

# Newton Raphson Full Model

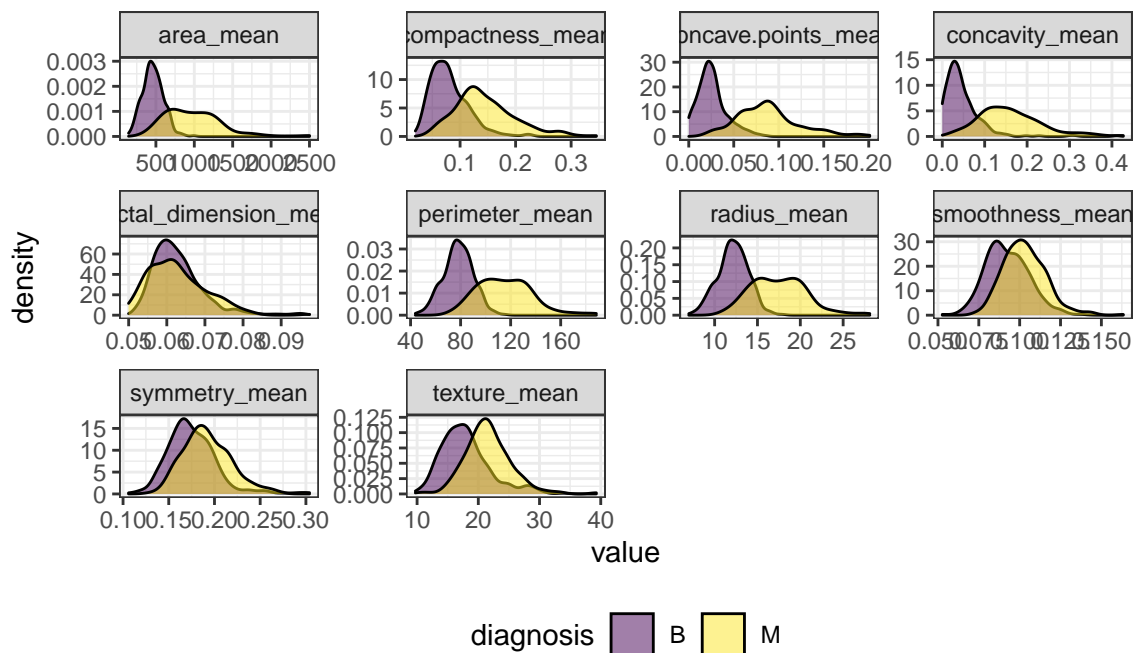
Waveley Qiu (wq2162)

