Variable selection with the strong heredity constraint and its oracle property

Sahir Bhatnagar

December 6, 2015

Trying to code the Strong Heredity Interaction Model (SHIM) in Choi et al. (2010).

1 Simulate the data

```
set.seed(123456)

# number of predictors
p = 10

# number of test subjects
n = 200

# correlation between X's
rho = 0.5

# signal to noise ratio
signal_to_noise_ratio = 4

# names of the main effects, this will be used in many of the
# functions
main_effect_names <- paste0("x", 1:p)

# names of the active set
true_var_names <- c("x1", "x2", "x3", "x4", "x1:x2", "x1:x3",
    "x1:x4", "x2:x3", "x2:x4", "x3:x4")

# different true coefficient vectors as in Table 1 of Choi et
```

```r
# al.
beta1 <- c(7, 2, 1, 1, 0, 0, 0, 0, 0, 0) %>% magrittr::set_names(true_var_names)
beta2 <- c(7, 2, 1, 1, 1, 0, 0, 0.5, 0.4, 0.1) %>% magrittr::set_names(true_var_names)
beta3 <- c(7, 2, 1, 1, 7, 7, 7, 2, 2, 1) %>% magrittr::set_names(true_var_names)
beta4 <- c(7, 2, 1, 1, 14, 14, 14, 4, 4, 2) %>% magrittr::set_names(true_var_names)
beta5 <- c(0, 0, 0, 0, 7, 7, 7, 2, 2, 1) %>% magrittr::set_names(true_var_names)

# simulate Toeplitz like correlation structure between X's
H <- abs(outer(1:p, 1:p, "-"))
cor <- rho^H

# generate X's from multivariate normal and label the matrix
DT <- MASS::mvrnorm(n = n, mu = rep(0, p), Sigma = cor) %>% magrittr::set_colnames(paste0("x",
    1:p)) %>% set_rownames(paste0("Subject", 1:n))

# create X matrix which contains all main effects and
# interactions but not the intercept each column is
# standardized to mean 0 and sd 1
X <- model.matrix(as.formula(paste0("~(", paste0(main_effect_names,
    collapse = "+"), ")^2-1")), data = DT %>% as.data.frame()) %>%
    scale

# check that means of columns are 0 and sd 1
colMeans(X) %>% sum
```

```
## [1] 1.6e-16
```

```r
apply(X, 2, sd) %>% sum
```

```
## [1] 55
```

```r
# generate response with user defined signal to noise ratio
# and center the response
y.star <- X[, names(beta4)] %*% beta4
error <- rnorm(n)
k <- sqrt(var(y.star)/(signal_to_noise_ratio * var(error)))
Y <- (y.star + k * error) %>% scale(center = TRUE, scale = FALSE)
colnames(Y) <- "Y"

# record mean of response before centering
(b0 <- mean(y.star + k * error))
```

```
## [1] 0.27
```

```
# names of interaction variables assuming interaction terms
# contain a ':' this will be used in many of the functions
interaction_names <- colnames(X) %>% grep(":", ., value = T)
```

References

Nam Hee Choi, William Li, and Ji Zhu. Variable selection with the strong heredity constraint and its oracle property. *Journal of the American Statistical Association*, 105(489):354–364, 2010. 1

A R Code

```
sessionInfo()
getPckg <- function(pckg) install.packages(pckg, repos = "http://cran.r-project.org")

pckg = try(require(knitr))
if (!pckg) {
    cat("Installing 'knitr' from CRAN\n")
    getPckg("knitr")
    require(knitr)
}

pckg = try(require(data.table))
if (!pckg) {
    cat("Installing 'data.table' from CRAN\n")
    getPckg("data.table")
    require(data.table)
}

pckg = try(require(magrittr))
if (!pckg) {
    cat("Installing 'magrittr' from CRAN\n")
    getPckg("magrittr")
    require(magrittr)
}

pckg = try(require(glmnet))
if (!pckg) {
    cat("Installing 'glmnet' from CRAN\n")
    getPckg("glmnet")
    require(glmnet)
}

pckg = try(require(stringr))
if (!pckg) {
    cat("Installing 'stringr' from CRAN\n")
    getPckg("stringr")
    require(stringr)
}
```

```
pckg = try(require(plyr))
if (!pckg) {
    cat("Installing 'plyr' from CRAN\n")
    getPckg("plyr")
    require(plyr)
}
################################### R source code file used to create simulated data from Choi
################################### et al 2009 JASA Created by Sahir, November 5, 2015 Updated
################################### Dec 6th, 2015 hosted on Github repo
################################### 'sahirbhatnagar/interactions' NOTE:
set.seed(123456)

# number of predictors
p = 10

# number of test subjects
n = 200

# correlation between X's
rho = 0.5

# signal to noise ratio
signal_to_noise_ratio = 4

# names of the main effects, this will be used in many of the
# functions
main_effect_names <- paste0("x", 1:p)

# names of the active set
true_var_names <- c("x1", "x2", "x3", "x4", "x1:x2", "x1:x3",
    "x1:x4", "x2:x3", "x2:x4", "x3:x4")

