# APPENDIX

#APPENDIX A

Optimization of hyper-parameters for machine learning methods.

Machine learning methods have method-specific parameters (called hyper-parameters) that are user-determined, and unlike the usual parameters of regression models, not optimized during the training of the model. Optimal values for hyper-parameters are usually determined by tuning: a range of values for each hyper-parameter are prescribed; models differing in these values are optimized on the training data, and the model with best cross-validated performance is selected as the final model. The range of values is usually determined either by a grid search through user-determined values or a random search through an optimal range of values specified by the statistical package. Random search is better for certain algorithms, especially at higher dimensions (Bergstra and Bengio, 2012). We use random search, except where grid search is appropriate for the method in question.

LASSO has one parameter λ, which we tune with a random search, sampling 30 values over a uniform distribution between 2^{-10} to 2^3, the default range for the random search in the caret package. For the elastic net, we do a random search for two hyper-parameters: λ (range same as for LASSO), and α, sampled randomly between values of 0 and 1. We tune two hyper-parameters for MARS: *nprune*, which specifies the maximum number of possible terms in the final pruned model, and *degree*, the maximum degree of interactions. Random search samples 30 values of nprune over a range from 2 to the number of predictors, examining first and 2nd degree interactions.

SVM was evaluated using radial basis kernels. The hyper-parameters we tune include C, which determines a trade-off between training error and model complexity; σ which controls non-linear nature of the decision boundary; and ϵ which determines the tolerance to errors in estimation. The ranges are as follows: $(2^{-2} - 2^5)$ for C; $(2^{-8} - 2^0)$ for σ; and $(2^{-8\text{˘}}2^{-1})$ for ϵ; these values are commonly used in SVM tuning (for example, Cortez and Embrechts, 2013; Khondoker, 2016; Hsu et al, 2003). For random forests, we do a random search sampling of 15 values for the parameter *mtry*, which determines the number of predictors to compare at each split of a tree. The neural network package nnet uses only one layer of hidden nodes. So we tune the number of nodes (*size*) at values 1, 3, 5, 10 and 20; we also tune the weight decay (*decay*) using values decreasing successively as follows: (5, 1, 0.5, 10-1 to 10-7).

References

Bergstra, J. and Bengio, Y. (2012) . Random search for hyper-parameter optimization. Journal of Machine Learning Research, 13, 281-305.

Cortez, P. & Embrechts, M. J. (2013). Using sensitivity analysis and visualization techniques to open black box data mining models. Information Sciences, 225, 1-17.

Hsu, C., Chang, C., and Lin, C. (2003). A practical guide to support vector classification. Retrieved from https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf.

Khondoker, M., Dobson, R., Skirrow, C., Simmons, A., & Stahl, D. (2016). A comparison of machine learning methods for classification using simulation with multiple real data examples from mental health studies. Statistical Methods in Medical Research, 25(5), 1804–1823. https://doi.org/10.1177/0962280213502437

#APPENDIX B

1. Caret_ML functions

load caret functions script and R libraries:

```
#####################################################################################
#--------------libraries----------------------------##
libs2use <- c("caret","caretEnsemble", "reshape2", "ggplot2"
              , "randomForest", "e1071",  "glmnet", "nnet", "earth")
libs2install<- libs2use[!libs2use %in% rownames(installed.packages()  ) ]
if (length(libs2install)) {  install.packages(libs2install, dependencies = TRUE) }
sapply(libs2use, require, character.only = TRUE)
```

```
## Loading required package: caret

## Loading required package: lattice

## Loading required package: ggplot2

## Loading required package: caretEnsemble

##
## Attaching package: 'caretEnsemble'

## The following object is masked from 'package:ggplot2':
##
##     autoplot

## Loading required package: reshape2

## Loading required package: randomForest

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin

## Loading required package: e1071

## Loading required package: glmnet
## Loading required package: Matrix

## Loading required package: foreach

## Loaded glmnet 2.0-18

## Loading required package: nnet

## Loading required package: earth

## Loading required package: Formula

## Loading required package: plotmo

## Loading required package: plotrix

## Loading required package: TeachingDemos

##          caret caretEnsemble      reshape2       ggplot2  randomForest
##           TRUE          TRUE          TRUE          TRUE          TRUE
##          e1071        glmnet          nnet         earth
##           TRUE          TRUE          TRUE          TRUE
```

```
##---------------------------------------------------------------------------
#########################################################################################
```

wrapper function for training all methods on CV folds and getting the result:

```
##--function to train models by giving a list of caret_ML functions defined below:

train_model.function <- function(formula, data.train, dv, CVfolds, functions.list){
  #create index of training folds for K-fold CV.
  indexPreds <- createFolds(data.train[, dv], k = CVfolds
                          , list = TRUE, returnTrain = FALSE) #for cv


  # #get all ML functions into a list for use in next line:
  # functions.list <- list( svm.function, rf.function, mars.function
  #                         , elastic.function,  lasso.function, lm.function )


  #train all models in a list
  results.list<-  lapply( functions.list,
                        ( function(f) f(formula, data.train, CVfolds, indexPreds)
                        )  )

  return (results.list)
}
#-------------------------------------------------------------
```

function takes in a caret result and gives CV accuracy measures in suitabl format:

```r
##---function to take a list of the custom caret model list and give the accuracy measures:--

caretmodel_accuracy.function2 <- function(multimodel) {
  #get the accuraacy estimates of all models....
  models.resamples <- resamples(multimodel)
  models.values <- models.resamples$values

  #RMSE
  models.rmse <- models.values[, grepl("RMSE", names(models.values))]
  #transform rmse to mse
  models.mse <- data.frame(models.rmse^2)
  colnames(models.mse) <- gsub( ".RMSE", "~MSE",   colnames(models.mse)   )
  #Rsquared
  models.Rsquared <- models.values[, grepl("Rsquared", names(models.values))]
  #MAE
  models.MAE <- models.values[, grepl("MAE", names(models.values))]


  #inside function to collate CV measures...
  plyr.function <- function( model, measure ){
    library(plyr)
    library(reshape2)
    model$fold <- 1:10
    model <- melt(model, variable.name = "method", value.name = "value", id.vars = "fold")
    model$method <- gsub( paste("~", measure, sep = ""), "", model$method   )
    model$measure <- measure
    return(model)
  }
  # plyr.function(models.Rsquared, "Rsquared")

  mae <- plyr.function(models.MAE, "MAE")
  rsq <- plyr.function(models.Rsquared, "Rsquared")
  mse <- plyr.function(models.mse, "MSE")

  out <- rbind(mae, rsq, mse)
  return (out)
}

##-----------------------------------------------------------------------
```

caret machine-learning functions inside wrapper functions for suitable formatting:

Neural Network:

```r
#-------------------------------------------------------------
#-------------------------------------------------------------
#------------------------------- Neural Net ----------------------------
```

```
nnet.function <- function(formula, mydata, kfold, indexPreds){

  cat("\n inside nnet.function\n")
  nnet.grid <- expand.grid(.decay = c(5, 1, 0.5, 0.1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7)
                           , .size = c(1,3, 5, 10, 20))

    nnet.control <- trainControl(
      method = "cv"
       , number = kfold#number of cv folds; set by kfold in function call; default = 10.
      # , repeats = repeats #repeats per fold; set by function call; default = 3
      , verboseIter = F
      # , returnData = T
      , savePredictions = T
    )

      nnet.model <- train(formula, data = mydata
        # , x = data1.mat[, -dv.index]
        # , y = data1.mat[, dv.index]
        , method = "nnet"
        , preProcess = c("center","scale")
        , metric = "RMSE"
        , trControl = nnet.control
        , tuneGrid = nnet.grid
        ,linout = TRUE
        ,verbose = FALSE
        ,trace = FALSE
      )


    nnet.tune <- nnet.model$bestTune
    return ( list( model = nnet.model, model.tune = nnet.tune,
                   train.control = nnet.control)  )

}
```

