```
#########################################################################
###                  R-script for the manuscript                      ###
###                                                                   ###
###               Survival models with preclustered                   ###
###                  gene groups as covariates                        ###
###                                                                   ###
###                                                                   ###
#########################################################################



# The following R-Code consists of four model selection and evaluation functions (L1, L2, univ., forw.).
# In each function the evaluation procedure (see Section 3) is implemeted.
# The input for each function is a data matrix "x" (rows: samples, columns: genes/GO groups),
# a vector "time" containing the survival times and a vector "status" containing the non-censoring indicators of all patients.

# For usage of the functions, it is necessary to install the "survival" and the "penalized" package.



################################################
########## function for L1 regression ##########
################################################


L1.reg <- function(x, time, status){
    list.sel.L1 <<- list(NULL)          # save all selected covariates
    num.L1 <<- NULL                     # save the number of chosen covariates
    lambda.L1 <<- NULL                  # save the optimal tuning parameter lambda
    p.median.L1 <<- NULL                # save the p-value obtained from logrank test
    p.pi.L1 <<- NULL                    # save the p-value obtained from likelihood ratio test for the prognostic index

    count <- 0

    for (j in 1:1000){                  # in case of matrix failure caused by the profiling and/or optimation procedure (penalized package)
    if(!count == 100){
    print(count)

        id <- sample(dim(x)[1], round(2/3*dim(x)[1]))   # get IDs for training set (2/3 of all patients)

        ### training data
        x.train <- as.data.frame(x[id, ])               # training data matrix
        time.train <- matrix(time[id])                  # training data survival times
        status.train <- matrix(status[id])              # training data non-censoring indicator
        S.train <- Surv(time.train, status.train)       # survival object for training data

        ### test data
        x.test <- as.data.frame(x[-id, ])               # test data matrix
        time.test <- matrix(time[-id])                  # test data survival times
        status.test <- matrix(status[-id])              # test data non-censoring indicator
        S.test <- Surv(time.test, status.test)          # survival object for test data
```

```
# number of folds for cross-validation (cv)
k <- 10


########## L1 model selection

# calculate the profile for the cross-validated log likelihood (cvl)
# set minlambda1 in case of matrix inversion failure (approx. between 3 and 5)
prof.L1 <- try(profL1(S.train, x.train, minlambda1 = 5, fold = k))

if(!inherits(prof.L1, "try-error")){

    # profile plot for the cvl
    plot(prof.L1$lambda, prof.L1$cvl, type = "l")

    # first approximation for optimal lambda
    lambda.opt <- prof.L1$lambda[which.max(prof.L1$cvl)]
    print(lambda.opt)
    print(max(prof.L1$cvl))

    # define the range for the optimation procedure
    steplength <- prof.L1$lambda[2]-prof.L1$lambda[1]

    # find the optimal tuning parameter via cvl
    opt.L1 <- try(optL1(S.train, x.train, fold = prof.L1$fold, minlambda1 = lambda.opt-2*steplength,
                        maxlambda1 = lambda.opt+2*steplength))

    if(!inherits(opt.L1, "try-error")){

        # optimal lambda1
        lambda.L1 <- opt.L1$lambda

        # selected covariates
        sel.L1 <- names(coefficients(opt.L1$fullfit))
        if(!is.null(sel.L1)){
            count <- count + 1