# different true coefficient vectors as in Table 1 of Choi et
# al.
beta1 <- c(7, 2, 1, 1, 0, 0, 0, 0, 0, 0) %>% magrittr::set_names(true_var_names)
beta2 <- c(7, 2, 1, 1, 1, 0, 0, 0.5, 0.4, 0.1) %>% magrittr::set_names(true_var_names)
beta3 <- c(7, 2, 1, 1, 7, 7, 7, 2, 2, 1) %>% magrittr::set_names(true_var_names)
beta4 <- c(7, 2, 1, 1, 14, 14, 14, 4, 4, 2) %>% magrittr::set_names(true_var_names)
beta5 <- c(0, 0, 0, 0, 7, 7, 7, 2, 2, 1) %>% magrittr::set_names(true_var_names)
```

```r
# simulate Toeplitz like correlation structure between X's
H <- abs(outer(1:p, 1:p, "-"))
cor <- rho^H

# generate X's from multivariate normal and label the matrix
DT <- MASS::mvrnorm(n = n, mu = rep(0, p), Sigma = cor) %>% magrittr::set_colnames(paste0("x",
    1:p)) %>% set_rownames(paste0("Subject", 1:n))

# create X matrix which contains all main effects and
# interactions but not the intercept each column is
# standardized to mean 0 and sd 1
X <- model.matrix(as.formula(paste0("~(", paste0(main_effect_names,
    collapse = "+"), ")^2-1")), data = DT %>% as.data.frame()) %>%
    scale

# check that means of columns are 0 and sd 1
colMeans(X) %>% sum
apply(X, 2, sd) %>% sum

# generate response with user defined signal to noise ratio
# and center the response
y.star <- X[, names(beta4)] %*% beta4
error <- rnorm(n)
k <- sqrt(var(y.star)/(signal_to_noise_ratio * var(error)))
Y <- (y.star + k * error) %>% scale(center = TRUE, scale = FALSE)
colnames(Y) <- "Y"

# record mean of response before centering
(b0 <- mean(y.star + k * error))

# names of interaction variables assuming interaction terms
# contain a ':' this will be used in many of the functions
interaction_names <- colnames(X) %>% grep(":", ., value = T)
#' variables: character vector of variable names for which you want the univariate regression est
#' must be contained in the column names of x
#' x: matrix that includes all data corresponding to variables with corresponding column names
#' y: response (matrix form)
#' returns OLS coefficients as a p x 1 data.frame
```

```r
uni_fun <- function(variables, x, y) {



    res <- plyr::ldply(variables, function(i) {
        fit <- lm.fit(x = x[, i, drop = F], y = y)  # dont need to add intercept because y has be
        fit$coefficients[1]
    }) %>% magrittr::set_rownames(variables) %>% magrittr::set_colnames("univariate_beta") %>%
        as.matrix
    return(res)

}

# function that takes a vector of betas (which are the main
# effects) and alphas which are the interaction effects, and
# outputs the main effects, and converts the alphas to gammas
# note that alpha_ij = gamma_ij * beta_i*beta_j, i < j
convert <- function(betas.and.alphas, main.effect.names, interaction.names,
    epsilon = 1e-04) {

    # betas.and.alphas is the result from uni_fun create output
    # matrix
    betas_and_gammas <- matrix(nrow = nrow(betas.and.alphas)) %>%
        magrittr::set_rownames(rownames(betas.and.alphas))

    for (k in interaction.names) {
        # get names of main effects corresponding to interaction
        (main <- betas.and.alphas[k, , drop = F] %>% rownames %>%
            stringr::str_split(":") %>% unlist)