Support Vector Machines (SVM):

```
#-----------------------------------------------------------------------------------------------
##---------------------------------------------: svm model  :---------------------------------- --------
#---------caret's train does not tune epsilon, but only cost C and sigma...
#---------So we tune C and sigma in a loop over epislon values...
#----------and choose the trian model with best cv rmse.

svm.function <- function(formula, mydata, kfold, indexPreds){

  cat("\n inside svm.function\n")
  C= 2^seq(-2, 5)
  sig =  2^seq(-8,0, length.out = 6)
  ranges.svm <- list( C = C,  sigma = sig  )
```

```r
epsilon = 2^seq(-8, -1)

svm.rmse<- vector(length = length(epsilon))
svm.models <- list()  # to store the trained models
svm.param <- list()  #to store the best parameters fo each epsilon iteration

svm.ctrl <- trainControl(method="cv"   # K-fold cross validation
                         , number  = kfold
                         , search = "grid"
                         ,index = indexPreds #index of folds; same for all models
                         , savePredictions = TRUE
)
# svm.ctrl$returnResamp = "all"

for(e in 1: length(epsilon)){



  svm.train<- train( formula, data=mydata
                     # method = "svmRadial",   # Radial kernel
                     , method = "svmRadialSigma"
                     # tuneLength = 30,                   # 9 values of the cost function
                     , preProc = c("center","scale")  # Center and scale data
                     , metric="RMSE"
                     , trControl= svm.ctrl
                     , epsilon = epsilon[e]
                     , tuneGrid = expand.grid(ranges.svm)
  )
  svm.models[[e]] <- svm.train #storing the trained models
  svm.eps <- svm.train$finalModel@param$epsilon
  svm.sigma <- svm.train$bestTune$sigma
  svm.cost <- svm.train$bestTune$C
  # svm.model$bestTune

  svm.results <- svm.train$results
  svm.rmse[e] <- svm.results[svm.results$C ==svm.cost & svm.results$sigma ==svm.sigma , "RMSE"]

  svm.param[[e]] <- list(cost = svm.cost, sigma = svm.sigma, epsilon = svm.eps)

} #end of for loop in epsilon



svm.model <- svm.models[[  which(svm.rmse==min(svm.rmse) ) ]]

#the optimal hyper-parameters
svm.param.best <- unlist ( svm.param[ svm.rmse==min(svm.rmse)   ] )
svm.C <- svm.param.best["cost"]
svm.sigma <- svm.param.best["sigma"]
svm.epsilon <- svm.param.best["epsilon"]

ranges.svm <- list(C = svm.C, sigma = svm.sigma)
```

```
  return ( list( model = svm.model, ranges.svm = ranges.svm,
                 svm.epsilon = svm.epsilon,
                 train.control = svm.ctrl)  )

}

##-----------------------------------------------------
#######################################################################
```

Random forest:

```
##------ RF model:----------------------------

rf.function <- function(formula, mydata, kfold, indexPreds ){
  cat("\n inside rf.function\n")
  mtry.length = 15  #length of tunings  for mtry param to be attempted by random search.
  ntree = 500 #number of trees


  rf.control <- trainControl(method="cv", number=kfold, search="random", verboseIter = FALSE
                             ,index = indexPreds #index of folds; same for all models
                             , savePredictions = TRUE
                             # , repeats = reps
  )
  # rf.control$returnResamp = "all"

    rf.model<- train(formula , data = mydata
                     , method = "rf"
                     , ntree = ntree
                     , tuneLength = mtry.length
                     , metric = "RMSE"
                     , trControl = rf.control
                     , preProc = c("center","scale")  # Center and scale data
                     , importance = TRUE
                     )




  rf.best <- rf.model$finalModel
  rf.mtry <- rf.best$mtry

  ranges.rf <- list(mtry = rf.mtry)

  return( list( model = rf.model, ranges.rf = ranges.rf, train.control = rf.control  ) )

}
##-------------------------------------------------------------------
############################################################################
```

LASSO:

```
##-----------------------------------------------------------------

##:---------------LASSO:

lasso.function <- function(formula, mydata, kfold, indexPreds ){

  cat("\n inside lasso.function\n")

  lasso.ctrl <- trainControl(method="cv"   # 10fold cross validation
                             , number  = kfold
                             , search = "grid"
                             ,index = indexPreds #index of folds; same for all models
                             ,savePredictions = TRUE

  )

  #ideally we want to keeep alpha fixed at 1 (for lasso),...
  #and do random search on lambda. However, caret random search randomzies both params.
  #So as a run-around, we see that random search in caret searches the range 2^runif(-10, 3)
  #so we use this range in grid form to do 'random' search in search = grid, while keeping alpha = 1
  tunelength = 30
  ranges.lasso = list(alpha = 1, lambda = 2^runif(tunelength, min = -10, 3))
  #Train and Tune the Lasso with caret:
  #sometimes optimal lambda gives legit RMSE but no R-square;...
  #we repeat the training until we get legit R-square valeus for optimal lambda:
  # repeat({
    lasso.model <- train(formula, data=mydata,
                         method = "glmnet",
                         # tuneLength = tunelength,                # 9 values of the cost function
                         preProc = c("center","scale"),  # Center and scale data
                         metric="RMSE"
                         ,trControl=lasso.ctrl
                         ,tuneGrid = expand.grid(ranges.lasso)    )


    lasso.lambda.best <- lasso.model$bestTune$lambda
    cat("\n lasso lambda best before edit =  ", lasso.lambda.best, "\n" )
    lasso.results <- lasso.model$results


    cat("\n is R squared values NA at values of optimal lambda?  :  ")
    cat( is.na(lasso.results$Rsquared[lasso.results$lambda==lasso.model$bestTune$lambda ]) )
    cat("\n")

    if( is.na(lasso.results$Rsquared[lasso.results$lambda==lasso.model$bestTune$lambda ] )  ){


      lasso.order <- order(lasso.results$RMSE)
      lasso.results.ordered <- lasso.results[lasso.order,]
      lasso.results.edit <- lasso.results.ordered[  !is.na(lasso.results.ordered$Rsquared),  ]
      cat("\n")
```

```r
      print(lasso.results.edit)
      cat("\n")
      lasso.lambda.best <- lasso.results.edit$lambda[1]
      cat("\n lasso lambda best after edit =  ", lasso.lambda.best, "\n" )

      ranges.lasso <- list(alpha = 1, lambda = lasso.lambda.best)
      lasso.model <- train(formula, data=mydata,
                          method = "glmnet",
                          # tuneLength = tunelength,                   # 9 values of the cost function
                          preProc = c("center","scale"),  # Center and scale data
                          metric="RMSE"
                          ,trControl=lasso.ctrl
                          ,tuneGrid = expand.grid(ranges.lasso)    )

      cat("\n lasso lambda best after second round model =  ", lasso.lambda.best, "\n" )

      lasso.results <- lasso.model$results
      cat("\n")
      print(lasso.results)
      cat("\n")

    }


  ranges.lasso <- list(alpha = 1, lambda = lasso.lambda.best)

  return( list( model = lasso.model, ranges.lasso = ranges.lasso,
               train.control = lasso.ctrl  ) )

}


##----------------------------------------------------------------------##
##########################################################################
```

Elastic Net:

```r
##-------------- Elastic Net -------------------------------------------##

elastic.function <- function(formula, mydata, kfold, indexPreds){
  cat("\n inside elastic.function\n")

  elastic.ctrl <- trainControl(method="cv"   # Kfold cross validation
                              , number  = kfold
                              ,savePredictions = TRUE
                              , search = "random"
                              ,index = indexPreds #index of folds; same for all models
  )

  #Train and Tune the Elastic with caret
  tunelength = 30
  # repeat({ #we repeat the training until we get legit R-square values for optimal lambda:

    elastic.model <- train(formula, data=mydata,
                           # method = "svmRadial",   # Radial kernel
                           method = "glmnet",
                           tuneLength = tunelength,                   # 9 values of the cost function
                           preProc = c("center","scale"),  # Center and scale data
                           metric="RMSE"
                           ,trControl= elastic.ctrl
                           # ,tuneGrid = expand.grid(ranges.elastic)
    )

    elastic.results <- elastic.model$results
    el.lambda.best <- elastic.model$bestTune$lambda
    el.alpha.best <- elastic.model$bestTune$alpha
    cat("\n elastic lambda best before edit =  ", el.lambda.best, "\n" )



    cat("\n is R sqaured values NA at values of optimal lambda?  :  ")
    cat( is.na(elastic.results$Rsquared[elastic.results$lambda==elastic.model$bestTune$lambda ]) )
    cat("\n")

    if( is.na(elastic.results$Rsquared[elastic.results$lambda==elastic.model$bestTune$lambda ] ) ){


      elastic.order <- order(elastic.results$RMSE)
      elastic.results.ordered <- elastic.results[elastic.order,]
      elastic.results.edit <- elastic.results.ordered[  !is.na(elastic.results.ordered$Rsquared),  ]
      cat("\n")
      print(elastic.results.edit)
      cat("\n")
      el.lambda.best <- elastic.results.edit$lambda[1]
      el.alpha.best <- elastic.results.edit$alpha[1]
      cat("\n elastic lambda best after edit =  ", el.lambda.best, "\n" )
      ranges.elastic <- list(alpha = el.alpha.best, lambda = el.lambda.best)

      #now redo the model with the new lambda and alpha
      elastic.ctrl$search = "grid"
      elastic.model <- train(formula, data=mydata,
                             # method = "svmRadial",   # Radial kernel
                             method = "glmnet",
                             tuneLength = tunelength,                 # 9 values of the cost function
                             preProc = c("center","scale"),  # Center and scale data
                             metric="RMSE"
                             ,trControl= elastic.ctrl
                             ,tuneGrid = expand.grid(ranges.elastic)
      )

      elastic.results <- elastic.model$results
      el.lambda.best <- elastic.model$bestTune$lambda
      el.alpha.best <- elastic.model$bestTune$alpha
      cat("\n elastic lambda best after second round model =  ", el.lambda.best, "\n" )
```

```
        cat("\n")
        print(elastic.results)
        cat("\n")

      }




    # })

  ranges.elastic <- list(alpha = el.alpha.best, lambda = el.lambda.best)

  return( list( model = elastic.model, ranges.elastic = ranges.elastic,
                train.control = elastic.ctrl) )

}

##----------------------------------------------------------------------##
#############################################################################
```