            # number of selected covariates
            num.L1 <- length(sel.L1)

            # penalized object
            pen.L1 <- penalized(S.train, x.train, lambda1 = opt.L1$lambda)

            ########## evaluation for L1

            # risk score
            rs.L1 <- as.matrix(x.test[, sel.L1]) %*% as.matrix(coefficients(pen.L1))

            # median time
            good.prog.L1 <- (rs.L1 < median(rs.L1))
            fit.L1 <- survfit(S.test ~ good.prog.L1)

            # kaplan-meier curves (for visualization)
```

```
                    plot(fit.L1, lwd = 2, lty = c(1,1), col = c("red","blue"), xlab = 'Time (years)', ylab = 'Estimated Survival Function')
                    legend(60, 0.8, legend=c('PI > median', 'PI < median'), lty = c(1,1), col = c("red", "blue"), lwd = 2)
                    title("Kaplan-Meier Curves")

                    # logrank test and p-value
                    logrank.L1 <- survdiff(S.test ~ good.prog.L1, data = x.test, rho = 0)
                    p.median.L1 <- 1-pchisq(logrank.L1$chisq, 1)

                    # prognostic index (risk score) and p-value
                    pi.L1 <- coxph(S.test ~ rs.L1)
                    p.pi.L1 <- 1-pchisq(pi.L1$score, 1)

                } else {
                    count <- count
                    sel.L1 <- NA
                    num.L1 <- NA
                    lambda.L1 <- NA
                    p.median.L1 <- NA
                    p.pi.L1 <- NA
                }

            } else {
                count <- count
                sel.L1 <- NA
                num.L1 <- NA
                lambda.L1 <- NA
                p.median.L1 <- NA
                p.pi.L1 <- NA
            }
        } else {
            count <- count
            sel.L1 <- NA
            num.L1 <- NA
            lambda.L1 <- NA
            p.median.L1 <- NA
            p.pi.L1 <- NA
        }
        # evaluation values
        list.sel.L1[[j]] <<- sel.L1
        num.L1[j] <<- num.L1
        lambda.L1[j] <<- lambda.L1
        p.median.L1[j] <<- p.median.L1
        p.pi.L1[j] <<- p.pi.L1
    }
    }
}



################################################
########## function for L2 regression ##########
################################################
```

```
L2.reg <- function(x, time, status){
    lambda.L2 <<- NULL
    p.median.L2 <<- NULL
    p.pi.L2 <<- NULL

    count <- 0

    for (j in 1:1000){
    if(!count == 100){
    print(count)

        id <- sample(dim(x)[1], round(2/3*dim(x)[1]))

        ### training data
        x.train <- as.data.frame(x[id, ])
        time.train <- matrix(time[id])
        status.train <- matrix(status[id])
        S.train <- Surv(time.train, status.train)

        ### test data
        x.test <- as.data.frame(x[-id, ])
        time.test <- matrix(time[-id])
        status.test <- matrix(status[-id])
        S.test <- Surv(time.test, status.test)

        # number of folds for cv
        k <- 10


        ########## L2 model selection

        # find the optimal tuning parameter via cvl

        opt.L2 <- try(optL2(S.train, as.data.frame(x.train), fold = k))
        if(!inherits(opt.L2, "try-error")){
            lambda.L2 <- opt.L2$lambda
            pen.L2 <- try(penalized(S.train, x.train, lambda2 = opt.L2$lambda))
            if(!inherits(pen.L2, "try-error")){
                count <- count + 1

                ########### evaluation for L2

                # risk score
                rs.L2 <- as.matrix(x.test) %*% as.matrix(coefficients(pen.L2))

                # median time
                good.prog.L2 <- (rs.L2 < median(rs.L2))
                fit.L2 <- survfit(S.test ~ good.prog.L2)

                # kaplan-meier curves
                plot(fit.L2, lwd = 2, lty = c(1,1), col = c("red","blue"), xlab = 'Time (years)', ylab = 'Estimated Survival Function')
                legend(60, 0.8, legend=c('PI > median', 'PI < median'), lty = c(1,1), col = c("red", "blue"), lwd = 2)
