Classification of Cancer Patients with Penalized Robust Nonconvex Loss Functions

Zhu Wang
UT Health San Antonio
wangz1@uthscsa.edu

January 14, 2019

This document presents analysis for the MAQC-II project, human breast cancer data set with penalized classification algorithms developed in Wang (2016) and implemented in R package `mpath`.

Dataset comes from the MicroArray Quality Control (MAQC) II project and includes 278 breast cancer samples with 164 estrogen receptor (ER) positive cases. The data files `GSE20194_series_matrix.txt.gz` and `GSE20194_MDACC_Sample_Info.xls` can be downloaded from http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?token=rhojvaiwkcsaihq&acc=GSE20194. After reading the data, some unused variables are removed. From 22283 genes, the dataset is pre-screened to obtain 3000 genes with the largest absolute values of the two-sample t-statistics. The 3000 genes are standardized.

```
# The data files below were downloaded on June 1, 2016
require("gdata")
library("mpath")
bc <- t(read.delim("GSE20194_series_matrix.txt.gz", sep = "",
    header = FALSE, skip = 80))
colnames(bc) <- bc[1, ]
bc <- bc[-1, -c(1, 2)]
### The last column is empty with variable name
### !series_matrix_table_end, thus omitted
bc <- bc[, -22284]
mode(bc) <- "numeric"  ### convert character to numeric
dat1 <- read.xls("GSE20194_MDACC_Sample_Info.xls", sheet = 1,
    header = TRUE)
y <- dat1$characteristics..ER_status
y <- ifelse(y == "P", 1, -1)
table(y)
res <- rep(NA, dim(bc)[2])
for (i in 1:dim(bc)[2]) res[i] <- abs(t.test(bc[, i] ~ y)$statistic)
### find 300 largest absolute value of t-statistic
tmp <- order(res, decreasing = TRUE)[1:3000]
dat <- bc[, tmp]
### standardize variables
```

```
dat <- scale(dat)
```

Set up configuration parameters.

```
### number of replicates
nrun <- 100
### penalty type
penalty <- c("enet", "snet", "mnet")
### Smallest value for lambda, as a fraction of lambda.max, the
### smallest value for which all coefficients are zero except
### the intercept
ratio <- 0.25
type.path <- "active"
nlam <- ifelse(type.path != "onestep", 30, 100)
### The training data is contaminated by randomly switching
### response variable labels at varying pre-specified
### proportions
per <- c(0, 0.05, 0.1, 0.15)
### what quantity is minimized for tuning parameter selection
tuning <- "error"
### In cross-validation, we set the number of cores for
### parallel computing by
n.cores <- 4
### robust nonconvex loss function, rfamily type and logistic
type <- c("closs", "gloss", "qloss", "binomial")
### and corresponding labels
type1 <- c("Closs", "Gloss", "Qloss", "Logistic")
### and corresponding tuning parameter
s <- c(0.9, 1.01, 0.5)
mstop <- 50
plot.it <- TRUE
```