Multivariate adaptive Regression Splines (MARS):

```
##----------------------------------------------------------------------##

##-------------------------------------MARS:------------------------------------

mars.function <- function(formula, mydata, kfold, indexPreds){
  cat("\n inside mars.function\n")

  mars.control <- trainControl( method = 'cv'
                                , number = kfold
                                , search = "random"
                                , verboseIter = FALSE
                                , savePredictions = TRUE
                                , allowParallel = TRUE

  )

  system.time({
    mars.model <- train(formula, data=mydata, method = "earth"
                        , metric="RMSE", trControl = mars.control, tuneLength= 30)

  })

  mars.nprune.best <- mars.model$bestTune$nprune
  mars.degree.best <- mars.model$bestTune$degree

  ranges.mars = list(nprune = mars.nprune.best, degree = mars.degree.best)

  return( list( model = mars.model, ranges.mars = ranges.mars,
                train.control = mars.control) )
}
##------------------------------------------------------------------------##
###############################################################################
```

regression:

```
##---------------------------- linear model lm:-----------------------------------

lm.function <- function(formula, mydata, kfold, indexPreds){

  cat("\n inside lm.function\n")

  #tuning lm with 10-fold cv:
  lm.control <- trainControl(method="cv", number=kfold, search="random"
                             , index = indexPreds
                             ,savePredictions = TRUE
                             ,verboseIter = FALSE)
  lm.model <- train(formula, data = mydata, trControl = lm.control, method = "lm")


  return(  list( model = lm.model, ranges.lm = NA, train.control = lm.control )  )

}


##---------------------------------------------------------##
############################################################################
```

2. Functions to estimate SE:

```
#########################################################################################
#--------------libraries---------------------------##
libs2use <- c("tidyr", "reshape2", "plyr", "dplyr","ggplot2", "meta")
libs2install<- libs2use[!libs2use %in% rownames(installed.packages()  ) ]
if (length(libs2install)) {  install.packages(libs2install, dependencies = TRUE) }
sapply(libs2use, require, character.only = TRUE)
```

```
## Loading required package: tidyr

##
## Attaching package: 'tidyr'

## The following object is masked from 'package:Matrix':
##
##     expand

## The following object is masked from 'package:reshape2':
##
##     smiths

## Loading required package: plyr

## Loading required package: dplyr

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:plyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize

## The following object is masked from 'package:randomForest':
##
##     combine

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

## Loading required package: meta

## Loading 'meta' package (version 4.9-6).
## Type 'help(meta)' for a brief overview.
```

```
##------------------------------------------------------------------------
########################################################################################
```

Meta-analysis:

```
#function to estimate meta-analytic results as dataframe for one study/many methods from ind folds:-
meta.function2 <- function( df1,
                           TE.name = "TE",
                           se.name = "seTE",
                           study.name = "study",
                           method.name = "method"){
  library(meta)
  # TE.name <- "value"
  # se.name <- "se"
  # study.name <- "study"
  # method.name <- "method"
  #
  # df1 <- cv.rsquare[cv.rsquare$study ==1,]
  #

  TE <- df1[, TE.name]
  se <- df1[, se.name]
  study <- df1[, study.name]
  method <- df1[, method.name]
  df1 <- data.frame(TE, se, study, method)

  meta.result <- metagen( TE = TE, seTE = se, data = df1
                         , studlab=study)
  meta.result  <- update(meta.result, byvar = method, bylab = "method") #by study analysis
  summary(meta.result)

  #we will construct a dataframe fit for ggplot:
  #first we get a dataframe with effects of individual studies and summaries:
  meta.dat <- as.data.frame(meta.result)

  #tidy up the column names etc...
  names(meta.dat)[1] <- study.name
  names(meta.dat) [names(meta.dat)%in% c("byvar")] <- method.name
  # names(meta.dat) [names(meta.dat)%in% c("byvar")] <- method.name
  # meta.dat <- meta.dat[ ,c("fold" , "TE","seTE", "lower", "upper", "method" )]
  meta.dat <- meta.dat[ ,c(study.name, method.name
                          , "TE", "seTE"
                          , "lower", "upper")  ]

  head(meta.dat)

  #now for 2nd dataframe: get fixed and random summary into a dataframe
  meta.sum.dat <- data.frame(
    study = rep( c("fixed", "random"), each = length(meta.result$bylevs)  )
    , TE = c( meta.result$TE.fixed.w, meta.result$TE.random.w)
    # , TE.random = meta.result$TE.random.w
    , seTE = c( meta.result$seTE.fixed.w, meta.result$seTE.random.w)
    # , seTE.random = seTE.random.w
    , lower = c( meta.result$lower.fixed.w, meta.result$lower.random.w)
    , upper = c( meta.result$upper.fixed.w, meta.result$upper.random.w)
    , method = rep(meta.result$bylevs, 2)
  )

  names(meta.sum.dat)[names(meta.sum.dat) %in% "study"] <- study.name
  names(meta.sum.dat)[names(meta.sum.dat) %in% "method"]  <- method.name

  #merge both to get plotting dataframe...
  meta.dat <- rbind(meta.dat, meta.sum.dat)
  meta.dat$type <- "individual"
  meta.dat$type[meta.dat[,study.name] =="fixed" | meta.dat[,study.name] =="random" ] <- "summary"
  # meta.dat

  return (meta.dat)

} # end of meta.function
##-------------------------------------------------------------------##
```

Olkin and Finn's equation 1 for SE of a sample R^2:

```
##--olkin and finn's SE----------------------------------------------------------------##
olkin.se <- function(rsquare, N){
  se = sqrt((4/N)*rsquare *(1 - rsquare)^2)

  return (data.frame(Rsquare = rsquare, se = se)  ) }
##------------------------------------------------------------------------------------##
```

Olkin and Finn's equation 2 for SE of R^2 difference

```
##--function to calculate se for Rsquare differences---------------------
Rsquare.diff.se <- function(rsquare.1, rsquare.2, N_1,N_2 ){


  d = rsquare.1 - rsquare.2

  #get 95% Prediction interval of difference between Rsq (Olkin&Finn, 1995):
  se_tot <- sqrt(  ( (4/N_1)*rsquare.1 *(1 - rsquare.1)^2 ) +
                     ( (4/N_2)*rsquare.2 *(1 - rsquare.2)^2 )
  )
  return  (data.frame( d = d, se = se_tot) )

}

##--------------------------------------------------------------
```

3. Replication procedures illustration for CV R^2

CV single replication

data input:

```
##-------single replications; difference b/w CV r squares:


#take in CV data:
cv.dat <- read.csv("cv.dat.csv")


#get the subset containing CV Rsquare:
cv.rsquare <- cv.dat[cv.dat$measure == "Rsquared",]
head(cv.rsquare)
```

```
##    fold method     value  measure study study.type train.N
## 71    1   nnet 0.4905351 Rsquared     0     Target    1333
## 72    2   nnet 0.4302690 Rsquared     0     Target    1333
## 73    3   nnet 0.3550680 Rsquared     0     Target    1333
## 74    4   nnet 0.3916048 Rsquared     0     Target    1333
## 75    5   nnet 0.6163268 Rsquared     0     Target    1333
## 76    6   nnet 0.4540757 Rsquared     0     Target    1333
```

estimate fold-level SE using olkin and Finn's equation 1:

```
cv.rsquare <-   ddply( cv.rsquare, c("study", "method", "fold"), function(x) {
  df1 <- olkin.se( x$value, x$train.N )
  df1$study.type <- x$study.type
  df1$train.N <- x$train.N
  df1
} )

head(cv.rsquare)
```

```
##   study  method fold   Rsquare         se study.type train.N
## 1     0 elastic    1 0.4753780 0.01981440     Target    1333
## 2     0 elastic    2 0.4692877 0.01991561     Target    1333
## 3     0 elastic    3 0.4490304 0.02022462     Target    1333
## 4     0 elastic    4 0.4755788 0.01981100     Target    1333
## 5     0 elastic    5 0.4796078 0.01974189     Target    1333
## 6     0 elastic    6 0.4470715 0.02025221     Target    1333
```

estimate study level SE using meta-analytic methods:

```r
library(plyr)
meta.dat <- ddply( cv.rsquare, "study", function(subdat){
  df1 <- meta.function2( subdat,
                          TE.name = "Rsquare",
                          se.name = "se",
                          study.name = "fold",
                          method.name = "method")
  df1
})
head(meta.dat)
```