```

```
                title("Kaplan-Meier Curves")

                # logrank test
                logrank.L2 <- survdiff(S.test ~ good.prog.L2, data = x.test, rho = 0)
                p.median.L2 <- 1-pchisq(logrank.L2$chisq, 1)

                # prognostic index (risk score)
                pi.L2 <- coxph(S.test ~ rs.L2)
                p.pi.L2 <- 1-pchisq(pi.L2$score, 1)
            } else {
                count <- count
                lambda.L2 <- NA
                p.median.L2 <- NA
                p.pi.L2 <- NA
            }
        } else {
        count <- count
        lambda.L2 <- NA
        p.median.L2 <- NA
        p.pi.L2 <- NA
        }

        # evaluation values
        lambda.L2[j] <<- lambda.L2
        p.median.L2[j] <<- p.median.L2
        p.pi.L2[j] <<- p.pi.L2
    }
    }
}



#########################################################
########## function for univariate selection ##########
#########################################################


uni.reg <- function(x, time, status){
    list.sel.uni <<- list(NULL)
    num.uni <<- NULL
    lambda.uni <<- NULL
    p.median.uni <<- NULL
    p.pi.uni <<- NULL

    count <- 0

    for (j in 1:1000){
    if(!count == 100){
    print(count)

        id <- sample(dim(x)[1], round(2/3*dim(x)[1]))

        ### training data
```

```
x.train <- as.data.frame(x[id, ])
time.train <- matrix(time[id])
status.train <- matrix(status[id])
S.train <- Surv(time.train, status.train)

### test data
x.test <- as.data.frame(x[-id, ])
time.test <- matrix(time[-id])
status.test <- matrix(status[-id])
S.test <- Surv(time.test, status.test)

# number of folds for cv
k <- 10
# number of covariates included
m <- dim(x)[2]


########## univariate selection

cox.p.uni <- NULL
for (i in 1:m){
        t <- coxph(S.train ~ x.train[, i], method = "breslow")
        cox.p.uni[i] <- 1-pchisq(t$score, 1)
    }
sel.uni <- order(cox.p.uni)

# cvl for the first covariates according to smallest p-value
cvl <- NULL
i <- 1
fit <- cvl(S.train, x.train[, sel.uni[i]], fold = k)
cvl[1] <- fit$cvl
while (cvl[i] > -Inf) {
    i <- i+1
    fit <- try(cvl(S.train, x.train[, sel.uni[1:i]], fold = k))
    if(!inherits(fit, "try-error")){
        cvl[i] <- fit$cvl
    }
    else{
    break
    }
}

# optimal tuning parameter
lambda.uni <- which.max(cvl)

# covariates included in the model
sel.uni <- sel.uni[1:lambda.uni]
num.uni <- length(sel.uni)

# creat a penalized object (= coxph(method = "breslow")) with optimal lambda.uni
pen.uni <- try(penalized(S.train, x.train[, sel.uni], data = x.train))
if(!inherits(pen.uni, "try-error")){
    count <- count + 1
```

```
            ######### evaluation for univariate selection

            # risk score
            rs.uni <- as.matrix(x.test[, sel.uni]) %*% as.matrix(coefficients(pen.uni))

            # median time
            good.prog.uni <- (rs.uni < median(rs.uni))
            fit.uni <- survfit(S.test ~ good.prog.uni)

            # kaplan-meier curves
            plot(fit.uni, lwd = 2, lty = c(1,1), col = c("red","blue"), xlab = 'Time (years)', ylab = 'Estimated Survival Function')
            legend(15, 0.8, legend=c('PI > median', 'PI < median'), lty = c(1,1), col = c("red", "blue"), lwd = 2)
            title("Kaplan-Meier Curves")

            # logrank test
            logrank.uni <- survdiff(S.test ~ good.prog.uni, rho = 0, data = x.test)
            p.median.uni <- 1-pchisq(logrank.uni$chisq, 1)

            # prognostic index (risk score)
            pi.uni <- coxph(S.test ~ rs.uni, method = "breslow")
            p.pi.uni <- 1-pchisq(pi.uni$score, 1)
        } else {
            count <- count
            sel.uni <- NA
            num.uni <- NA
            lambda.uni <- NA
            p.median.uni <- NA
            p.pi.uni <- NA
        }

        # evaluation values
        list.sel.uni[[j]] <<- sel.uni
        num.uni[j] <<- num.uni
