Tinkering

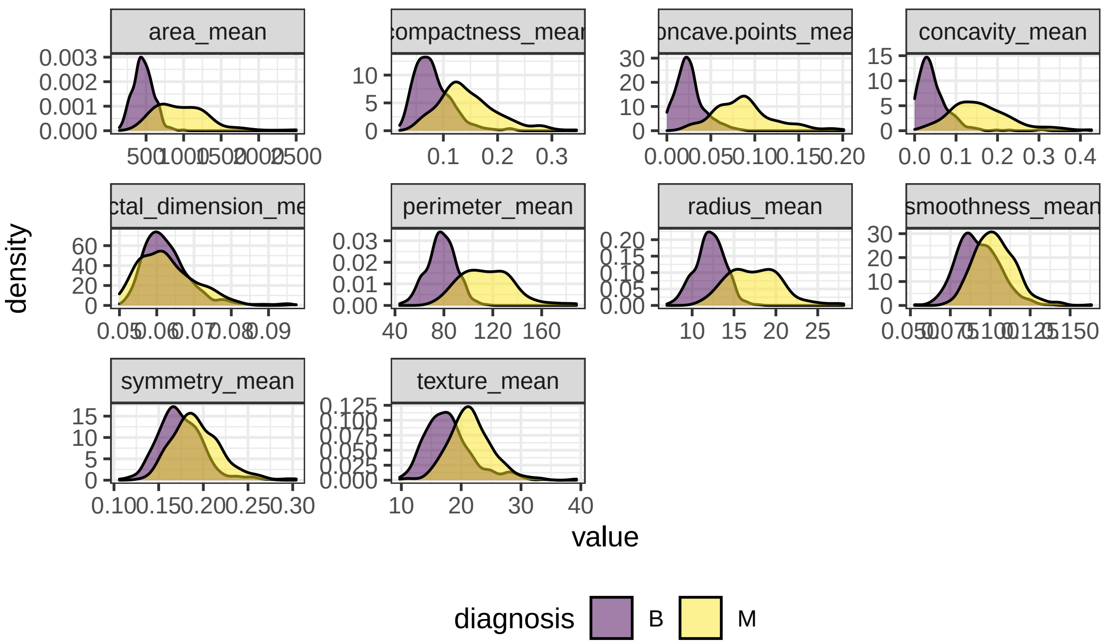
Waveley Qiu (wq2162)

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:

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

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 - diag(nrow(newhess))
       # 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(0.0000001, 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)
ans
```

```
##     [,1]       [,2]        [,3]        [,4]        [,5]       [,6]         [,7]
## res    0 -394.39362   0.0000001   0.0000001  0.00000010 0.00000010  0.000000100
##        1 -134.65976 -10.0872468  -0.8710882  0.01818188 0.09495944  0.001271339
##        2  -77.34464 -17.9619834  -0.8285658  0.01145304 0.12245311 -0.000105099
##        3  -50.21369 -26.9936742  -1.1212145 -0.02369701 0.23690998 -0.002931911
##        4  -36.03778 -38.1724054  -2.6041585 -0.06359077 0.50900004 -0.005061973
##        5  -27.91930 -51.7602164  -4.9812918 -0.08528592 0.83713654 -0.004087297
##        6  -21.31729 -59.2805864  -5.4368164 -0.07605913 0.78438916 -0.006234764
##        7  -17.69324 -69.8922197  -7.6468871 -0.07624811 0.80863738  0.004163781
##        8  -15.48629 -71.2350788 -12.9872758 -0.02857948 0.87690698  0.046230391
##        9  -13.69064 -54.4945746 -22.0825765  0.12796151 0.75735139  0.141505122
##       10        NaN         NaN         NaN         NaN        NaN          NaN
##            [,8]         [,9]       [,10]       [,11]       [,12]       [,13]
## res   0.0000001    0.0000001   0.0000001   0.0000001   0.0000001   0.0000001
##       0.3387568  -16.8881411   5.5919894   8.5673321   0.4108367   0.1330465
##       5.6757806  -26.6680623  11.2380676  12.3078592  -0.2566558  -7.1708961
##      11.3740176  -35.3945319  15.4028252  17.0862948  -2.1786950 -13.0830842
##      19.8555809  -48.3871269  18.7540084  21.5583888  -4.2205064  -5.3281997
##      46.6685834  -70.5251252  23.9497692  25.4441459  -6.1144340  20.3402698
##     108.6440390  -95.0994443  37.0461972  43.8561061 -13.1302998  34.9821460
##     181.1114671 -134.6026859  58.0610640  65.0149899 -24.3834378  68.9058563
##     275.0526125 -189.4998594  77.5314340  97.0507502 -42.8927194 106.7262977
```