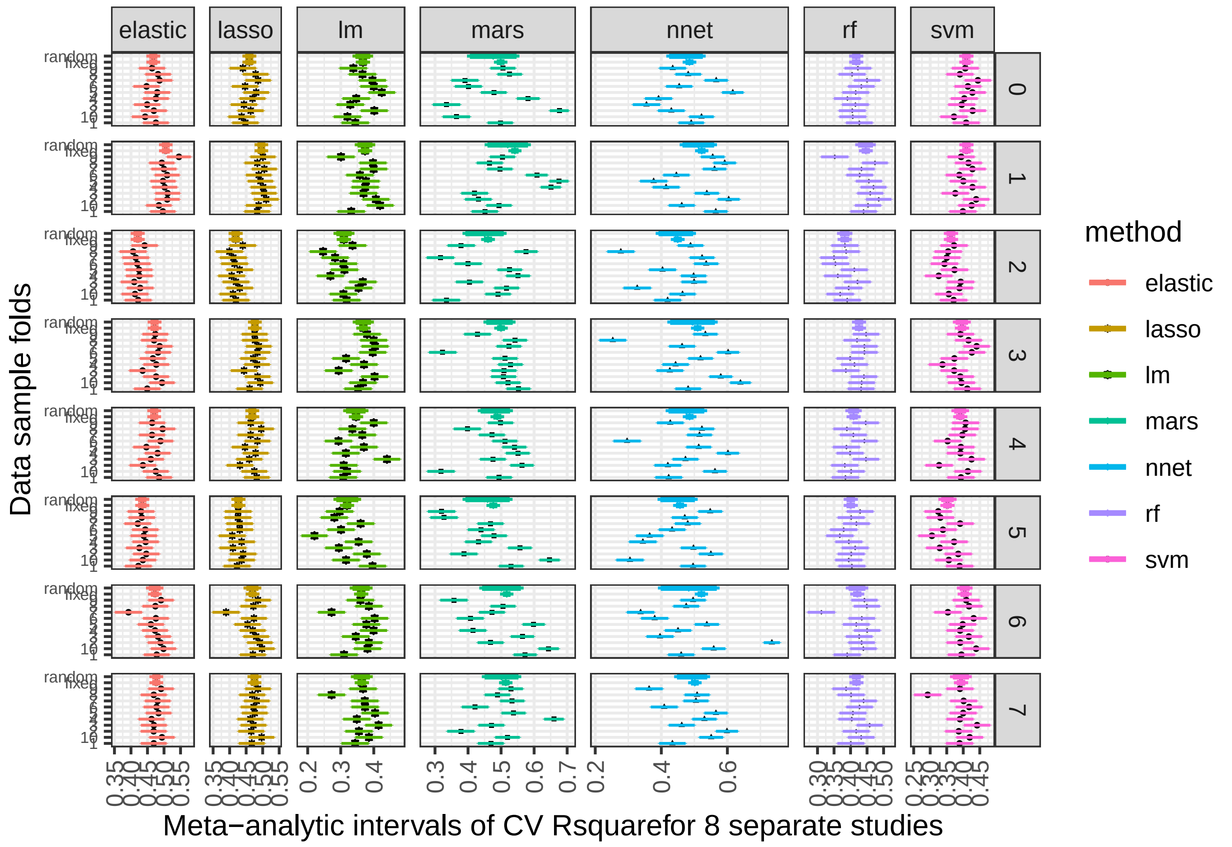
```
##   study fold  method        TE       seTE     lower     upper       type
## 1     0    1 elastic 0.4753780 0.01981440 0.4365425 0.5142135 individual
## 2     0    2 elastic 0.4692877 0.01991561 0.4302538 0.5083216 individual
## 3     0    3 elastic 0.4490304 0.02022462 0.4093909 0.4886699 individual
## 4     0    4 elastic 0.4755788 0.01981100 0.4367499 0.5144076 individual
## 5     0    5 elastic 0.4796078 0.01974189 0.4409144 0.5183012 individual
## 6     0    6 elastic 0.4470715 0.02025221 0.4073779 0.4867651 individual
```

```r
tail(meta.dat)
```

```
##     study   fold method        TE        seTE     lower     upper    type
## 667     7 random  lasso 0.4750491 0.006266239 0.4627675 0.4873307 summary
## 668     7 random     lm 0.3637711 0.012524340 0.3392239 0.3883184 summary
## 669     7 random   mars 0.5014132 0.026039707 0.4503763 0.5524501 summary
## 670     7 random   nnet 0.4932158 0.023348473 0.4474536 0.5389779 summary
## 671     7 random     rf 0.4177676 0.006881117 0.4042808 0.4312543 summary
## 672     7 random    svm 0.3928862 0.012449960 0.3684847 0.4172877 summary
```

plot:

```r
library(ggplot2)
measure = "Rsquare"
ggplot(meta.dat, aes(y=fold, x=TE, xmin=lower, xmax=upper, shape = method, color = method))+
  #Add data points and color them black
  geom_point(color = 'black', size = 0.5)+
  #Add 'special' points for the summary estimates, by making them diamond shaped
  geom_point(data=subset(meta.dat, type=='summary'),
             aes(color = method), shape = 18, size= 1)+
  #add the CI error bars
  geom_errorbarh(height=.1)+
  geom_errorbarh( data = subset(meta.dat, type=='summary'),  height=.5, size = 1)+
  # geom_errorbar(data = subset(meta.dat, type=='summary'), aes(ymin=lower, ymax= upper, width=.5, size
  #manually  specify shapes
  scale_shape_manual(values=c(1, 8, 11, 15, 17, 18, 19)) +
  #Specify the limits of the x-axis and relabel it to something more meaningful
  # scale_x_continuous(limits=c(-2,2), name='Standardized Mean Difference (d)')+
  #Give y-axis a meaningful label
  ylab('Data sample folds')+
  xlab(paste("Meta-analytic intervals of CV ", measure, "for 8 separate studies"  , sep = "") ) +
  #Add a vertical dashed line indicating an effect size of zero, for reference
  # geom_vline(xintercept=0, color='black', linetype='dashed')+
  #Create sub-plots (i.e., facets) based on levels of setting
  #And allow them to have their own unique axes (so authors don't redundantly repeat)
  facet_grid(study ~ method, scales= 'free', space='free')+
  theme_bw() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1)) +
  theme(axis.text.y = element_text(size = 6))
```