The training data contains randomly selected 50 samples with positive estrogen receptor status and 50 samples with negative estrogen receptor status, and the rest were designated as the test data. The training data is contaminated by randomly switching response variable labels at varying pre-specified proportions `per`=0, 0.05, 0.1, 0.15. This process is repeated `nrun`=100 times. Robust non-convex loss functions include C-loss, G-loss and Q-loss, each with penalty LASSO, SCAD and MCP. The initial values are derived using the boosting package `bst` with `mstop`=50 and `nu` provided below depending on loss function `type`. For SCAD and MCP penalty, a penalty tuning parameter `gam` is provided below. By default, the solution path is computed backward as `direction="bwd"` in `nclreg` function. To select optimal penalization tuning parameters, we run five-fold cross-validation averaging classification errors. In cross-validation, we set the number of cores for parallel computing by `n.cores`=4. The classification errors and number of selected variables are summarized and these results can be plotted if `plot.it=TRUE`. This script also contains results with penalized logistic regression using `glmreg`. The script takes 7 hours to execute on Intel® Core™ i7-4790 CPU @ 3.60GHz × 4 processor.

```
summary7 <- function(x) c(summary(x), sd = sd(x))
ptm <- proc.time()
for (k in (1:4)) {
    ### k controls family argument rfamily type (see above)
    if (type[k] == "gloss")
        nu <- 0.1 else nu <- 0.01
    for (j in (1:3)) {
        ### j controls argument penalty type (see above)
        gam <- ifelse(penalty[j] == "snet", 3.7, 12)
        err.m1 <- nvar.m1 <- errbest.m1 <- lambest.m1 <- matrix(NA,
            ncol = 4, nrow = nrun)
        nvarbest.m1 <- mstopcv.m1 <- matrix(NA, ncol = 4, nrow = nrun)
        colnames(err.m1) <- c("cont-0%", "cont-5%", "cont-10%",
            "cont-15%")
        colnames(mstopcv.m1) <- colnames(nvarbest.m1) <- colnames(err.m1)
        colnames(nvar.m1) <- colnames(err.m1)
        colnames(errbest.m1) <- colnames(err.m1)
        colnames(lambest.m1) <- colnames(err.m1)
        for (ii in 1:nrun) {
            set.seed(1000 + ii)
            trid <- c(sample(which(y == 1))[1:50], sample(which(y ==
                -1))[1:50])
            dtr <- dat[trid, ]
            dte <- dat[-trid, ]
            ytrold <- y[trid]
            yte <- y[-trid]
            ### number of patients/no. variables in training and test data
            dim(dtr)
            dim(dte)
            ### randomly contaminate data
            ntr <- length(trid)
            con <- sample(ntr)
            for (i in (1:4)) {
                ### i controls how many percentage of data contaminated, see
                ### argument per above
                ytr <- ytrold
                percon <- per[i]
                ### randomly flip labels of the samples in training set
                ### according to pre-defined contamination level
                if (percon > 0) {
                  ji <- con[1:(percon * ntr)]
                  ytr[ji] <- -ytrold[ji]
                }
                ### fit a model with nclreg for nonconvex loss or glmreg for
                ### logistic loss, and use cross-validation to select best
                ### penalization parameter
                if (type[k] %in% c("closs", "gloss", "qloss")) {
                  dat.m1 <- nclreg(x = dtr, y = ytr, s = s[k],
                    iter = 100, rfamily = type[k], penalty = penalty[j],
```

```
      lambda.min.ratio = ratio, gamma = gam, mstop.init = mstop,
      nu.init = nu, type.path = type.path)
    cvm1 <- cv.nclreg(x = dtr, y = ytr, nfolds = 5,
      n.cores = n.cores, s = s[k], lambda = dat.m1$lambda[1:nlam],
      rfamily = type[k], penalty = penalty[j],
      gamma = gam, type = tuning, plot.it = FALSE,
      type.init = dat.m1$type.init, mstop.init = dat.m1$mstop.init,
      nu.init = dat.m1$nu.init, type.path = type.path)
    err1 <- predict(dat.m1, newdata = dte, newy = yte,
      type = "error")
  } else {
    dat.m1 <- glmreg(x = dtr, y = (ytr + 1)/2,
      family = type[k], penalty = penalty[j], lambda.min.ratio = ratio,
      gamma = gam)
    cvm1 <- cv.glmreg(x = dtr, y = (ytr + 1)/2,
      nfolds = 5, n.cores = n.cores, lambda = dat.m1$lambda,
      family = type[k], penalty = penalty[j], gamma = gam,
      plot.it = FALSE)
    err1 <- apply((yte > -1) != predict(dat.m1,
      newx = dte, type = "class"), 2, mean)
  }
  optmstop <- cvm1$lambda.which
  err.m1[ii, i] <- err1[optmstop]
  nvar.m1[ii, i] <- length(predict(dat.m1, which = optmstop,
    type = "nonzero"))
  errbest.m1[ii, i] <- min(err1, na.rm = TRUE)
  lambest.m1[ii, i] <- which.min(err1)
  nvarbest.m1[ii, i] <- length(predict(dat.m1,
    which = which.min(err1), type = "nonzero"))
}
if (ii%%nrun == 0) {
  if (type[k] %in% c("closs", "gloss", "qloss"))
    cat(paste("\nfamily ", type1[k], ", s=", s[k],
      sep = ""), "\n") else cat(paste("\nfamily ", type1[k], sep = ""),
    "\n")
  pentype <- switch(penalty[j], enet = "LASSO",
    mnet = "MCP", snet = "SCAD")
  cat("penalty=", pentype, "\n")
  if (penalty[j] %in% c("snet", "mnet"))
    cat("gamma=", gam, "\n")
  cat("best misclassification error\n")
  print(round(apply(errbest.m1, 2, summary7), 4))
  cat("which lambda has best error\n")
  print(round(apply(lambest.m1, 2, summary7), 1))
  cat("number of variables selected with best error\n")
  print(round(apply(nvarbest.m1, 2, summary7),
    1))
  cat("CV based misclassification error\n")
  print(round(apply(err.m1, 2, summary7), 4))
```

```
                cat("number of variables selected by CV\n")
                print(round(apply(nvar.m1, 2, summary7), 1))
                if (plot.it) {
                  par(mfrow = c(2, 1))
                  boxplot(err.m1, main = "Misclassification error",
                    subset = "", sub = paste(type1[k], "-", pentype,
                      sep = ""))
                  boxplot(nvar.m1, main = "No. variables", subset = "",
                    sub = paste(type1[k], "-", pentype, sep = ""))
                }
              }
          }
      }
}
print(proc.time() - ptm)


sessionInfo()
## R version 3.5.2 (2018-12-20)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 18.04.1 LTS
##
## Matrix products: default
## BLAS: /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.7.1
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.7.1
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=C
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets
## [6] methods   base
##
## other attached packages:
## [1] knitr_1.21
##
## loaded via a namespace (and not attached):
##  [1] compiler_3.5.2    magrittr_1.5      formatR_1.5
##  [4] tools_3.5.2       R.rsp_0.43.0      stringi_1.2.4
##  [7] R.methodsS3_1.7.1 stringr_1.3.1     xfun_0.4
## [10] digest_0.6.18     R.cache_0.13.0    R.utils_2.7.0
## [13] evaluate_0.12     R.oo_1.22.0
```

References

Zhu Wang. Quadratic majorization for robust nonconvex loss with applications to variable selection. 2016. manuscript.