Task 2

Task 2: Develop a Newton-Raphson algorithm to estimate your model.

The target function f given in task 1:

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

We develop a modified Newton-Raphson algorithm including a step-halving step. *(we probably don't need to ensure that the direction of the step is an ascent direction, since in this example Hessian is always negative-definite. but Hessian could be computationally singular when the starting points are bad)*

Algorithm 1 Newton-Raphson algorithm including a step-halving step

---

Require: $f(\boldsymbol{\beta})$ - target function as given in (1); $\boldsymbol{\beta}_0$ - starting value
Ensure: $\widehat{\boldsymbol{\beta}}_{approx}$ such that $\widehat{\boldsymbol{\beta}}_{approx} \approx \arg\max_{\boldsymbol{\beta}} f(\boldsymbol{\beta})$
$i \leftarrow 0$, where i is the current number of iterations
$f(\boldsymbol{\beta}_{-1}) \leftarrow -\infty$
while convergence criterion is not met **do**
  $i \leftarrow i + 1$
  $\mathbf{d}_i \leftarrow -[\nabla^2 f(\boldsymbol{\beta}_{i-1})]^{-1} \nabla f(\boldsymbol{\beta}_{i-1})$, where $\mathbf{d}_i$ is the direction in the i-th iteration
  $\lambda_i \leftarrow 1$, where λ_i is the multiplier in the i-th iteration
  $\boldsymbol{\beta}_i \leftarrow \boldsymbol{\beta}_{i-1} + \lambda_i \mathbf{d}_i$
  while $f(\boldsymbol{\beta}_i) \leq f(\boldsymbol{\beta}_{i-1})$ **do**
    $\lambda_i \leftarrow \lambda_i / 2$
    $\boldsymbol{\beta}_i \leftarrow \boldsymbol{\beta}_{i-1} + \lambda_i \mathbf{d}_i$
  end while
end while
$\widehat{\boldsymbol{\beta}}_{approx} \leftarrow \boldsymbol{\beta}_i$

We write an **R**-function `NewtonRaphson` to implement the algorithm.

```
NewtonRaphson <- function(dat, func, start, tol = 1e-10) {
  i <- 0
  cur <- start
  stuff <- func(dat, cur)
  res <- c(0, stuff$f, cur)
  prevf <- -Inf
  while (abs(stuff$f - prevf) > tol) {
    i <- i + 1
    prevf <- stuff$f
    prev <- cur
    d <- -solve(stuff$Hess) %*% stuff$grad
    cur <- prev + d
    lambda <- 1
    maxhalv <- 0
    while (func(dat, cur)$f < prevf && maxhalv < 50) {
```

```r
      maxhalv <- maxhalv + 1
      lambda <- lambda / 2
      cur <- prev + lambda * d
    }
    stuff <- func(dat, cur)
    res <- rbind(res, c(i, stuff$f, cur))
  }
  colnames(res) <- c("iter", "target_function", "(Intercept)", names(dat)[-1])
  return(res)
}
```

Data preprocessing and data partition.

```r
bc_df <- read.csv("breast-cancer.csv")[-c(1, 33)] %>% # remove variable ID and an NA column
  mutate(diagnosis = ifelse(diagnosis == "M", 1, 0)) # code malignant cases as 1
bc_df[, -1] <- scale(bc_df[, -1]) # predictors are standardized for the logistic-LASSO model in task 3

set.seed(1)
indexTrain <- createDataPartition(y = bc_df$diagnosis, p = 0.8, list = FALSE)
Training <- bc_df[indexTrain, ]
Test <- bc_df[-indexTrain, ]

# correlation coefficients close to 1. need to remove some variables to make the algorithm converge
glm(diagnosis ~ ., family = binomial(link = "logit"), data = Training)
```