        # convert alpha to gamma BUT NEED TO CHECK IF BETAS ARE 0!!!!
        betas_and_gammas[k, ] <- if (any(abs(betas.and.alphas[main,
            ]) < epsilon))
            0 else betas.and.alphas[k, ]/prod(betas.and.alphas[main,
            ])
    }

    # add back the main effects which dont need to be transformed
    for (j in main.effect.names) {
        betas_and_gammas[j, ] <- betas.and.alphas[j, ]
```

```
    }

    return(betas_and_gammas)

}

# function that converts gammas to alphas
convert2 <- function(betas.and.gammas, main.effect.names, interaction.names) {

    # betas.and.gammas = rbind2(beta_hat_next,
    # gamma_hat_previous) betas.and.gammas is the result from
    # shim function create output matrix
    betas.and.alphas <- matrix(nrow = nrow(betas.and.gammas)) %>%
        magrittr::set_rownames(rownames(betas.and.gammas))

    for (k in interaction.names) {
        # k='x1:x10' get names of main effects corresponding to
        # interaction
        (main <- betas.and.gammas[k, , drop = F] %>% rownames %>%
            stringr::str_split(":") %>% unlist)
        # convert alpha to gamma
        betas.and.alphas[k, ] <- betas.and.gammas[k, ] * prod(betas.and.gammas[main,
            ])
    }

    # add back the main effects which dont need to be transformed
    for (j in main.effect.names) {
        betas.and.alphas[j, ] <- betas.and.gammas[j, ]
    }

    return(betas.and.alphas)

}

# function to calculate xtilde in step 3 of algorithm
xtilde <- function(interaction.names, data.main.effects, beta.main.effects) {

    #' interaction.names character vector of interaction names
    #' data.main.effects data frame or matrix containing the main effects data
```

```
    #' beta.main.effects data frame or matrix containing the coefficients of main effects
    #' Only the main effects are used in this step,
    #' however you can provide this function, either the betas.and.alphas or
    #' betas.and.gammas because only the betas (main effect) paramters
    #' are used in the calculation of xtilde

    # create output matrix
    xtildas <- matrix(ncol = length(interaction.names), nrow = nrow(data.main.effects)) %>%
        magrittr::set_colnames(interaction.names)

    for (k in interaction.names) {
        # get names of main effects corresponding to interaction
        (main <- k %>% stringr::str_split(":") %>% unlist)

        # step 3 to calculate x tilda
        xtildas[, k] <- prod(beta.main.effects[main, ]) * data.main.effects[,
            main[1], drop = F] * data.main.effects[, main[2],
            drop = F]
    }

    return(xtildas)

}

# adaptive weights used in fitting algorithm, intercept set
# to 0 because Y is centered
ridge_weights <- function(x, y, main.effect.names, interaction.names) {
    #' interaction.names character vector of interaction names
    #' main.effect.names character vector of main effect names
    #' must be contained in the column names of x
    #' x: matrix that includes all data corresponding to variables with corresponding column names
    #' all columns have mean 0 and variance 1
    #' y: centered response (matrix form)
    #' returns ridge weights as a p x 1 data.frame
    #' STILL NEED TO DECIDE IF I SHOULD MULTIPLY WEIGHTS BY LOG(N)/N AS DONE IN SECTION 3.1 AND I
    #' PAPER BY WANG LI AND TSAI 2007 JRSSB

    n <- length(y)
    # fit the ridge to get betas and alphas
```

```
    (fit <- glmnet::cv.glmnet(x = x, y = y, alpha = 0, standardize = F,
        intercept = F))
    # remove intercept
    (betas.and.alphas <- coef(fit, s = "lambda.1se") %>% as.matrix() %>%
        magrittr::extract(-1, , drop = F))