2022-03-22

## 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.danielsjoberg.com/gtsummary/articles/rmarkdown.html
## To suppress this message, include `message = FALSE` in code chunk header.
```

<b>**Characteristic**</b>	<b>**B**, N = 357</b>	<b>**M**, N = 212</b>
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)
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, y, which will be 1 if `diagnosis == 'M'` and 0 if `diagnosis == 'B'`.

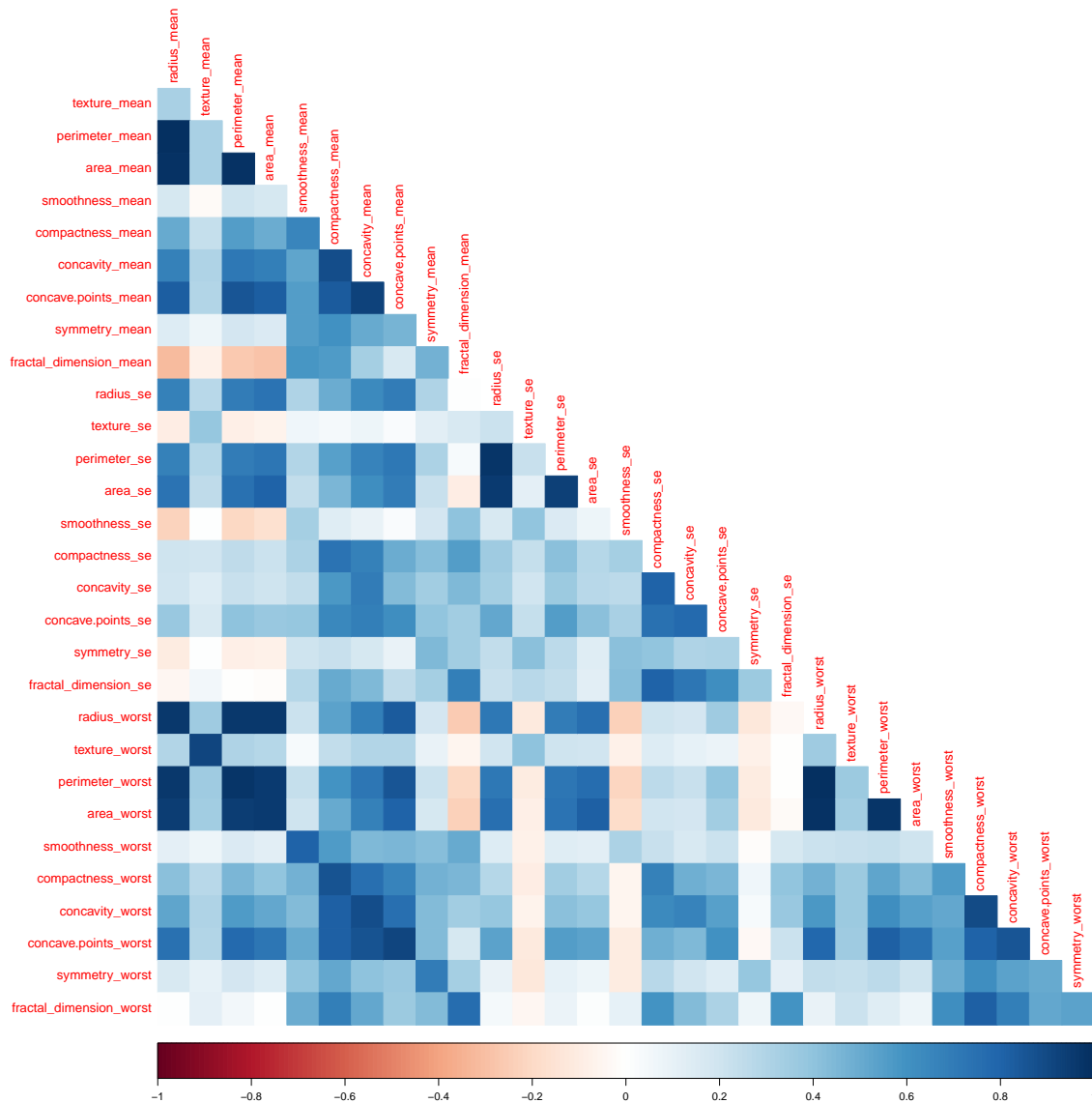
```
bc <- bc %>% mutate(y = ifelse(diagnosis == "M", 1, 0)) %>% relocate(y)
```

## Multicollinearity Investigation

Let's look at the correlations for all predictors.

```
cor_cov_all <- cor(bc %>% select(-c(id, y, diagnosis)))
```

```
corrplot::corrplot(cor_cov_all,
                    method = "color",
                    type = "lower",
                    insig = "blank",
                    diag = FALSE)
```



Yikes. Let's make some cuts ( $r^2 > 0.9$ ) and reevaluate. To begin, we zoom in on our culprits.