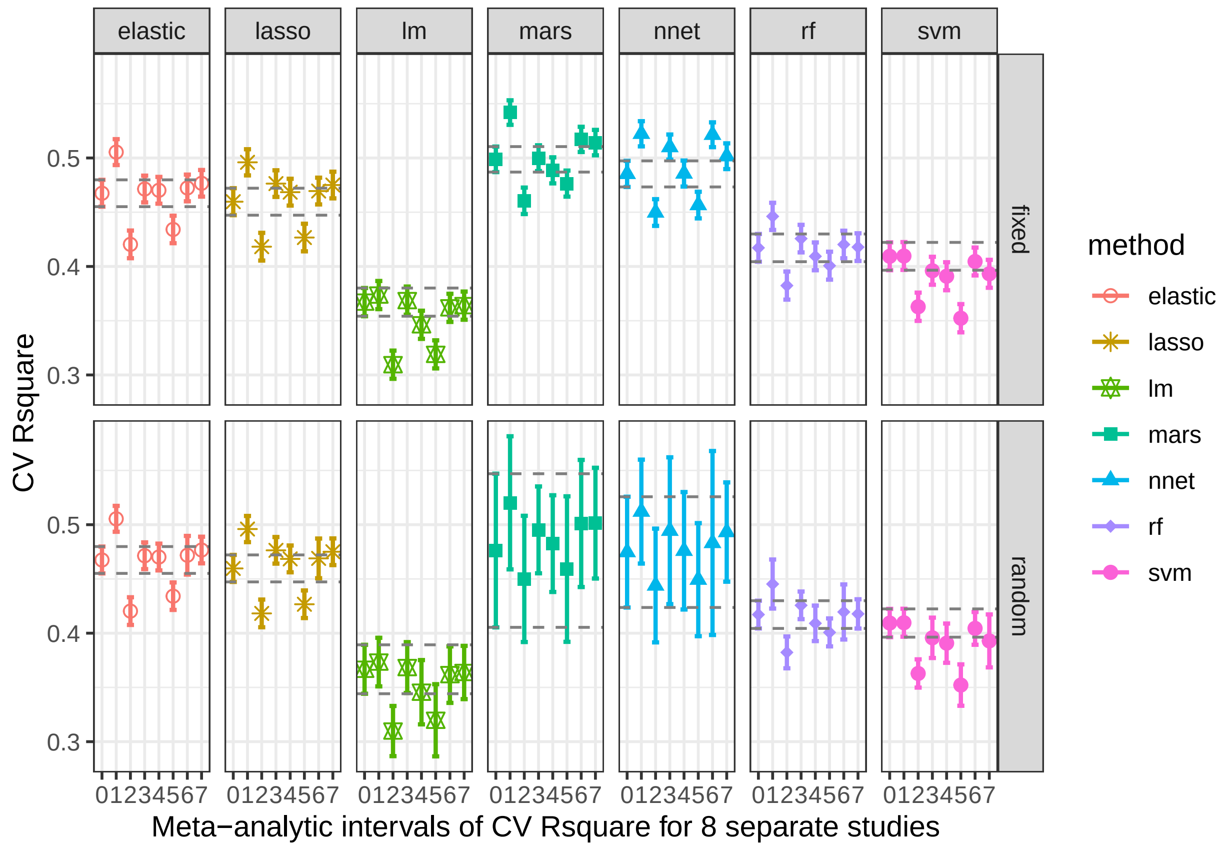


```
meta.summary <- meta.dat[meta.dat$type =="summary",]
head(meta.summary)
```

```
##    study  fold  method        TE        seTE     lower     upper    type
## 71     0 fixed elastic 0.4674605 0.006304062 0.4551047 0.4798162 summary
## 72     0 fixed   lasso 0.4596935 0.006341276 0.4472649 0.4721222 summary
## 73     0 fixed      lm 0.3671208 0.006618634 0.3541485 0.3800931 summary
## 74     0 fixed    mars 0.4987242 0.005994030 0.4869761 0.5104723 summary
## 75     0 fixed    nnet 0.4852968 0.006133008 0.4732763 0.4973172 summary
## 76     0 fixed      rf 0.4171570 0.006516648 0.4043847 0.4299294 summary
```

```
#Make a plot, and map citation data to y-axis, effect sizes to x-axis
#specify the min and max of the CIs, and give different shapes based on levels of tester
library(ggplot2)
measure = "Rsquare"
```

```r
meta.summary$study <- factor(meta.summary$study)
# meta.summary$fold <- factor(meta.summary$fold, levels = c( 1:10, "fixed", "random"))

ggplot(meta.summary, aes(y=TE, x=study, ymin=lower, ymax=upper, shape = method, color = method))+
  #Add data points and color them black
  # geom_point(color = 'black', size = 0.5)+
  #Add 'special' points for the summary estimates, by making them diamond shaped
  geom_point(size= 2)+
  #add the CI error bars
  # geom_errorbar(height=.1)+
  # geom_errorbarh( data = subset(meta.summary, type=='summary'),  height =.15, size = 0.75)+
  geom_errorbar( data = subset(meta.summary, type=='summary'),
                 width =.5, size = 0.75)+
  geom_hline(data = subset(meta.summary, study == 0),
             aes(yintercept=lower), color='grey50', linetype='dashed', size = 0.5)+
  geom_hline(data = subset(meta.summary, study == 0),
             aes(yintercept=upper), color='grey50', linetype='dashed', size = 0.5)+
  # geom_errorbar(data = subset(meta.dat, type=='summary'), aes(ymin=lower, ymax= upper, width=.5, size
  #manually  specify shapes
  scale_shape_manual(values=c(1, 8, 11, 15, 17, 18, 19)) +
  #Specify the limits of the x-axis and relabel it to something more meaningful
  # scale_x_continuous(limits=c(-2,2), name='Standardized Mean Difference (d)')+
  #Give y-axis a meaningful label
  ylab('CV Rsquare')+
  xlab(paste("Meta-analytic intervals of CV ", measure, " for 8 separate studies"  , sep = "") ) +
  #Add a vertical dashed line indicating an effect size of zero, for reference
  # geom_vline(xintercept=0, color='black', linetype='dashed')+
  #Create sub-plots (i.e., facets) based on levels of setting
  #And allow them to have their own unique axes (so authors don't redundantly repeat)
  facet_grid(fold ~ method) + #, scales= 'free', space='free')+
  theme_bw()
```



```
##-------------------
```

getting CV R^2 difference estimate & SE:

We give example of random-effects; same procedure is used for fixed-effects.

```
# get the random-effects study-level SE:
meta.random <- meta.summary[meta.summary$fold =="random",]
meta.random  <- subset(meta.random , select = c(TE, seTE, study, method) ) #select mean, se and study c
meta.sub <- meta.random


#get the difference estimate between studies 1-7 and study 0:
library(dplyr)
cv.mean <- ddply( meta.sub , "method", function(subdata){
  subdata  %>%
    # mutate_if(is.numeric, funs(.-first(.)))
  mutate_if(is.numeric, list(~.-first(.)))
})

cv.mean <- subset(cv.mean, select = c(TE, study, method))


#get se of diff = sqrt( sum of variance )
cv.var <- subset(meta.sub , select = c(seTE, method, study) )
```

```r
cv.var$var <- cv.var[, 1]^2
cv.var <- cv.var[,-1]
cv.var <- ddply( cv.var, "method", function(subdata){
  subdata  %>%
    # mutate_if(is.numeric, funs(.+first(.)))
  mutate_if(is.numeric, list(~. +first(.)))
})


#merge to get dataframe with both difference R^2 and its SE
cv.diff <- merge(cv.mean, cv.var)
cv.diff$se <- sqrt(cv.var$var)

#remove redundant row of study 0:
cv.diff <- subset(cv.diff, !study == 0)
cv.diff$study <- factor(cv.diff$study)

head(cv.diff)
```

```
##    study  method          TE          var          se
## 8      1 elastic 0.037938778 7.680596e-05 0.008882554
## 9      1   lasso 0.036291664 7.800674e-05 0.008967918
## 10     1      lm 0.006622852 2.627152e-04 0.008832142
## 11     1    mars 0.043986218 2.284239e-03 0.009091622
## 12     1    nnet 0.037348550 1.274925e-03 0.008909138
## 13     1      rf 0.028143297 1.754270e-04 0.008936927
```

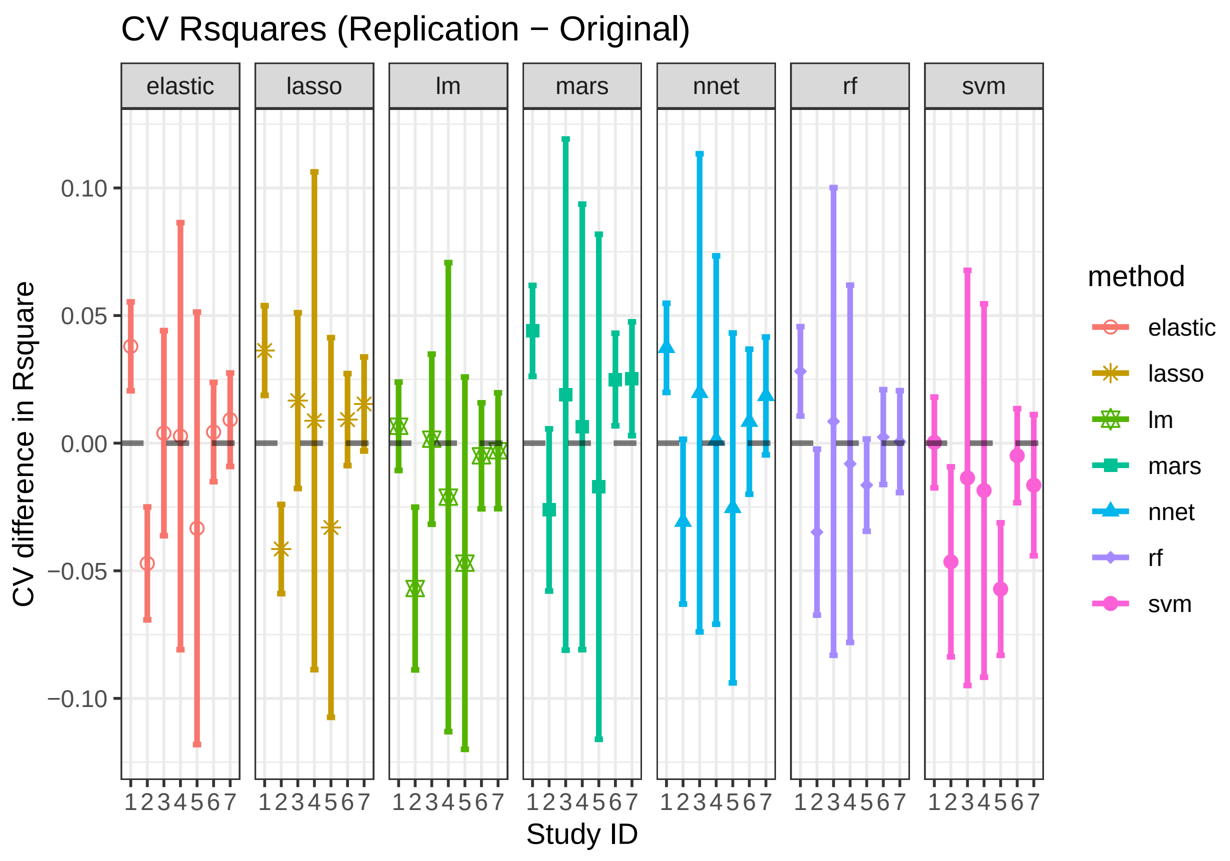
```r
str(cv.diff)
```

```
## 'data.frame':    49 obs. of  5 variables:
##  $ study : Factor w/ 7 levels "1","2","3","4",..: 1 1 1 1 1 1 1 2 2 2 ...
##  $ method: Factor w/ 7 levels "elastic","lasso",..: 1 2 3 4 5 6 7 1 2 3 ...
##  $ TE    : num  0.03794 0.03629 0.00662 0.04399 0.03735 ...
##  $ var   : num  7.68e-05 7.80e-05 2.63e-04 2.28e-03 1.27e-03 ...
##  $ se    : num  0.00888 0.00897 0.00883 0.00909 0.00891 ...
```