    # (fit <- lm.fit(x = x, y = y)) # remove intercept
    # (betas.and.alphas <- coef(fit) %>% as.matrix())

    # create output matrix
    weights <- matrix(nrow = nrow(betas.and.alphas)) %>% magrittr::set_rownames(rownames(betas.and

    # main effects weights
    for (j in main.effect.names) {
        weights[j, ] <- abs(1/betas.and.alphas[j, ])  #* log(n)/n
    }

    for (k in interaction.names) {
        # get names of main effects corresponding to interaction
        (main <- betas.and.alphas[k, , drop = F] %>% rownames %>%
            stringr::str_split(":") %>% unlist)
        weights[k, ] <- abs(prod(betas.and.alphas[main, ])/betas.and.alphas[k,
            ])  #* log(n)/n
    }

    return(weights)

}

soft <- function(x, y, beta, lambda, weight) {
    # user must supply x AND y, or beta.. but not both i set it
    # up this way because to get the sequence of lambdas, I use
    # the beta argument so that I only compute this once. I use
    # the x, y argument for the CV folds lambda can be a vector
    # and this functions will return each thresholded beta for
    # each lambda e.g. soft(0.25, lambda =
    # seq(0.001,0.65,length.out = 100), 1.5)

    if (missing(x) & missing(y) & missing(beta))
```

```
        stop("user must supply x AND y, or beta but not both")
    if (missing(x) & missing(y))
        return(list(beta = sign(beta) * pmax(0, abs(beta) - lambda *
            weight)))
    if (missing(beta)) {
        # (beta <- lm.fit(x = cbind2(rep(1, length(y)),x), y = y) %>%
        # coef %>% magrittr::extract(2))
        (beta <- lm.fit(x = x[, 1, drop = F], y = y) %>% coef %>%
            magrittr::extract(1))

        # lm.fit(x = cbind2(rep(1, length(y_tilde_2)),x_tilde_2), y =
        # y_tilde_2) %>% coef %>% magrittr::extract(2)
        b_lasso <- sign(beta) * pmax(0, abs(beta) - lambda *
            weight)
        # return(list('beta' = b_lasso, 'df' = nonzero(b_lasso)))
        return(b_lasso)
    }

}

cv_lspath <- function(outlist, lambda, x, y, foldid) {
    # typenames <- c(misclass = 'Misclassification Error', loss =
    # 'Margin Based Loss') if (pred.loss == 'default') pred.loss
    # <- 'loss' if (!match(pred.loss, c('loss'), FALSE)) {
    # warning('Only 'loss' available for least squares
    # regression; 'loss' used') pred.loss <- 'loss' }
    y <- as.double(y)
    nfolds <- max(foldid)
    predmat <- matrix(NA, length(y), length(lambda))
    nlams <- double(nfolds)
    for (i in seq(nfolds)) {
        # i=1
        which <- foldid == i
        # this gives be the fitted object for each CV fold
        fitobj <- outlist[[i]]

        # this gives the predicted responses for the subjects in the
        # held-out fold for each lambda so if each fold has 20
        # subjects, and there are 100 lambdas, then this will return
```

```
        # a 20 x 100 matrix
        preds <- x[which, , drop = FALSE] %*% t(fitobj$beta)
        # preds <- predict(fitobj, x[which, , drop = FALSE], type =
        # 'link')
        nlami <- length(fitobj$lambda)
        predmat[which, seq(nlami)] <- preds
        nlams[i] <- nlami
    }
    cvraw <- (y - predmat)^2
    cvob <- cvcompute(cvraw, foldid, nlams)
    cvraw <- cvob$cvraw
    N <- cvob$N
    cvm <- apply(cvraw, 2, mean, na.rm = TRUE)
    cvsd <- sqrt(apply(scale(cvraw, cvm, FALSE)^2, 2, mean, na.rm = TRUE)/(N -
        1))
    list(cvm = cvm, cvsd = cvsd, name = "MSE")
}

cvcompute <- function(mat, foldid, nlams) {
    nfolds <- max(foldid)
    outmat <- matrix(NA, nfolds, ncol(mat))
    good <- matrix(0, nfolds, ncol(mat))
    mat[is.infinite(mat)] <- NA
    for (i in seq(nfolds)) {