```
# call correlation matrix
corr_df <- cor(bc %>% select(-c(id, y, diagnosis)))

# convert matrix to tidy data frame and restrict to highly correlated pairs
corr_df <-
  data.frame(
    v1 = rownames(corr_df)[row(corr_df)[upper.tri(corr_df)]]
    , v2 = colnames(corr_df)[col(corr_df)[upper.tri(corr_df)]]
    , corr = corr_df[upper.tri(corr_df)]
  ) %>%
  arrange(-abs(corr)) %>%
  filter(abs(corr) > 0.90)

# output results
corr_df %>% knitr::kable()
```

v1	v2	corr
radius_mean	perimeter_mean	0.9978553
radius_worst	perimeter_worst	0.9937079
radius_mean	area_mean	0.9873572
perimeter_mean	area_mean	0.9865068
radius_worst	area_worst	0.9840146
perimeter_worst	area_worst	0.9775781
radius_se	perimeter_se	0.9727937
perimeter_mean	perimeter_worst	0.9703869
radius_mean	radius_worst	0.9695390
perimeter_mean	radius_worst	0.9694764
radius_mean	perimeter_worst	0.9651365
area_mean	radius_worst	0.9627461
area_mean	area_worst	0.9592133
area_mean	perimeter_worst	0.9591196
radius_se	area_se	0.9518301
perimeter_mean	area_worst	0.9415498
radius_mean	area_worst	0.9410825
perimeter_se	area_se	0.9376554
concavity_mean	concave.points_mean	0.9213910
texture_mean	texture_worst	0.9120446
concave.points_mean	concave.points_worst	0.9101553

The variables {area\_mean, area\_worst, perimeter\_mean, perimeter\_worst, radius\_mean, radius\_worst} are all mutually correlated. Mathematically, if we consider the equivalence classes of variables that are highly correlated, these six variables would belong to the same equivalence class. Thus, it would be great if we could identify the variable that is the best collective proxy for the other members of this class.

```
# identify largest equivalence class of proxies
corr_df_sub1 <- corr_df %>%
  filter(
    v1 == "area_mean"      | v2 == "area_mean"
  | v1 == "area_worst"     | v2 == "area_worst"
  | v1 == "perimeter_mean" | v2 == "perimeter_mean"
  | v1 == "perimeter_worst" | v2 == "perimeter_worst"
  | v1 == "radius_mean"    | v2 == "radius_mean"
  | v1 == "radius_worst"   | v2 == "radius_worst"
  )

# identify variable that is the best proxy for the other ones
rbind(corr_df_sub1[, -2] %>% rename(v = v1), corr_df_sub1[, -1] %>% rename(v = v2)) %>%
  group_by(v) %>%
  summarize(
    mean_corr = mean(corr)
  , std_corr  = sd(corr)
  , min_corr  = min(corr)
  ) %>%
  arrange(-mean_corr) %>%
  knitr::kable()
```

v	mean_corr	std_corr	min_corr
radius_worst	0.9758968	0.0126266	0.9627461
perimeter_worst	0.9731858	0.0133317	0.9591196
perimeter_mean	0.9731550	0.0212630	0.9415498
radius_mean	0.9721941	0.0218701	0.9410825
area_mean	0.9709886	0.0146306	0.9591196
area_worst	0.9606876	0.0198886	0.9410825

The variable `radius_worst` is the best collective proxy for the other five variables in its collective equivalence class, so we will retain `radius_worst` and discard the rest. Let's look at the remaining variables.

```
corr_df %>%
  filter(!(
    v1 == "area_mean"      | v2 == "area_mean"
  | v1 == "area_worst"    | v2 == "area_worst"
  | v1 == "perimeter_mean" | v2 == "perimeter_mean"
  | v1 == "perimeter_worst" | v2 == "perimeter_worst"
  | v1 == "radius_mean"   | v2 == "radius_mean"
  | v1 == "radius_worst"  | v2 == "radius_worst"
  )) %>%
  knitr::kable()
```

v1	v2	corr
radius_se	perimeter_se	0.9727937
radius_se	area_se	0.9518301
perimeter_se	area_se	0.9376554
concavity_mean	concave.points_mean	0.9213910
texture_mean	texture_worst	0.9120446
concave.points_mean	concave.points_worst	0.9101553

Again, in the spirit of identifying the best representative of each equivalence class, we keep `radius_se`, `concave.points_worst`, and `texture_mean`. In total, we have discarded ten variables.