Procedures for assessing replicability in CV R^2

1) Test of inconsistency:

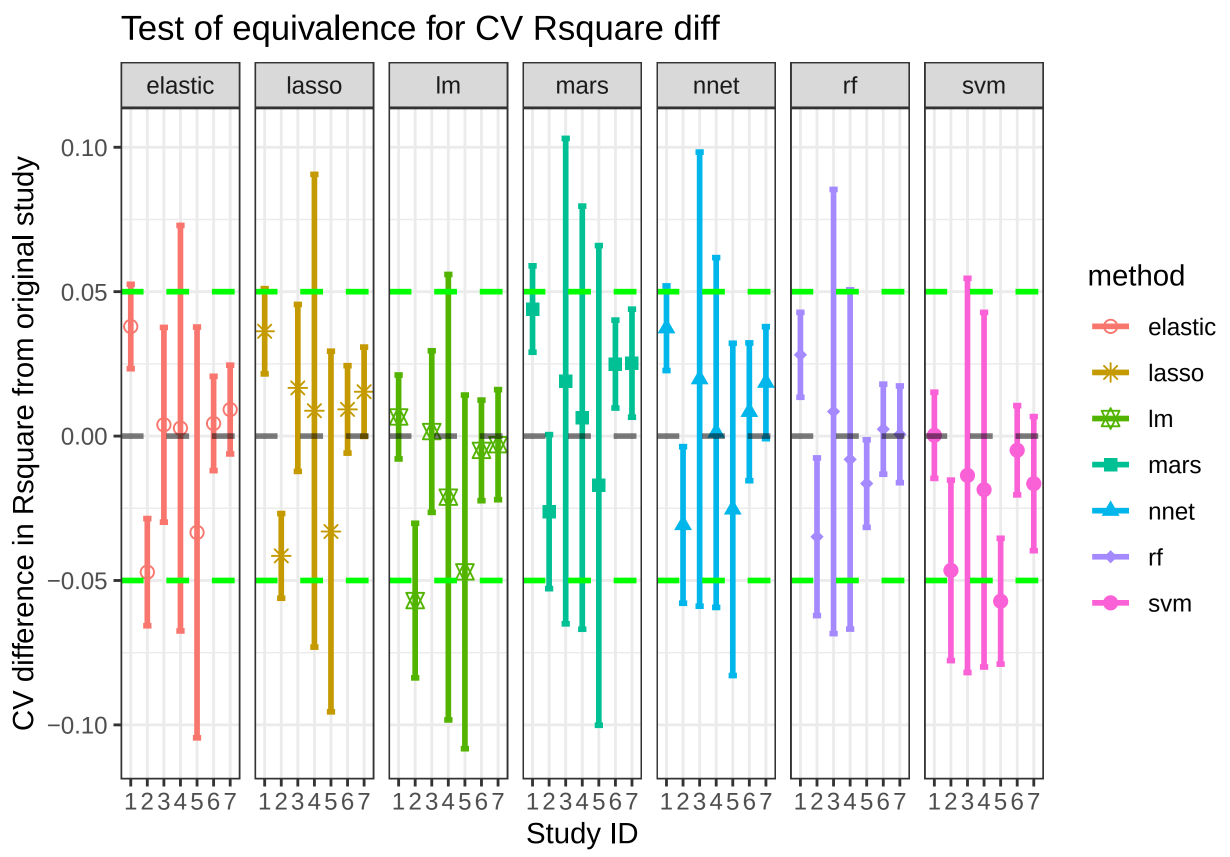
```r
p.alpha = 0.975
library(ggplot2)

ggplot( cv.diff, aes(study, TE, group = method
                     , color = method, shape = method ) ) +
  facet_grid(~ method) +
  geom_point(size = 2) + # geom_line(size = 1) +
  geom_errorbar(aes(ymin=TE - qnorm(p.alpha) * se, ymax=TE + qnorm(p.alpha)*se), width=.5, size = 1 ) +
  geom_hline(yintercept = 0, color = "black", linetype="dashed", size = 1, alpha = 0.5) +
  scale_shape_manual(values=c(1, 8, 11, 15, 17, 18, 19)) +
  ylab (paste("CV difference in", measure, sep = " ")) + xlab("Study ID") +
  theme_bw()+
  # stat_summary(fun.y = mean, geom = "line") +
  # stat_summary(fun.y = mean, geom = "point", size = 2) +
  # # scale_x_discrete(
  # #   breaks = levels( ) ) +
  # stat_summary(fun.data = mean_se, geom = "errorbar", width = 0.3) +
  # theme_bw() +
  # # theme(legend.justification = c(1, 0), legend.position = c(0.95, 0.05)) +
  ggtitle(paste("CV Rsquares (Replication - Original)")  )
```



```
# ggtitle(paste("Test of equivalence for CV Rsquare diff _dv: ", dv)  )
```

2) Test of Consistency:

```
p.alpha = 0.95
tost.value = 0.05
library(ggplot2)

ggplot( cv.diff, aes(study, TE, group = method
                     , color = method, shape = method ) ) +
  facet_grid(~ method) +
  geom_point(size = 2) + # geom_line(size = 1) +
  geom_errorbar(aes(ymin=TE - qnorm(p.alpha) * se, ymax=TE + qnorm(p.alpha)*se), width=.5, size = 1 ) +
  geom_hline(yintercept = 0, color = "black", linetype="dashed", size = 1, alpha = 0.5) +
  geom_hline(yintercept = +tost.value, color = "green", linetype="dashed", size = 1, alpha = 1) +
  geom_hline(yintercept = -tost.value, color = "green", linetype="dashed", size = 1, alpha = 1) +
  scale_shape_manual(values=c(1, 8, 11, 15, 17, 18, 19)) +
  ylab (paste("CV difference in", measure,"from original study", sep = " ")) + xlab("Study ID") +
  theme_bw()+
  # stat_summary(fun.y = mean, geom = "line") +
  # stat_summary(fun.y = mean, geom = "point", size = 2) +
  # # scale_x_discrete(
  # #   breaks = levels( ) ) +
  # stat_summary(fun.data = mean_se, geom = "errorbar", width = 0.3) +
  # theme_bw() +
  # # theme(legend.justification = c(1, 0), legend.position = c(0.95, 0.05)) +
  # ggtitle(paste("CV Rsquares (Replication - Original)  _dv: ", dv)  )
  ggtitle(paste("Test of equivalence for CV Rsquare diff ")  )
```



3) Estimating population R^2 by combining studies:

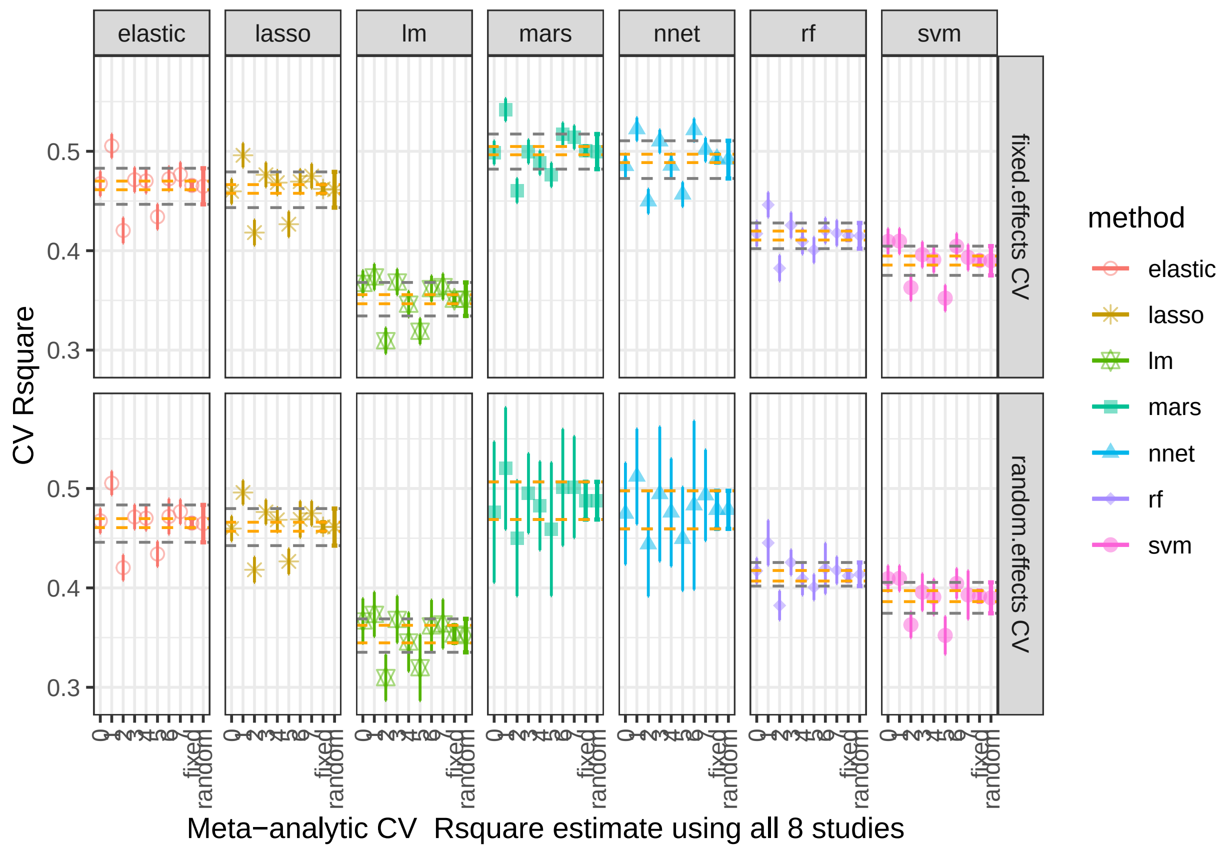
```
library(meta)

meta.fixed <- meta.summary[meta.summary$fold =="fixed",]
meta.random <- meta.summary[meta.summary$fold =="random",]
```

```r
#get the fixed and random-effects intervals from studies
#with SE levels earlier formed by fixed, or random effects meta-analysis
metaresult.fixed <- meta.function2(meta.fixed)
metaresult.random <- meta.function2(meta.random)
metaresult.fixed$CI.type <- "fixed.effects CV"
metaresult.random$CI.type <- "random.effects CV"
meta.summary <- data.frame(rbind(metaresult.fixed,
                                 metaresult.random) )
head(meta.summary)
```

```
##   study  method        TE        seTE     lower     upper       type
## 1     0 elastic 0.4674605 0.006304062 0.4551047 0.4798162 individual
## 2     0   lasso 0.4596935 0.006341276 0.4472649 0.4721222 individual
## 3     0      lm 0.3671208 0.006618634 0.3541485 0.3800931 individual
## 4     0    mars 0.4987242 0.005994030 0.4869761 0.5104723 individual
## 5     0    nnet 0.4852968 0.006133008 0.4732763 0.4973172 individual
## 6     0      rf 0.4171570 0.006516648 0.4043847 0.4299294 individual
##             CI.type
## 1 fixed.effects CV
## 2 fixed.effects CV
## 3 fixed.effects CV
## 4 fixed.effects CV
## 5 fixed.effects CV
## 6 fixed.effects CV
```

plot:

```r
##plotting:-----
library(ggplot2)
measure = "Rsquare"
meta.summary$study <- factor(meta.summary$study)

ggplot(meta.summary, aes(y=TE, x=study, ymin=lower, ymax=upper, shape = method, color = method))+
  #Add data points and color them black
  # geom_point(color = 'black', size = 0.5)+
  #Add 'special' points for the summary estimates, by making them diamond shaped
  geom_point(size= 2, alpha = 0.5)+
  #add the CI error bars
  geom_errorbar(width =.1)+
  # geom_errorbarh( data = subset(meta.summary, type=='summary'),  height =.15, size = 0.75)+
  geom_errorbar( data = subset(meta.summary, type=='summary'),
                 width =.5, size = 0.75)+
  geom_hline(data = subset(meta.summary, study == "random"),
             aes(yintercept=lower), color='grey50',
             linetype='dashed', size = 0.5)+
  geom_hline(data = subset(meta.summary, study == "random"),
             aes(yintercept=upper), color='grey50',
             linetype='dashed', size = 0.5)+
  geom_hline(data = subset(meta.summary, study == "fixed"),
             aes(yintercept=lower), color='orange',
             linetype='dashed', size = 0.5)+
  geom_hline(data = subset(meta.summary, study == "fixed"),
             aes(yintercept=upper), color='orange',
             linetype='dashed', size = 0.5)+

  # geom_errorbar(data = subset(meta.dat, type=='summary'), aes(ymin=lower, ymax= upper, width=.5, size
  #manually  specify shapes
  scale_shape_manual(values=c(1, 8, 11, 15, 17, 18, 19)) +
  #Specify the limits of the x-axis and relabel it to something more meaningful
  # scale_x_continuous(limits=c(-2,2), name='Standardized Mean Difference (d)')+
  #Give y-axis a meaningful label
  ylab('CV Rsquare')+
  xlab(paste("Meta-analytic CV  ", measure, " estimate using all 8 studies"  , sep = "") ) +
  #Add a vertical dashed line indicating an effect size of zero, for reference
  # geom_vline(xintercept=0, color='black', linetype='dashed')+
  #Create sub-plots (i.e., facets) based on levels of setting
  #And allow them to have their own unique axes (so authors don't redundantly repeat)
  facet_grid(CI.type ~ method) +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))
```



4. Replication procedures illustration for Test R^2

Test single replication

data input:

```
test.dat <- read.csv("test.dat.csv")
head(test.dat)
```

```
##       nnet      svm       rf     mars  elastic    lasso       lm   idno
## 1 8.334476 8.988936 8.429667 8.531130 8.407599 8.551015 8.764905 103514
## 2 7.408751 7.822744 7.140833 7.861859 7.838147 7.851571 7.414139 106520
## 3 8.272364 7.755536 7.904367 7.861859 7.837324 7.816556 7.801267 107125
## 4 8.316337 8.383162 7.988900 7.861859 7.909128 8.062112 8.062755 107220
## 5 5.185649 5.209050 4.768133 6.045954 6.268462 6.013210 5.656322 108525
## 6 5.245489 6.275323 6.135567 6.045954 6.359892 6.289378 6.350032 109528
##   y.obs study test.N
## 1     9     0    667
## 2     8     0    667
## 3     8     0    667
## 4     7     0    667
## 5     3     0    667
## 6     7     0    667
```

```
study <- 0:7
target.study <- 0
dv <- "happy"

#into long format:
test.dat <- melt( test.dat, id.vars = c("idno", "y.obs", "study", "test.N"),
                  value.name = "y.pred", variable.name = "method")
head(test.dat)
```

```
##     idno y.obs study test.N method   y.pred
## 1 103514     9     0    667   nnet 8.334476
## 2 106520     8     0    667   nnet 7.408751
## 3 107125     8     0    667   nnet 8.272364
## 4 107220     7     0    667   nnet 8.316337
## 5 108525     3     0    667   nnet 5.185649
## 6 109528     7     0    667   nnet 5.245489
```

creating test R^2 dataframe:

```
library(plyr)
Rsq.dat <- ddply(test.dat, c("study", "method"), function(x) {
  Rsquare <- cor(x$y.obs, x$y.pred)^2
  test.N <- unique(x$test.N)
  cbind(Rsquare, test.N)
} )

names(Rsq.dat)[3] <- "Rsquare"
Rsq.dat$study.type <- ifelse(Rsq.dat$study == target.study
                             , "Target", "Replication")
#a slight complication cos of zero value; hence add 1 to study and put that value into train.N

head(Rsq.dat)
```

```
##   study  method   Rsquare test.N study.type
## 1     0    nnet 0.4335675    667     Target
## 2     0     svm 0.4804074    667     Target
## 3     0      rf 0.4723777    667     Target
## 4     0    mars 0.4661091    667     Target
## 5     0 elastic 0.4801965    667     Target
## 6     0   lasso 0.4836806    667     Target
```

```
tail(Rsq.dat )
```

```
##    study  method   Rsquare test.N  study.type
## 51     7     svm 0.4323126    667 Replication
## 52     7      rf 0.4251733    667 Replication
## 53     7    mars 0.4322098    667 Replication
## 54     7 elastic 0.4416262    667 Replication
## 55     7   lasso 0.4418310    667 Replication
## 56     7      lm 0.4450046    667 Replication
```