        mati <- mat[foldid == i, ]
        outmat[i, ] <- apply(mati, 2, mean, na.rm = TRUE)
        good[i, seq(nlams[i])] <- 1
    }
    N <- apply(good, 2, sum)
    list(cvraw = outmat, N = N)
}

lamfix <- function(lam) {
    llam <- log(lam)
    lam[1] <- exp(2 * llam[2] - llam[3])
    lam
}

nonzero <- function(beta, bystep = FALSE) {
```

```
    beta <- as.matrix(beta)
    nr = nrow(beta)
    if (nr == 1) {
        if (bystep)
            apply(beta, 2, function(x) if (abs(x) > 0)
                1 else NULL) else {
            if (any(abs(beta) > 0))
                1 else NULL
        }
    } else {
        beta = abs(beta) > 0
        which = seq(nr)
        ones = rep(1, ncol(beta))
        nz = as.vector((beta %*% ones) > 0)
        which = which[nz]
        if (bystep) {
            if (length(which) > 0) {
                beta = as.matrix(beta[which, , drop = FALSE])
                nzel = function(x, which) if (any(x))
                  which[x] else NULL
                which = apply(beta, 2, nzel, which)
                if (!is.list(which))
                  which = data.frame(which)
                which
            } else {
                dn = dimnames(beta)[[2]]
                which = vector("list", length(dn))
                names(which) = dn
                which
            }
        } else which
    }
}

getmin <- function(lambda, cvm, cvsd) {
    cvmin <- min(cvm)
    idmin <- cvm <= cvmin
    lambda.min <- max(lambda[idmin])
    idmin <- match(lambda.min, lambda)
```

```
    semin <- (cvm + cvsd)[idmin]
    idmin <- cvm <= semin
    lambda.1se <- max(lambda[idmin])
    list(lambda.min = lambda.min, lambda.1se = lambda.1se)
}

lambda.interp <- function(lambda, s) {
    if (length(lambda) == 1) {
        nums <- length(s)
        left <- rep(1, nums)
        right <- left
        sfrac <- rep(1, nums)
    } else {
        s[s > max(lambda)] <- max(lambda)
        s[s < min(lambda)] <- min(lambda)
        k <- length(lambda)
        sfrac <- (lambda[1] - s)/(lambda[1] - lambda[k])
        lambda <- (lambda[1] - lambda)/(lambda[1] - lambda[k])
        coord <- approx(lambda, seq(lambda), sfrac)$y
        left <- floor(coord)
        right <- ceiling(coord)
        sfrac <- (sfrac - lambda[right])/(lambda[left] - lambda[right])
        sfrac[left == right] <- 1
    }
    list(left = left, right = right, frac = sfrac)
}

plot.cv.uninet <- function(x, sign.lambda = 1, ...) {
    cvobj <- x
    xlab <- "log(Lambda)"
    if (sign.lambda < 0)
        xlab <- paste("-", xlab, sep = "")
    plot.args <- list(x = sign.lambda * log(cvobj$lambda), y = cvobj$cvm,
        ylim = range(cvobj$cvupper, cvobj$cvlo), xlab = xlab,
        ylab = cvobj$name, type = "n")
    new.args <- list(...)
    if (length(new.args))
        plot.args[names(new.args)] <- new.args
    do.call("plot", plot.args)
```

```
    error.bars(sign.lambda * log(cvobj$lambda), cvobj$cvupper,
        cvobj$cvlo, width = 0.01, col = "darkgrey")
    points(sign.lambda * log(cvobj$lambda), cvobj$cvm, pch = 20,
        col = "red")
    axis(side = 3, at = sign.lambda * log(cvobj$lambda), labels = paste(cvobj$nz),
        tick = FALSE, line = 0)
    abline(v = sign.lambda * log(cvobj$lambda.min), lty = 3)
    abline(v = sign.lambda * log(cvobj$lambda.1se), lty = 3)
    invisible()
}

coef.cv.uninet <- function(object, s = c("lambda.1se", "lambda.min"),
    ...) {
    if (is.numeric(s))
        lambda <- s else if (is.character(s)) {
        s <- match.arg(s)
        lambda <- object[[s]]
    } else stop("Invalid form for s")
    coef.uninet(object$uninet.fit, s = lambda, ...)