```
bad_vars <- c(
  "area_mean", "area_worst", "perimeter_mean", "perimeter_worst", "radius_mean"
, "perimeter_se", "area_se"
, "concave.points_worst", "concavity_mean"
, "texture_worst"
)

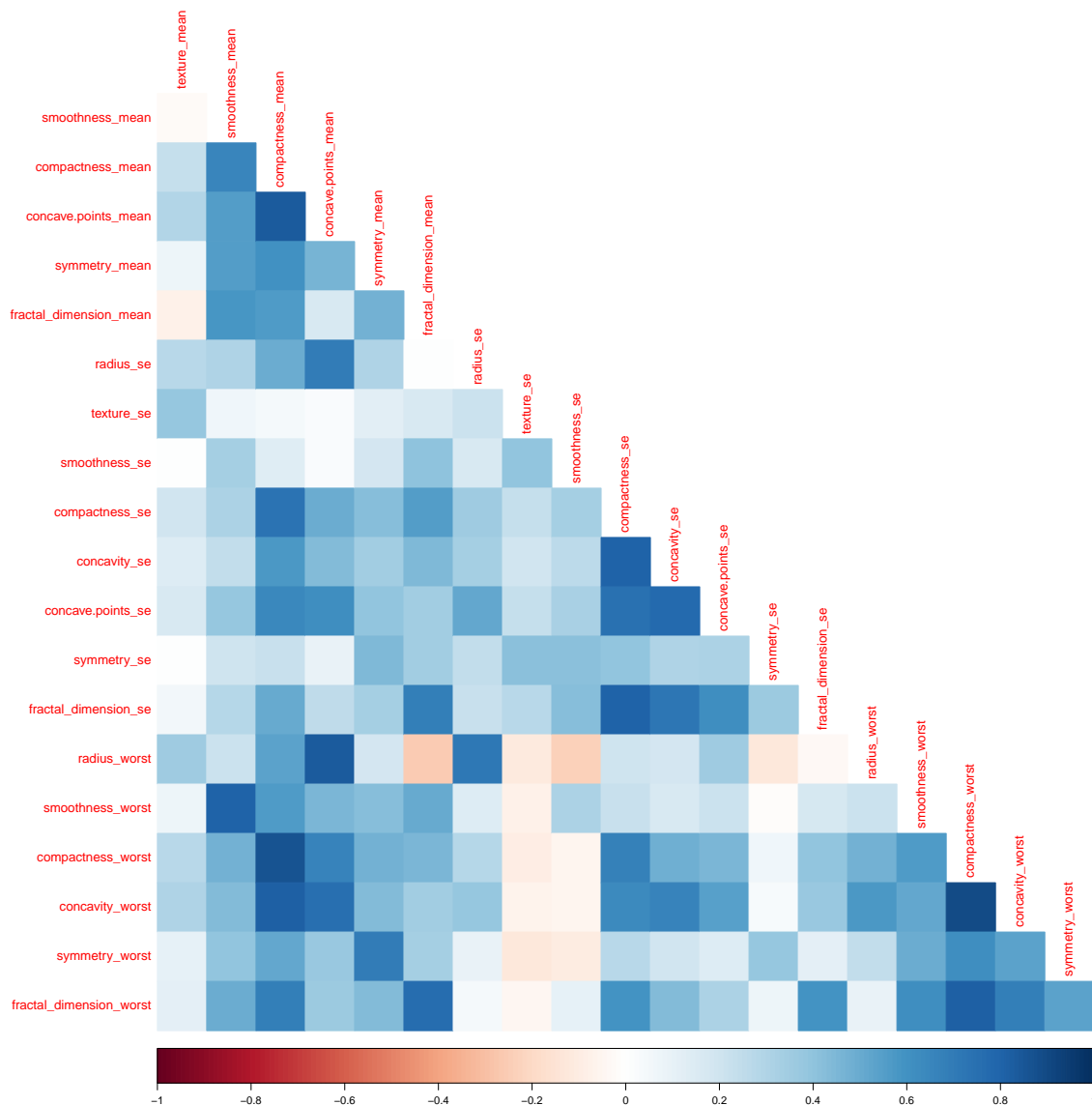
covariates <-
  bc %>%
  select(-c(id, y, diagnosis)) %>%
  dplyr::select(-contains("area")) %>%
  dplyr::select(-contains("perimeter")) %>%
  dplyr::select(-ends_with("_worst")) %>%
  dplyr::select(-c("concavity_mean",
    "compactness_mean",
    "concave.points_mean",
    "compactness_se",
    "concave.points_se"))

