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: $\widehat{\boldsymbol{\beta}}$ such that $\widehat{\boldsymbol{\beta}} \approx \arg\max_{\boldsymbol{\beta}} f(\boldsymbol{\beta})$
 1: $i \leftarrow 0$, where $i$ is the current number of iterations
 2: $f(\boldsymbol{\beta}_{-1}) \leftarrow -\infty$
 3: while convergence criterion is not met do
 4:     $i \leftarrow i + 1$
 5:     $\mathbf{d}_i \leftarrow -[\nabla^2 f(\boldsymbol{\beta}_{i-1})]^{-1} \nabla f(\boldsymbol{\beta}_{i-1})$, where $\mathbf{d}_i$ is the direction in the $i$-th iteration
 6:     $\lambda_i \leftarrow 1$, where $\lambda_i$ is the multiplier in the $i$-th iteration
 7:     $\boldsymbol{\beta}_i \leftarrow \boldsymbol{\beta}_{i-1} + \lambda_i \mathbf{d}_i$
 8:     while $f(\boldsymbol{\beta}_i) \le f(\boldsymbol{\beta}_{i-1})$ do
 9:         $\lambda_i \leftarrow \lambda_i / 2$
10:         $\boldsymbol{\beta}_i \leftarrow \boldsymbol{\beta}_{i-1} + \lambda_i \mathbf{d}_i$
11:     end while
12: end while
13: $\widehat{\boldsymbol{\beta}} \leftarrow \boldsymbol{\beta}_i$
```

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

```r
NewtonRaphson <- function(dat, func, start, tol = 1e-8, maxiter = 200) {
  i <- 0
  cur <- start
  stuff <- func(dat, cur)
  res <- c(0, stuff$f, cur)
  prevf <- -Inf
  X <- cbind(rep(1, nrow(dat)), as.matrix(dat[, -1]))
  y <- dat[, 1]
  warned <- 0
  while (abs(stuff$f - prevf) > tol && i < maxiter) {
    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) {
      maxhalv <- maxhalv + 1
      lambda <- lambda / 2
      cur <- prev + lambda * d
    }
    stuff <- func(dat, cur)
    res <- rbind(res, c(i, stuff$f, cur))
    y_hat <- ifelse(X %*% cur > 0, 1, 0)
    if (warned == 0 && sum(y - y_hat) == 0) {
      warning("Complete separation occurs. Algorithm does not converge.")
      warned <- 1
    }
  }
  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, ]

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
```

```
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)))
```

```
## Warning in NewtonRaphson(dat = Training, func = logisticstuff, start = rep(0, :
## Complete separation occurs. Algorithm does not converge.
```

```
tail(res)
```

```
##   iter target_function (Intercept) radius_mean texture_mean perimeter_mean
##     26   -3.235158e-07    89.24826   -2822.544    -1.373617       960.4715
##     27   -1.190284e-07    94.06615   -2977.405    -1.518686      1010.5839
##     28   -4.379281e-08    98.80114   -3134.863    -1.704236      1063.1357
##     29   -1.611204e-08   103.40055   -3296.525    -1.956417      1119.6337
##     30   -5.927773e-09   107.76671   -3465.377    -2.323911      1182.8800
##     31   -2.180833e-09   111.72302   -3646.824    -2.894920      1257.9481
##   area_mean smoothness_mean compactness_mean concavity_mean concave.points_mean
##    1628.129        140.0014        -368.0153      2.2563839            224.3770
##    1719.801        147.7362        -388.2773      2.1569756            236.9795
##    1811.729        155.5634        -408.6480      1.9513295            249.5825
##    1904.079        163.5407        -429.1918      1.5722084            262.1834
##    1997.154        171.7748        -450.0285      0.8948694            274.7774
##   2091.500        180.4591        -471.3749     -0.3063034            287.3555
##  symmetry_mean fractal_dimension_mean radius_se texture_se perimeter_se
##       8.431909              -31.04890  664.0699  -82.56232    -964.1721
##       9.125769              -32.85952  700.8645  -87.29044   -1017.2613
##       9.763178              -34.69764  737.9928  -92.00209   -1070.4606
##      10.311347              -36.58090  775.6589  -96.68750   -1123.8325
##      10.709177              -38.54224  814.2424 -101.32850   -1177.4929
##      10.845978              -40.64135  854.4291 -105.89208   -1231.6515
##   area_se smoothness_se compactness_se concavity_se concave.points_se
##  627.0977     -65.29781       368.1306    -343.4408          393.0646
##  661.3045     -69.00116       388.5425    -362.5350          414.9751
##  695.0465     -72.68019       409.1130    -381.7210          436.9259
##  728.0267     -76.31976       429.9373    -401.0546          458.9405
##  759.6946     -79.89169       451.1922    -420.6395          481.0630
##  789.0550     -83.34497       473.1970    -440.6629          503.3729
##  symmetry_se fractal_dimension_se radius_worst texture_worst perimeter_worst
##    -113.2103            -360.5082     247.8024      164.0343        1108.082
##    -119.4203            -380.3806     263.6037      173.3317        1168.164
##    -125.6423            -400.3884     279.9778      182.6679        1228.146
##    -131.8815            -420.6120     297.2940      192.0677        1287.947
##    -138.1475            -441.2006     316.2365      201.5777        1347.420
##    -144.4581            -462.4243     338.0409      211.2818        1406.298
##  area_worst smoothness_worst compactness_worst concavity_worst
##   -659.5481        -32.14856         -373.8623        401.9769
##   -696.8089        -33.84766         -394.4311        424.5118
##   -734.3769        -35.62248         -415.2146        447.2448
##   -772.4418        -37.52094         -436.3424        470.2984
##   -811.3548        -39.63214         -458.0558        493.8996
##   -851.7493        -42.11738         -480.7909        518.4595
##  concave.points_worst symmetry_worst fractal_dimension_worst
##             -169.3830       77.46596                300.1686
##             -178.8370       81.34102                316.7304
##             -188.3071       85.28220                333.3894
##             -197.8017       89.32603                350.2048
##             -207.3369       93.54077                367.2864
##             -216.9415       98.05014                384.8331
```

Our function also does not converge, because a complete separation occurs. A complete separation in a logistic regression, sometimes also referred as perfect prediction, which occurs whenever there exists some vector of coefficients $\boldsymbol{\beta}$ such that $Y_i = 1$ whenever $\mathrm{x}_i^\top \boldsymbol{\beta} > 0$ and $Y_i = 0$ whenever $\mathrm{x}_i^\top \boldsymbol{\beta} \leq 0$. In other words, complete separation occurs whenever a linear function of predictors can generate perfect predictions of response.