}

coef.uninet <- function(object, s = NULL, type = c("coefficients",
    "nonzero"), ...) {
    type <- match.arg(type)
    # b0 <- t(as.matrix(object£b0)) rownames(b0) <- '(Intercept)'
    nbeta <- rbind2(object$beta)
    if (!is.null(s)) {
        vnames <- dimnames(nbeta)[[1]]
        # dimnames(nbeta) <- list(NULL, NULL)
        lambda <- object$lambda
        lamlist <- lambda.interp(lambda, s)
        nbeta <- nbeta[, lamlist$left, drop = FALSE] * lamlist$frac +
            nbeta[, lamlist$right, drop = FALSE] * (1 - lamlist$frac)
        # dimnames(nbeta) <- list(vnames, paste(seq(along = s)))
    }
    if (type == "coefficients")
        return(nbeta)
    if (type == "nonzero")
        return(nonzero(nbeta[1, , drop = FALSE], bystep = TRUE))
```

```
}

# fits weighted lasso with single predictor and no intercept
uninet <- function(x, y, nlambda = 100, method = c("ls"), lambda.factor = ifelse(nobs <
    nvars, 0.01, 1e-04), lambda = NULL, penalty.factor = rep(1,
    nvars)) {

    # x=x_tilde_2; y=y_tilde_2; nlambda = 100; method = c('ls');
    # lambda.factor = ifelse(nobs < nvars, 0.01, 1e-04); lambda =
    # NULL; penalty.factor = rep(1, nvars); standardize = TRUE;
    # eps = 1e-08; maxit = 1e+06

    method <- match.arg(method)
    this.call <- match.call()
    y <- drop(y)
    x <- as.matrix(x)
    np <- dim(x)
    nobs <- as.integer(np[1])
    nvars <- as.integer(np[2])
    vnames <- colnames(x)
    N = nrow(x)

    if (is.null(vnames))
        vnames <- paste("V", seq(nvars), sep = "")
    if (length(y) != nobs)
        stop("x and y have different number of observations")
    if (length(penalty.factor) != nvars)
        stop("The size of L1 penalty factor must be same as the number of input variables")
    # if (lambda2 < 0) stop('lambda2 must be non-negative') maxit
    # <- as.integer(maxit) lam2 <- as.double(lambda2)
    penalty.factor <- as.double(penalty.factor)
    # pf2 <- as.double(pf2) isd <- as.integer(standardize) eps <-
    # as.double(eps) dfmax <- as.integer(dfmax) pmax <-
    # as.integer(pmax) if (!missing(exclude)) { jd <-
    # match(exclude, seq(nvars), 0) if (!all(jd > 0)) stop('Some
    # excluded variables out of range') jd <-
    # as.integer(c(length(jd), jd)) } else jd <- as.integer(0)

    if (is.null(lambda)) {
```

```
        if (lambda.factor >= 1)
            stop("lambda.factor should be less than 1")
        # flmin <- as.double(lambda.factor)
        max_lam <- double(1)
        # loop to figure out max lambda its the minimum lambda that
        # gives a beta of 0
        beta_tilde <- lm.fit(x = x, y = y) %>% coef
        # ulam = 0
        b = 1
        while (b != 0) {
            max_lam <- max_lam + 0.001
            b <- soft(beta = beta_tilde, lambda = max_lam, weight = penalty.factor)
        }

        lambda <- seq(lambda.factor * max_lam, max_lam, length.out = 100)
    } else {
        flmin <- as.double(1)
        if (any(lambda < 0))
            stop("lambdas should be non-negative")
        ulam <- as.double(rev(sort(lambda)))
        nlam <- as.integer(length(lambda))
    }

    fit <- switch(method, ls = soft(x = x, y = y, lambda = lambda,
        weight = penalty.factor))

    fit$lambda <- if (is.null(lambda))
        lamfix(lambda) else lambda
    fit$call <- this.call
    # fit£df <- nonzero(fit£beta)
    class(fit) <- c("uninet", class(fit))
    fit
}


cv.uninet <- function(x, y, lambda = NULL, nfolds = 10, foldid,
    ...) {

    # penalty.factor = adaptive.weights[colnames(x),] lambda =
```

```r
# NULL x = x_tilde_2 ; y = y_tilde_2
np <- dim(x)
nobs <- as.integer(np[1])
nvars <- as.integer(np[2])
# lambda.factor = ifelse(nobs < nvars, 0.01, 1e-04) nfolds =
# 10
N = nrow(x)
y <- drop(y)

# if (is.null(lambda)) { if (lambda.factor >= 1)
# stop('lambda.factor should be less than 1') #flmin <-
# as.double(lambda.factor) ulam <- double(1) } # loop to
# figure out max lambda # its the minimum lambda that gives a
# beta of 0 beta_tilde <- lm.fit(x = x, y = y) %>% coef #ulam
# = 0 b = 1 while (b != 0) { ulam <- ulam + 0.001 b <-
# soft(beta = beta_tilde, lambda = ulam, weight =
# penalty.factor) } lambda <- seq(lambda.factor*ulam, ulam,
# length.out = 100)

# betas for each lambda sapply(lambda, function(i) soft(beta
# = beta_tilde, lambda = i, weight = penalty.factor))

uninet.object <- uninet(x, y, lambda = lambda, ...)