covariates <-
  bc %>%
  select(-c(id, y, diagnosis)) %>%
  select(-bad_vars)
```

```
## Note: Using an external vector in selections is ambiguous.
## i Use `all_of(bad_vars)` instead of `bad_vars` to silence this message.
## i See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This message is displayed once per session.
```

```
cor_cov <- cor(covariates)
```

```
corrplot::corrplot(cor_cov,
  method = "color",
  type = "lower",
  insig = "blank",
  diag = FALSE)
```



Better. We'll proceed with these predictors.

```
fin_names <- covariates %>% names()
```

## Full Model

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:

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

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

$$\nabla f(\beta_0, \beta_1, \dots, \beta_p) = \begin{pmatrix} \sum_{i=1}^n Y_i - \pi_i \\ \sum_{i=1}^n x_{i1}(Y_i - \pi_i) \\ \sum_{i=1}^n x_{i2}(Y_i - \pi_i) \\ \vdots \\ \sum_{i=1}^n x_{ip}(Y_i - \pi_i) \end{pmatrix}$$

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

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

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

```
loglike_func <- function(dat, betavec){

  dat = bc_train

  # x matrix
  dat_temp <-
```

```

    dat %>%
    mutate(intercept = 1) %>%
    select(-y) %>%
    relocate(intercept)

dat_x <-
  dat_temp %>%
  as.matrix() %>%
  unname()

# pi vector
u <- dat_x %*% betavec
pi <- exp(u) / (1 + exp(u))

# loglikelihood
loglik <- sum(dat$y*u - log(1 + exp(u)))

#gradient
grad <- t(dat_x) %*% (dat$y - pi)

# Hessian
W <- diag(nrow(pi))
diag(W) <- pi*(1 - pi)
hess <- -(t(dat_x) %*% W %*% (dat_x))

return(list(loglik = loglik, grad = grad, hess = hess))
}

```

Now, let's construct a Newton Raphson algorithm to determine  $\beta_i$  coefficients that maximize the likelihood of the function.

```

NewtonRaphson <- function(dat, start, tol = 1e-8, maxiter = 200){

  i <- 0
  cur <- start
  stuff <- loglike_func(dat, cur)
  res <- c(i = 0, "loglik" = stuff$loglik, "step" = 1, cur)

  prevloglik <- -Inf # to make sure it iterates

  while (i < maxiter && abs(stuff$loglik - prevloglik) > tol) {
    step <- 1
    i <- i + 1
    prevloglik <- stuff$loglik

    # check negative definite
    eigen_vals <- eigen(stuff$hess)

    if (max(eigen_vals$values) <= 0 ) { # check neg def, if not change
      hess <- stuff$hess
    } else { # if it is pos def then need to adjust
      hess <- stuff$hess - (max(eigen_vals$values) + 0.1)*diag(nrow(stuff$hess))
    }
  }
}

```



```

prev <- cur
cur <- prev - step*(solve(stuff$hess) %*% stuff$grad)
stuff <- loglike_func(dat, cur) # log-lik, gradient, Hessian

# step halving
while (stuff$loglik < prevloglik) {
  stuff <- loglike_func(dat, prev)
  step <- step / 2 # this is where half stepping happens
  cur <- prev - step*(solve(stuff$hess) %*% stuff$grad)
  stuff <- loglike_func(dat, cur)
}
# add current values to results matrix
res <- rbind(res, c(i, stuff$loglik, step, cur))
}

colnames(res) <- c("i", "loglik", "step", "intercept", names(dat[, -1]))
return(res)
}

```

## Splitting Testing and Training Data

Using a function from `partition.R`, we will split our data into testing and training. We will include only the variables we have identified as being sufficiently uncorrelated.