```
X <- cbind(rep(1, nrow(Training)), model.matrix(diagnosis ~ ., Training)[, -1])
y <- Training$diagnosis
coef_newton <- res[nrow(res), -c(1, 2)]
y_hat <- ifelse(X %*% coef_newton > 0, 1, 0) # predictions
sum(y - y_hat) # prefect separation
```

```
## [1] 0
```

We compare the results of using the `glm` function and our `NewtonRaphson` function. (meaningless, since both do not converge)

```
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)	111.7230205	90.9690365
radius_mean	-3646.8235396	-2560.3938902
texture_mean	-2.8949199	0.8812037
perimeter_mean	1257.9481182	789.2724398
area_mean	2091.5001211	1539.8345650
smoothness_mean	180.4591376	128.8762247
compactness_mean	-471.3749334	-346.6691873
concavity_mean	-0.3063034	9.0810082
concave.points_mean	287.3555092	215.4808031
symmetry_mean	10.8459784	10.4772590
fractal_dimension_mean	-40.6413497	-28.6916549
radius_se	854.4290934	617.5687358
texture_se	-105.8920782	-79.9788638
perimeter_se	-1231.6514585	-917.3916527
area_se	789.0550144	628.6222313
smoothness_se	-83.3449730	-63.7660367
compactness_se	473.1970464	344.3180183
concavity_se	-440.6629326	-323.8533730
concave.points_se	503.3728692	374.7610875
symmetry_se	-144.4580558	-108.6211792
fractal_dimension_se	-462.4242540	-339.1646360
radius_worst	338.0408767	197.9314738
texture_worst	211.2817780	155.0787339
perimeter_worst	1406.2984247	1068.1068808
area_worst	-851.7493135	-606.1657870
smoothness_worst	-42.1173805	-26.4759499
compactness_worst	-480.7909132	-346.4051824
concavity_worst	518.4595221	373.2089180
concave.points_worst	-216.9414972	-161.6177074
symmetry_worst	98.0501371	71.9488540
fractal_dimension_worst	384.8330525	282.8253512

We probably won't conduct resampling. Ignore the below.

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

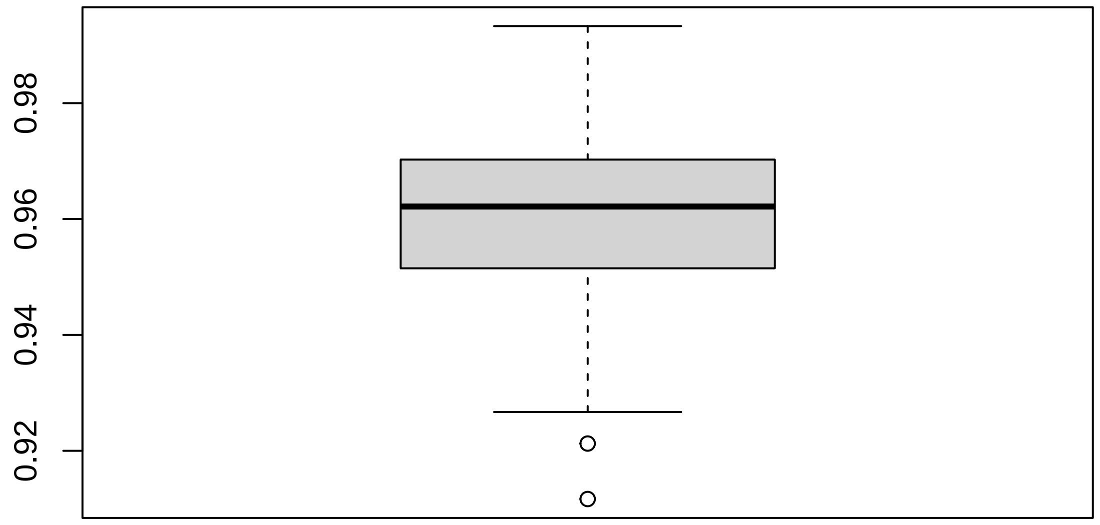
https://ro.uow.edu.au/cgi/viewcontent.cgi?article=3494&context=commpapers

RW-OOB

```r
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])
  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.
##  0.9117  0.9518  0.9622  0.9611  0.9703  0.9933
```

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


0.92
0.94
0.96
0.98