        lambda.uni[j] <<- lambda.uni
        p.median.uni[j] <<- p.median.uni
        p.pi.uni[j] <<- p.pi.uni
    }
    }
}




#########################################################
########## function for forward selection #############
#########################################################


forw.reg <- function(x, time, status){
    list.sel.forw <<- list(NULL)
    num.forw <<- NULL
    lambda.forw <<- NULL
```

```
p.median.forw <<- NULL
p.pi.forw <<- NULL

count <- 0

for (j in 1:100){
print(count)

    id <- sample(dim(x)[1], round(2/3*dim(x)[1]))

    ### training data
    x.train <- as.data.frame(x[id, ])
    time.train <- matrix(time[id])
    status.train <- matrix(status[id])
    S.train <- Surv(time.train, status.train)

    ### test data
    x.test <- as.data.frame(x[-id, ])
    time.test <- matrix(time[-id])
    status.test <- matrix(status[-id])
    S.test <- Surv(time.test, status.test)

    # number of folds for cv
    k <- 10
    # number of covariates included
    m <- dim(x)[2]



    ######################### forward selection

    # first covariate selected via univariate approach
    cox.p.uni <- NULL
    for (i in 1:m){
            t <- coxph(S.train ~ x.train[, i], method = "breslow")
            cox.p.uni[i] <- 1-pchisq(t$score, 1)
        }

    # include covariates step by step (max 100 covariates)
    x.train <- as.matrix(x.train)
    active.set <- NULL
    for (z in 1:100){
        cox.p <- rep(Inf, m)
            for (i in (1:m)[!(1:m)%in%active.set]){
                t <- coxph(S.train ~ x.train[, c(active.set, i)], method = "breslow")
                cox.p[i] <- 1-pchisq(t$score, 1)
            }
        print(min(cox.p))
        if(min(cox.p)>0){
            active.set <- c(active.set, which.min(cox.p))
        }else{
            active.set <- c(active.set, setdiff(order(cox.p.uni), active.set))
            break
```

```
    }
}


# cvl for the first covariates according to smallest p-value
cvl <- NULL
i <- 1
fit <- cvl(S.train, x.train[, active.set[1:i]], fold = k)
cvl[1] <- fit$cvl
while (cvl[i] > -Inf) {
    i <- i+1
    fit <- try(cvl(S.train, x.train[, active.set[1:i]], fold = k))
    if(!inherits(fit, "try-error")){
        cvl[i] <- fit$cvl
    } else{
    break
    }
}


# optimal tuning parameter
lambda.forw <- which.max(cvl)

# covariates included in the model
sel.forw <- active.set[1:lambda.forw]
num.forw <- length(sel.forw)

# create a penalized object (= coxph(method = "breslow")) with optimal lambda.forw
pen.forw <- try(penalized(S.train, x.train[, sel.forw], data = as.data.frame(x.train)))
if(!inherits(pen.forw, "try-error")){

    ########## evaluation for forward selection

    # risk score
    rs.forw <- as.matrix(x.test[, sel.forw]) %*% as.matrix(coefficients(pen.forw))

    # median time
    good.prog.forw <- (rs.forw < median(rs.forw))
    fit.forw <- survfit(S.test ~ good.prog.forw)

    # kaplan-meier curves
    plot(fit.forw, lwd = 2, lty = c(1,1), col = c("red","blue"), xlab = 'Time (years)', ylab = 'Estimated Survival Function')
    legend(15, 0.8, legend=c('PI > median', 'PI < median'), lty = c(1,1), col = c("red", "blue"), lwd = 2)
    title("Kaplan-Meier Curves")

    # logrank test
    logrank.forw <- survdiff(S.test ~ good.prog.forw, rho = 0, data = x.test)
    p.median.forw <- 1-pchisq(logrank.forw$chisq, 1)

    # prognostic index (risk score)
    pi.forw <- coxph(S.test ~ rs.forw, method = "breslow")
    p.pi.forw <- 1-pchisq(pi.forw$score, 1)
```

```
        } else {
            count <- count
            sel.forw <- NA
            num.forw <- NA
            lambda.forw <- NA
            p.median.forw <- NA
            p.pi.forw <- NA
        }

        # evaluation values
        list.sel.forw[[j]] <<- sel.forw
        num.forw[j] <<- num.forw
        lambda.forw[j] <<- lambda.forw
        p.median.forw[j] <<- p.median.forw
        p.pi.forw[j] <<- p.pi.forw

        count <- count + 1
    }
}
```