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

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

##
## Call:  glm(formula = diagnosis ~ ., family = binomial(link = "logit"),
##     data = Training)
##
## Coefficients:
##             (Intercept)              radius_mean             texture_mean
##                 90.9690               -2560.3939                   0.8812
##          perimeter_mean                area_mean          smoothness_mean
##                789.2724                1539.8346                 128.8762
##        compactness_mean           concavity_mean      concave.points_mean
##               -346.6692                   9.0810                 215.4808
##           symmetry_mean   fractal_dimension_mean                radius_se
##                 10.4773                 -28.6917                 617.5687
##              texture_se             perimeter_se                  area_se
##                -79.9789                -917.3917                 628.6222
##           smoothness_se           compactness_se             concavity_se
##                -63.7660                 344.3180                -323.8534
##       concave.points_se              symmetry_se     fractal_dimension_se
##                374.7611                -108.6212                -339.1646
##            radius_worst            texture_worst          perimeter_worst
##                197.9315                 155.0787                1068.1069
##              area_worst         smoothness_worst        compactness_worst
##               -606.1658                 -26.4759                -346.4052
```

```
##         concavity_worst     concave.points_worst           symmetry_worst
##                373.2089                -161.6177                  71.9489
## fractal_dimension_worst
##                282.8254
##
## Degrees of Freedom: 455 Total (i.e. Null);  425 Residual
## Null Deviance:       601.3
## Residual Deviance: 1.311e-06     AIC: 62
```

Remove some variables. Here we select all mean predictors as an example (still have highly-correlated variables). **(Should remove variables that are highly correlated. NOT DECIDED!)**

```r
# select some variables. should decide which variables to choose
useful <- names(bc_df)[1:11] # e.g., select all mean variables
bc_df2 <-
  bc_df %>%
  select(all_of(useful))

set.seed(1)
indexTrain <- createDataPartition(y = bc_df2$diagnosis, p = 0.8, list = FALSE)
Training <- bc_df2[indexTrain, ]
Test <- bc_df2[-indexTrain, ]

glm(diagnosis ~ ., family = binomial(link = "logit"), data = Training)
```

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

```
##
## Call:  glm(formula = diagnosis ~ ., family = binomial(link = "logit"),
##     data = Training)
##
## Coefficients:
##            (Intercept)             radius_mean            texture_mean
##                 0.4117                -18.3611                  1.6021
##         perimeter_mean               area_mean         smoothness_mean
##                 8.6626                 15.1088                  1.0744
##       compactness_mean          concavity_mean     concave.points_mean
##                -0.0888                  0.3343                  2.2067
##          symmetry_mean  fractal_dimension_mean
##                 0.6342                 -0.5820
##
## Degrees of Freedom: 455 Total (i.e. Null);  445 Residual
## Null Deviance:       601.3
## Residual Deviance: 116.7     AIC: 138.7
```

```r
logisticstuff <- function(dat, betavec) {
  dat <- as.matrix(dat)
  n <- nrow(dat)
  p <- ncol(dat) - 1
  X <- cbind(rep(1, n), dat[, -1]) # design matrix
  y <- dat[, 1] # response vector
  u <- X %*% betavec # x_i^T beta, i=1,...,n
  f <- sum(y * u - log1pexp(u)) # function `log1pexp` to compute log(1 + exp(x)))
```

```
  p_vec <- sigmoid(u) # function `sigmoid` to compute exp(x)/(1 + exp(x))
  grad <- t(X) %*% (y - p_vec)
  Hess <- -t(X) %*% diag(c(p_vec * (1 - p_vec))) %*% X
  return(list(f = f, grad = grad, Hess = Hess))
}
```

We fit a logistic regression model on the training data using our `NewtonRaphson` function.

```
res <- NewtonRaphson(dat = Training, func = logisticstuff, start = rep(0, ncol(Training)))
res
```