# uninet.object <- uninet(x, y, lambda = lambda,
# penalty.factor = adaptive.weights[colnames(x_tilde_2),] )

lambda <- uninet.object$lambda
nz <- sapply(coef(uninet.object, type = "nonzero"), length)


if (missing(foldid))
    foldid = sample(rep(seq(nfolds), length = N)) else nfolds = max(foldid)
if (nfolds < 3)
    stop("nfolds must be bigger than 3; nfolds=10 recommended")
outlist = as.list(seq(nfolds))

for (i in seq(nfolds)) {
    which = foldid == i
```

```
        if (is.matrix(y))
            y_sub = y[!which, ] else y_sub = y[!which]
        outlist[[i]] = uninet(x = x[!which, , drop = FALSE],
            y = y_sub, lambda = lambda, ...)
    }

    cvstuff <- do.call(cv_lspath, list(outlist, lambda, x, y,
        foldid))
    cvm <- cvstuff$cvm
    cvsd <- cvstuff$cvsd
    cvname <- cvstuff$name
    out <- list(lambda = lambda, cvm = cvm, cvsd = cvsd, cvupper = cvm +
        cvsd, cvlo = cvm - cvsd, name = cvname, nzero = nz, name = cvname,
        uninet.fit = uninet.object)
    lamin <- getmin(lambda, cvm, cvsd)
    obj <- c(out, as.list(lamin))
    class(obj) <- "cv.uninet"
    obj
}

Q_theta <- function(x, y, beta, gamma, weights, lambda.beta,
    lambda.gamma, main.effect.names, interaction.names) {

    betas.and.alphas <- convert2(betas.and.gammas = rbind2(beta,
        gamma), main.effect.names = main.effect.names, interaction.names = interaction.names)
    crossprod(y - x %*% betas.and.alphas - b0) + lambda.beta *
        (crossprod(weights[main.effect.names, ], abs(beta))) +
        lambda.gamma * (crossprod(weights[interaction.names,
            ], abs(gamma)))
}
```

B Session Information

```
sessionInfo()
```

```
## R version 3.2.2 (2015-08-14)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 14.04 LTS
##
## locale:
##  [1] LC_CTYPE=en_CA.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_CA.UTF-8        LC_COLLATE=en_CA.UTF-8
##  [5] LC_MONETARY=en_CA.UTF-8    LC_MESSAGES=en_CA.UTF-8
##  [7] LC_PAPER=en_CA.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_CA.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets
## [6] base
##
## other attached packages:
## [1] plyr_1.8.3       stringr_1.0.0    glmnet_2.0-2
## [4] foreach_1.4.3    Matrix_1.2-2     magrittr_1.5
## [7] data.table_1.9.4 knitr_1.11
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.2      lattice_0.20-33  codetools_0.2-14
##  [4] MASS_7.3-44      chron_2.3-45     grid_3.2.2
##  [7] formatR_1.2.1    evaluate_0.8     highr_0.5.1
## [10] stringi_1.0-1    reshape2_1.4.1   iterators_1.0.8
## [13] tools_3.2.2      methods_3.2.2
```