced with a similar matrix that is negative definite. To do this, we will update the algorithm to be as follows:

$$\theta_i = \theta_{i-1} - [\nabla^2 f(\theta_{i-1}) - kI]^{-1} \nabla f(\theta_{i-1}),$$

where I is the identity matrix and k is a constant that allows $\nabla^2 f(\theta_{i-1}) - kI$ to be negative definite. If $\nabla^2 f(\theta_{i-1})$ is already negative definite, k will be 0.

Next, we want to add step-halving into our algorithm. We will proceed as follows:

$$\theta_i = \theta_{i-1} - \frac{1}{2^j} [\nabla^2 f(\theta_{i-1}) - kI]^{-1} \nabla f(\theta_{i-1}),$$

where j is chosen in a stepwise fashion, until $f(\theta_i) > f(\theta_{i-1})$.

```
NewtonRaphson <- function(dat, func, start, tol = 1e-8, maxiter = 200) {
  i <- 0
  cur <- start
  stuff <- func(dat, cur)
  res <- c(0, stuff$loglik, cur)
  prevloglik <- -Inf

  while (i < maxiter && abs(stuff$loglik - prevloglik) > tol && !is.na(stuff$loglik)) {
    i <- i + 1
    prevloglik <- stuff$loglik
    prev <- cur
    newhess <- ((stuff$hess + t(stuff$hess))/2)

    if (!is.negative.definite(newhess)) { # redirection
     while (!is.negative.definite(newhess)) {
       # subtracts identity matrix until a negative definite matrix is achieved
        newhess1 <- newhess - 0.0001*diag(31)
       # sanity check print("changing ascent direction")
        newhess <- ((newhess1 + t(newhess1))/2)
      }
    }

    cur <- prev - solve(newhess) %*% stuff$grad
    stuff <- func(dat, cur)
```

```
    if (stuff$loglik < prevloglik) {  # back tracking (half-step)
      j = 1
      while (stuff$loglik < prevloglik & (!is.na(stuff$loglik))) {
         halfstep = 1/(2^j)
         cur <- prev - halfstep*solve(newhess) %*% stuff$grad
         stuff <- func(dat, cur)
        # sanity check print("backtracking")
         j = j + 1
      }
    }
    res <- rbind(res, c(i, stuff$loglik, cur))
  }
  return(res)
}
```

Let's start with all beta coefficients being 0.001.

```
beta_init <- rep(0.001, 31) %>% as.matrix()

test1 <- logistic_stuff(
  list(x = bc[,-c(1,2, 3)] %>% as.matrix(),
       y = bc$bin_out %>% as.matrix()),
  beta = beta_init)

ans <- NewtonRaphson(
      list(x = bc[,-c(1,2, 3)] %>% as.matrix(),
       y = bc$bin_out %>% as.matrix()),
       logistic_stuff,
       beta_init)
```

The beta estimates are as follows:

```
if (sum(is.na(ans[nrow(ans),])) > 0) {
  beta_est <- ans[nrow(ans) - 1, -c(1,2)]
}

if (sum(is.na(ans[nrow(ans),])) == 0) {
  beta_est <- ans[nrow(ans), -c(1,2)]
}

tibble(beta_subscript = seq(0, 30), beta_estimates = beta_est) %>% knitr::kable()
```

beta_subscript	beta_estimates
0	-35.6788855
1	-25.8450679
2	0.2364029
3	0.5233921
4	0.1930977
5	374.2231831
6	-274.0661692
7	112.3745441
8	197.4735980
9	-81.7535878
10	159.0248149

beta_subscript	beta_estimates
11	-8.7382958
12	-3.6859107
13	-3.5505959
14	0.6320060
15	229.4048089
16	491.9823911
17	-394.7461298
18	1835.8287537
19	-331.3771487
20	-5753.5678242
21	9.2644874
22	0.8250521
23	0.1906329
24	-0.0454310
25	-131.4051756
26	-43.7531684
27	47.8195818
28	-22.8843059
29	80.0735290
30	555.5843554

Logistic-Lasso Model

Now, we want to establish a logistic-lasso model, in which we want to minimize the weighted residual sum of squares of the logistic regression.

$$f(\beta_0,\beta_1,\ldots,\beta_{30}) \approx -\frac{1}{2n}\sum_{i=1}^{n} w_i \left(z_i - \begin{pmatrix}\mathbf{1} & \mathbf{x_i}\end{pmatrix}\begin{pmatrix}\beta_0\\ \beta_1\\ \vdots\\ \beta_{30}\end{pmatrix}\right)^2 + C(\tilde{\beta_0},\tilde{\beta_1},\ldots,\tilde{\beta_{30}})$$

$$z_i = \begin{pmatrix}\mathbf{1} & \mathbf{x_i}\end{pmatrix}\begin{pmatrix}\tilde{\beta_0}\\ \tilde{\beta_1}\\ \vdots\\ \tilde{\beta_{30}}\end{pmatrix} + \frac{\mathbf{y_i} - \tilde{\mathbf{p}}_\mathbf{i}(\mathbf{x_i})}{\tilde{\mathbf{p}}_\mathbf{i}(\mathbf{x_i})(\mathbf{1}-\tilde{\mathbf{p}}_\mathbf{i}(\mathbf{x_i}))}$$

$$w_i = \tilde{p_i}(x_i)(1-\tilde{p_i}(x_i))$$

$$\tilde{p_i}(x_i) = \frac{\exp\left(\begin{pmatrix}\mathbf{1} & \mathbf{x_i}\end{pmatrix}\begin{pmatrix}\tilde{\beta_0}\\ \tilde{\beta_1}\\ \vdots\\ \tilde{\beta_{30}}\end{pmatrix}\right)}{1+\exp\left(\begin{pmatrix}\mathbf{1} & \mathbf{x_i}\end{pmatrix}\begin{pmatrix}\tilde{\beta_0}\\ \tilde{\beta_1}\\ \vdots\\ \tilde{\beta_{30}}\end{pmatrix}\right)}$$

Quadratic Approximation to the Log-likelihood

```r
quad_loglik <- function(dat, beta){ # beta vector includes beta_0

  x <- dat[[1]] %>% unname() %>% as.matrix()
  y <- dat[[2]] %>% unname() %>% as.matrix()


  x_with_1 <- cbind(1, x)

  u <- x_with_1 %*% beta
  expu <- exp(u)

  p <- expu/(1 + expu) # estimated outcome probability
  w_i <- p * (1 - p)  # weights

  z_i <- x_with_1 %*% beta + (y - p)/(p * (1 - p)) # working response

  loglik <- -(1/(2*nrow(x))) * t(w_i) %*% ((z_i - x_with_1 %*% beta)^2)

  return(loglik)
}
```

```r
beta_init <- rep(0.001, 31) %>% as.matrix()

test_quad <- quad_loglik(
  list(x = bc[,-c(1,2, 3)] %>% as.matrix(),
       y = bc$bin_out %>% as.matrix()),
  beta = beta_init)

test_quad
```

```
##           [,1]
## [1,] -1.210166
```

Lasso Minimization

We want to achieve the following minimization:

$$\min_{(\beta_0,\beta_1,\ldots,\beta_{30})} L(\beta_0,\beta_1,\ldots,\beta_{30},\lambda) = \left\{ -l(\beta_0,\beta_1,\ldots,\beta_{30}) + \lambda \sum_{j=0}^{30} |\beta_j| \right\}$$