```
##     iter target_function (Intercept)  radius_mean texture_mean perimeter_mean
## res    0      -316.07511   0.0000000   0.00000000    0.0000000      0.0000000
##        1      -129.88900  -0.5213358   7.46247726    0.3593164     -6.2762894
##        2       -86.99235  -0.7300300  10.08483576    0.6391330     -8.2532785
##        3       -69.28577  -0.8159302   7.59038364    0.9534132     -5.9835656
##        4       -62.07699  -0.6613894  -0.09853138    1.2435119     -0.2505531
##        5       -59.17237  -0.1869629  -9.84136501    1.4404154      5.1385186
##        6       -58.39991   0.2759214 -16.39521429    1.5559376      7.8043536
##        7       -58.35396   0.4048031 -18.24786635    1.5989307      8.6072512
##        8       -58.35380   0.4116856 -18.36070646    1.6021338      8.6623617
##        9       -58.35380   0.4117085 -18.36110260    1.6021463      8.6625595
##       10       -58.35380   0.4117085 -18.36110260    1.6021463      8.6625595
##      area_mean smoothness_mean compactness_mean concavity_mean
## res  0.0000000       0.0000000       0.00000000      0.0000000
##     -1.0612656       0.1067701       0.19577818      0.1839443
##     -1.4030153       0.2615841       0.28540534      0.3765710
##     -0.4711325       0.4832265       0.21409284      0.5105891
##      2.6852622       0.7505353       0.04538621      0.5466722
##      8.5735310       0.9668330      -0.05607219      0.4640845
##     13.6393255       1.0512265      -0.07560626      0.3615667
##     15.0304471       1.0726835      -0.08764250      0.3353827
##     15.1085159       1.0743573      -0.08879371      0.3342717
##     15.1087870       1.0743641      -0.08879778      0.3342689
##     15.1087870       1.0743641      -0.08879778      0.3342689
##     concave.points_mean symmetry_mean fractal_dimension_mean
## res            0.000000     0.0000000             0.00000000
##                1.015298     0.1221843            -0.04918957
##                1.522512     0.2502993            -0.19391377
##                1.842827     0.3972425            -0.33787796
##                1.952803     0.5017934            -0.43708438
##                1.987360     0.5565752            -0.50011707
##                2.128872     0.6070663            -0.55603567
##                2.201789     0.6322255            -0.58020461
##                2.206636     0.6341719            -0.58195424
##                2.206653     0.6341794            -0.58196081
##                2.206653     0.6341794            -0.58196081
```

We compare the results of using the `glm` function and our `NewtonRaphson` function.

```
tibble(
  predictor = c("(Intercept)", names(Training)[-1]),
```

```
  ours = res[nrow(res), -c(1, 2)],
  glm = glm(diagnosis ~ ., family = binomial(link = "logit"), data = Training)$coefficients
) %>%
  knitr::kable()
```

predictor	ours	glm
(Intercept)	0.4117085	0.4117085
radius_mean	-18.3611026	-18.3611026
texture_mean	1.6021463	1.6021463
perimeter_mean	8.6625595	8.6625595
area_mean	15.1087870	15.1087870
smoothness_mean	1.0743641	1.0743641
compactness_mean	-0.0887978	-0.0887978
concavity_mean	0.3342689	0.3342689
concave.points_mean	2.2066530	2.2066530
symmetry_mean	0.6341794	0.6341794
fractal_dimension_mean	-0.5819608	-0.5819608

Compute the test AUC. (should NOT be used for model comparison)

```
betavec.logit <- res[nrow(res), 3:ncol(res)]
# test data
X_test <- cbind(rep(1, nrow(Test)), model.matrix(diagnosis ~ ., Test)[, -1])
y_test <- Test$diagnosis
# AUC
u_test <- X_test %*% betavec.logit
phat_test <- sigmoid(u_test)[, 1]
roc.logit.test <- roc(response = y_test, predictor = phat_test)
auc.logit.test <- roc.logit.test$auc[1]
```

Resampling on training data: *Does the following resampling method work?*

```
?caret::resamples
```

Hothorn et al. The design and analysis of benchmark experiments. Journal of Computational and Graphical Statistics (2005) vol. 14 (3) pp. 675-699

https://ro.uow.edu.au/cgi/viewcontent.cgi?article=3494&context=commpapers

RW-OOB

```
B = 100 # number of bootstrap samples
set.seed(1)
auc.logit <- rep(NA, B)
for (i in 1:B) {
  index_bs <- sample(nrow(Training), replace = TRUE)
  sample <- Training[index_bs, ]
  out <- Training[-index_bs, ]
  res <- NewtonRaphson(dat = sample, func = logisticstuff, start = rep(0, ncol(sample)))
  betavec <- res[nrow(res), 3:ncol(res)]
  X <- cbind(rep(1, nrow(out)), model.matrix(diagnosis ~ ., out)[, -1])
```

```r
  y <- out$diagnosis
  u <- X %*% betavec
  phat <- sigmoid(u)[, 1]
  roc <- roc(response = y, predictor = phat)
  auc <- roc$auc[1]
  auc.logit[i] <- auc
}
summary(auc.logit)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.9623  0.9764  0.9811  0.9812  0.9863  0.9976
```

```r
boxplot(auc.logit)
```