```

col.num <- which(colnames(bc) %in% fin_names) # get those variables
bc_trunc <- bc[, c(1, col.num)]

bc_test_train <- partition(0.8, bc_trunc)

bc_test <- bc_test_train %>% filter(part_id == "test") %>% select(-part_id)
bc_train <- bc_test_train %>% filter(part_id == "train") %>% select(-part_id)

```

## Running Newton-Raphson Algorithm

Let's establish a beta vector and run the Newton-Raphson algorithm we've established on the training data.

```

betavec <- c(rep(0.01, ncol(bc_train))) %>% as.matrix()
nr1 <- NewtonRaphson(bc_train, betavec)
beta_est <- nr1[nrow(nr1), -c(1:3)] %>% as.vector()

```

## Compare with GLM and GLMNET

Let's run GLM.

```

glm_fit <- glm(y~., data = bc_train, family = "binomial")

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
result_glm <- summary(glm_fit)

glm_est <- glm_fit %>%
  broom::tidy() %>%

```

```

select(term, estimate) %>%
mutate(glm_est = round(estimate, 3)) %>%
select(-estimate) %>%
mutate(term = ifelse(term == "(Intercept)", "intercept", term)) %>% pull(glm_est)

```

Now, let's run GLMNET.

```

xdat <- bc_train %>% select(-y) %>% as.matrix()
glmnet_fit <- glmnet(x = xdat, y = bc_train$y, family = "binomial", lambda = 0, thresh = 10^-12)
glmnet_est <- coef(glmnet_fit) %>% as.vector() %>% round(3)

```

```

tibble(
  beta_sub = 0:(length(beta_est) - 1),
  nr_est = round(beta_est, 3),
  glm_est = glm_est,
  glmnet_est = glmnet_est
) %>%
  knitr::kable()

```

beta_sub	nr_est	glm_est	glmnet_est
0	-162.116	-162.116	-162.067
1	1.923	1.923	1.922
2	-180.934	-180.934	-180.933
3	-206.952	-206.952	-206.897
4	539.502	539.502	539.382
5	-71.919	-71.919	-71.906
6	555.503	555.503	555.377
7	66.823	66.823	66.806
8	-3.717	-3.717	-3.716
9	424.734	424.734	424.570
10	264.450	264.450	264.304
11	-215.047	-215.047	-214.985
12	1443.885	1443.885	1443.446
13	-1672.713	-1672.713	-1672.277
14	-4616.356	-4616.356	-4614.349
15	1.917	1.917	1.916
16	29.366	29.366	29.371
17	-91.468	-91.468	-91.435
18	71.713	71.713	71.695
19	198.978	198.978	198.929
20	429.955	429.955	429.742

## AUC

```

auc_calc_full <- function(beta_est, test_data){
  # pulling out the terms used in the full model (should be all)

  # we have this flexible in case we want to test fewer variables
  # terms <- beta_est %>% pull(term)
  # col.num <- which(colnames(test_data) %in% terms)
  # select the desired x values

  xvals <- test_data[,-1] %>%

```

```

    mutate(
      intercept = 1 # create a intercept variable
    ) %>%
    relocate(intercept) # move it to the front

    pred <- as.matrix(xvals) %*% beta_est # get the cross product of the linear model
    logit_pred <- exp(pred) / (1 + exp(pred)) # link function to get probabilities

    auc_val <- auc(test_data$y, as.vector(logit_pred)) # calculating the AUC

    # roc(tst_data$y, as.vector(logit_pred)) %>% plot( legacy.axes=TRUE) # graphs AUC
    return(auc_val)
  }

auc <- auc_calc_full(beta_est, bc_test)

```

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

The AUC of the Newton-Raphson Model is 0.9764706.