```
##     380.9556133 -259.0116909 102.0427663 159.7497196 -74.4211209 129.7876054
##             NaN          NaN         NaN         NaN         NaN         NaN
##          [,14]       [,15]       [,16]        [,17]       [,18]       [,19]
## res  0.0000001  0.00000010  0.00000010  0.000000100   0.0000001   0.0000001
##      1.7398237 -0.02703389 -0.09008103 -0.003692872  63.4172830   0.2596137
##      4.1261474 -0.25563038 -0.07523596 -0.014421133  98.9603410   4.6588336
##      8.6867250 -0.68939750 -0.10325967 -0.034241669 118.2843551  15.0260838
##     14.8138015 -1.11008752 -0.35726313 -0.054877673 124.7603205  39.5387996
##     18.3167708 -1.42295913 -0.71486521 -0.046821793 159.6615067  88.1515317
##      8.3856675 -1.68421366 -0.81995621  0.080668201 299.4056738 149.7514689
##      4.9114530 -2.29249816 -1.16256488  0.174352853 429.8802608 234.7173251
##      5.6659937 -2.92234721 -2.01990720  0.267084155 512.3284629 344.9368015
##      3.5901441 -3.74642322 -3.63071793  0.482956031 430.8702214 488.2357296
##            NaN         NaN         NaN          NaN         NaN         NaN
##            [,20]        [,21]        [,22]         [,23]     [,24]      [,25]
## res    0.0000001    0.0000001    0.0000001     0.0000001 0.0000001 0.00000010
##      -14.2618721   42.2718056    6.7893627   -28.5857616 0.7807325 0.02863750
##      -24.0803502   67.0924393    1.6326042   -70.0992306 0.9976320 0.08506012
##      -31.1252061   95.0608994   -8.4308087  -201.4592786 1.0003382 0.18302816
##      -38.1781089  146.9805763  -15.6627346  -513.8715773 1.0857117 0.28882249
##      -52.6230114  264.4364257  -31.6631207 -1163.0396598 1.5909969 0.37956273
##      -82.6347245  477.5853916  -79.0589252 -2175.9245545 2.0469205 0.43983560
##     -134.7771551  724.2681761 -127.9576973 -3253.1310952 3.1915222 0.56749404
##     -218.3572251 1058.9298766 -176.1470005 -4447.3942241 4.8101374 0.70511926
##     -347.5712439 1591.5134547 -288.3578551 -5786.0403060 7.5707296 0.83799591
##              NaN          NaN          NaN           NaN       NaN        NaN
##            [,26]        [,27]        [,28]       [,29]      [,30]       [,31]
## res  0.000000100  0.000000100    0.0000001   0.0000001  0.0000001   0.0000001
##     -0.009740202 -0.004044893    2.1714274   0.2686332  1.5247648   1.8572396
##     -0.014191928 -0.005368530    2.2515131  -0.3200753  2.6882829   2.6683360
##     -0.011949583 -0.005673658    4.0146921  -2.9205673  4.4838701   2.4216228
##      0.011233421 -0.006656244    7.5834183  -7.1469371  6.6506824   1.2240037
##      0.033297825 -0.009143438    2.3849995 -12.1114420  9.1350504  -2.5825743
##      0.015671046 -0.003591089  -31.8112429 -16.1186450 11.6334714 -10.8301760
##      0.022478388 -0.004181932  -66.5179970 -23.2845589 16.7156625 -14.4921225
##      0.083549133 -0.012228085 -104.3394801 -32.4480905 27.3596866 -13.6041150
##      0.213184845 -0.035171046 -141.0451991 -44.2979594 43.9152919 -14.9041251
##              NaN          NaN          NaN         NaN        NaN         NaN
##          [,32]       [,33]
## res  0.0000001   0.0000001
##      2.2271502  17.2139324
##      5.2824566  30.1670200
##      8.9195137  49.2799147
##     11.7968361  78.5449600
##     14.6627973 134.5606633
##     21.8678929 226.9043303
##     32.5777962 324.7602336
##     46.2116667 429.4285398
##     70.4772028 553.3210084
##            NaN         NaN
```

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	-54.4945746
1	-22.0825765
2	0.1279615
3	0.7573514
4	0.1415051
5	380.9556133
6	-259.0116909
7	102.0427663
8	159.7497196
9	-74.4211209
10	129.7876054
11	3.5901441
12	-3.7464232
13	-3.6307179
14	0.4829560
15	430.8702214
16	488.2357296
17	-347.5712439
18	1591.5134547
19	-288.3578551
20	-5786.0403060
21	7.5707296
22	0.8379959
23	0.2131848
24	-0.0351710
25	-141.0451991
26	-44.2979594
27	43.9152919
28	-14.9041251
29	70.4772028
30	553.3210084

GLM

```
glm(bin_out ~ ., data = bc[,-c(2, 3)], family="binomial")
```

```
## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

##
## Call:  glm(formula = bin_out ~ ., family = "binomial", data = bc[, -c(2,
##     3)])
##
```

```
## Coefficients:
##             (Intercept)              radius_mean             texture_mean
##              -2.881e+06                2.427e+06                1.958e+05
##          perimeter_mean                area_mean          smoothness_mean
##               1.473e+06               -1.301e+05               -1.525e+08
##        compactness_mean           concavity_mean      concave.points_mean
##              -6.428e+06                1.042e+06               -1.716e+07
##           symmetry_mean   fractal_dimension_mean                radius_se
##               4.049e+07               -4.233e+07                3.328e+07
##              texture_se             perimeter_se                  area_se
##               6.368e+06                1.701e+06               -6.393e+05
##           smoothness_se           compactness_se             concavity_se
##               7.492e+08               -1.773e+08                1.529e+08
##       concave.points_se              symmetry_se     fractal_dimension_se
##              -1.260e+09                2.890e+08                1.512e+09
##            radius_worst            texture_worst          perimeter_worst
##              -6.130e+06               -5.832e+05               -3.538e+05
##              area_worst         smoothness_worst        compactness_worst
##               8.950e+04               -2.161e+07                8.986e+06
##         concavity_worst     concave.points_worst           symmetry_worst
##              -3.028e+07                1.431e+08               -2.474e+07
## fractal_dimension_worst
##              -3.698e+07
##
## Degrees of Freedom: 568 Total (i.e. Null);  538 Residual
## Null Deviance:       751.4
## Residual Deviance: 32010     AIC: 32070
```

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

```
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)
}
```

```
beta_init <- rep(0.01, 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,]  NaN
```

Lasso Minimization

We want to achieve the following minimization:

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