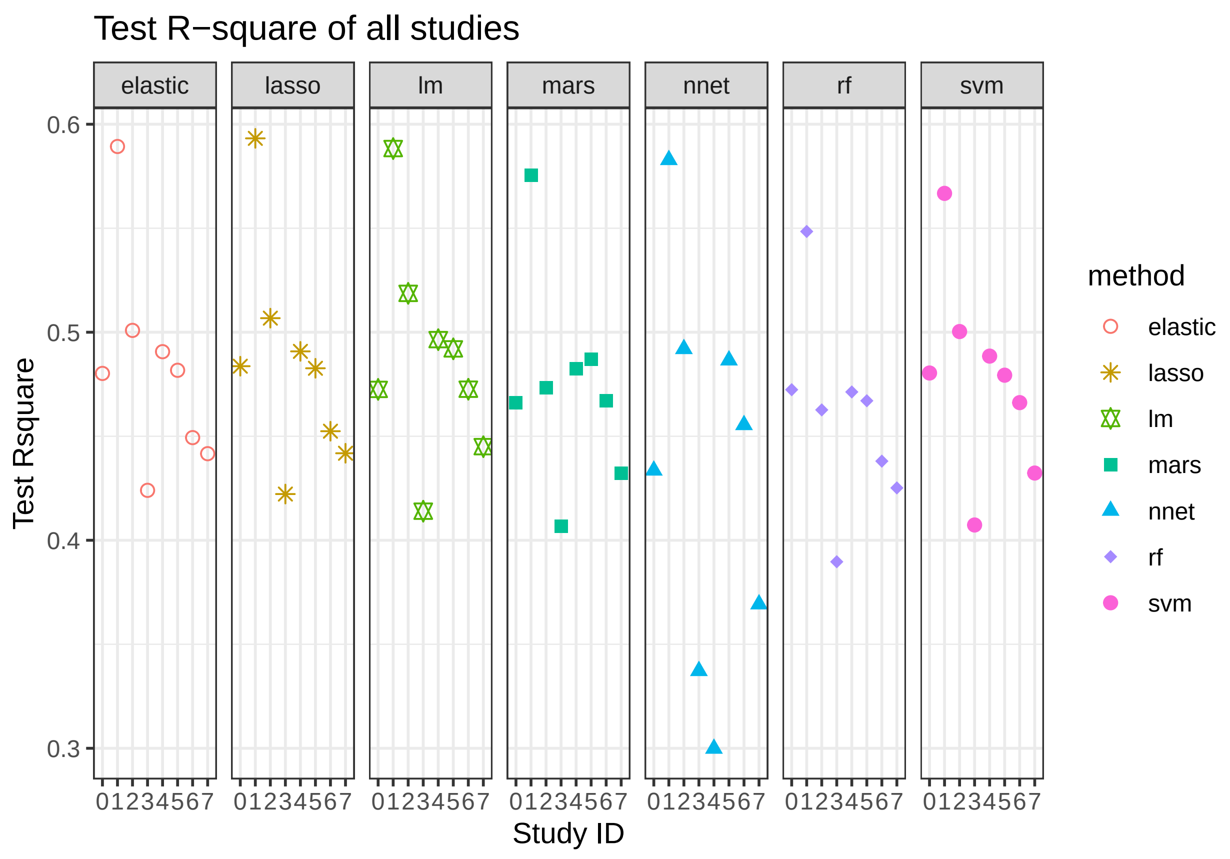
```
measure <- "Rsquare"
Rsq.dat$method <- factor(Rsq.dat$method,
                         levels = c("elastic", "lasso", "lm", "mars", "nnet", "rf", "svm"))
Rsq.dat$study <- factor(Rsq.dat$study)
Rsq.dat$study.type <- factor(Rsq.dat$study.type, levels = c("Target", "Replication") )
str(Rsq.dat)
```

```
## 'data.frame':    56 obs. of  5 variables:
##  $ study     : Factor w/ 8 levels "0","1","2","3",..: 1 1 1 1 1 1 1 2 2 2 ...
##  $ method    : Factor w/ 7 levels "elastic","lasso",..: 5 7 6 4 1 2 3 5 7 6 ...
##  $ Rsquare   : num  0.434 0.48 0.472 0.466 0.48 ...
##  $ test.N    : num  667 667 667 667 667 667 667 667 667 667 ...
##  $ study.type: Factor w/ 2 levels "Target","Replication": 1 1 1 1 1 1 1 2 2 2 ...
```

```
##------------------------
```

plot:

```
library(ggplot2)

ggplot( Rsq.dat, aes(study, Rsquare, group = method
                  , color = method, shape = method ) ) +
  facet_grid(~ method) +
  geom_point(size = 2) + # geom_line(size = 1) +
  # geom_errorbar(aes(ymin=mean-se, ymax=mean+se), width=.5, size = 1 ) +
  scale_shape_manual(values=c(1, 8, 11, 15, 17, 18, 19)) +
  ylab (paste("Test", measure, sep = " ")) + xlab("Study ID") +
  theme_bw()+
  # stat_summary(fun.y = mean, geom = "line") +
  # stat_summary(fun.y = mean, geom = "point", size = 2) +
  # # scale_x_discrete(
  # #   breaks = levels( ) ) +
  # stat_summary(fun.data = mean_se, geom = "errorbar", width = 0.3) +
  # theme_bw() +
  # # theme(legend.justification = c(1, 0), legend.position = c(0.95, 0.05)) +
  ggtitle(paste("Test R-square of all studies ")  )
```



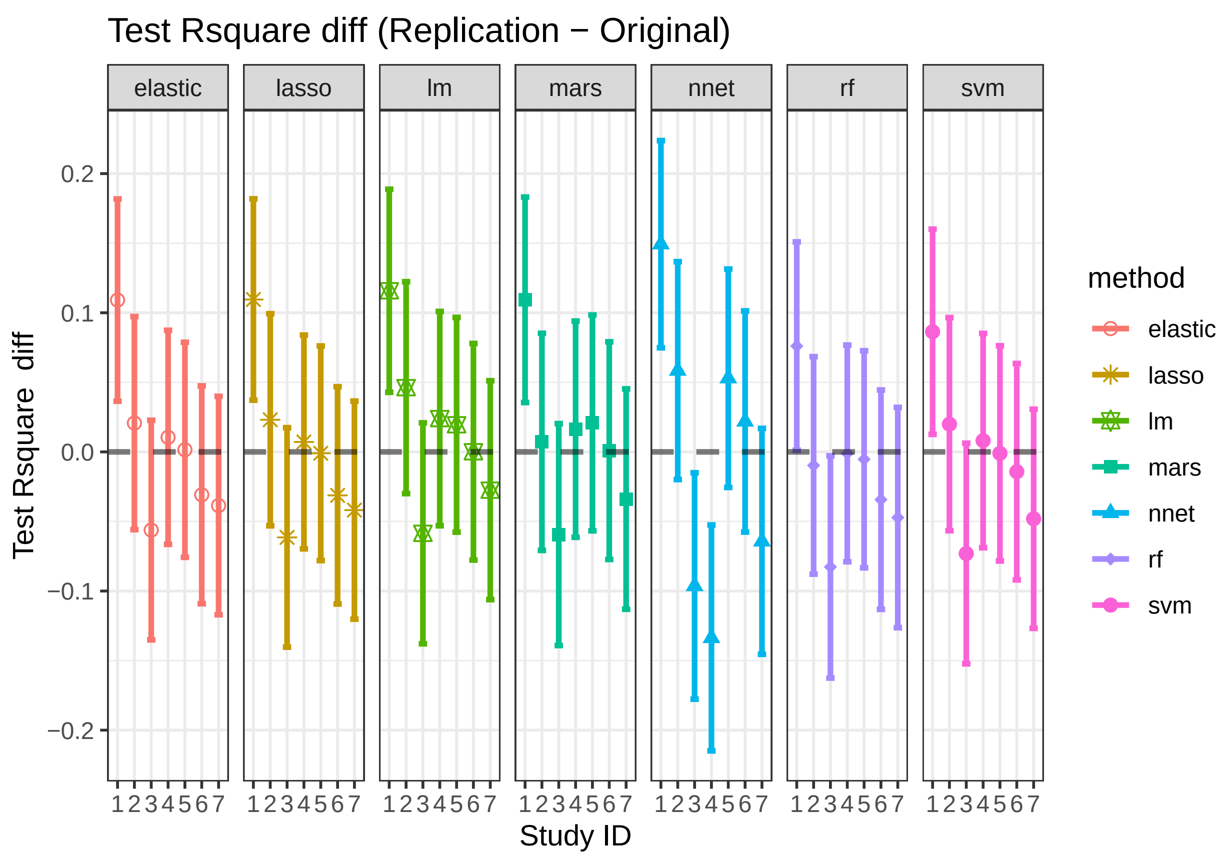
Replication procedures for test data:

```
##--get Rsq diff:---
Rsq.diff <- ddply( Rsq.dat , "method", function(subdata){
  ddply(subdata, "study", function(x){
    Rsquare.diff.se(x$Rsquare,subdata$Rsquare[subdata$study==target.study],
                N_1 = unique(Rsq.dat$test.N[Rsq.dat$study.type =="Replication"]),
               N_2 = unique(Rsq.dat$test.N[Rsq.dat$study.type =="Target"])
               )

  })
})

Rsq.diff <- Rsq.diff[!(Rsq.diff$study == 0),]
head(Rsq.diff)
```

```
##    method study            d         se
## 2 elastic     1  0.109100669 0.03707020
## 3 elastic     2  0.020696483 0.03906887
## 4 elastic     3 -0.056193336 0.04027042
## 5 elastic     4  0.010465283 0.03926133
## 6 elastic     5  0.001512319 0.03942209
## 7 elastic     6 -0.030844717 0.03993972
```

1) Test of Inconsistency:

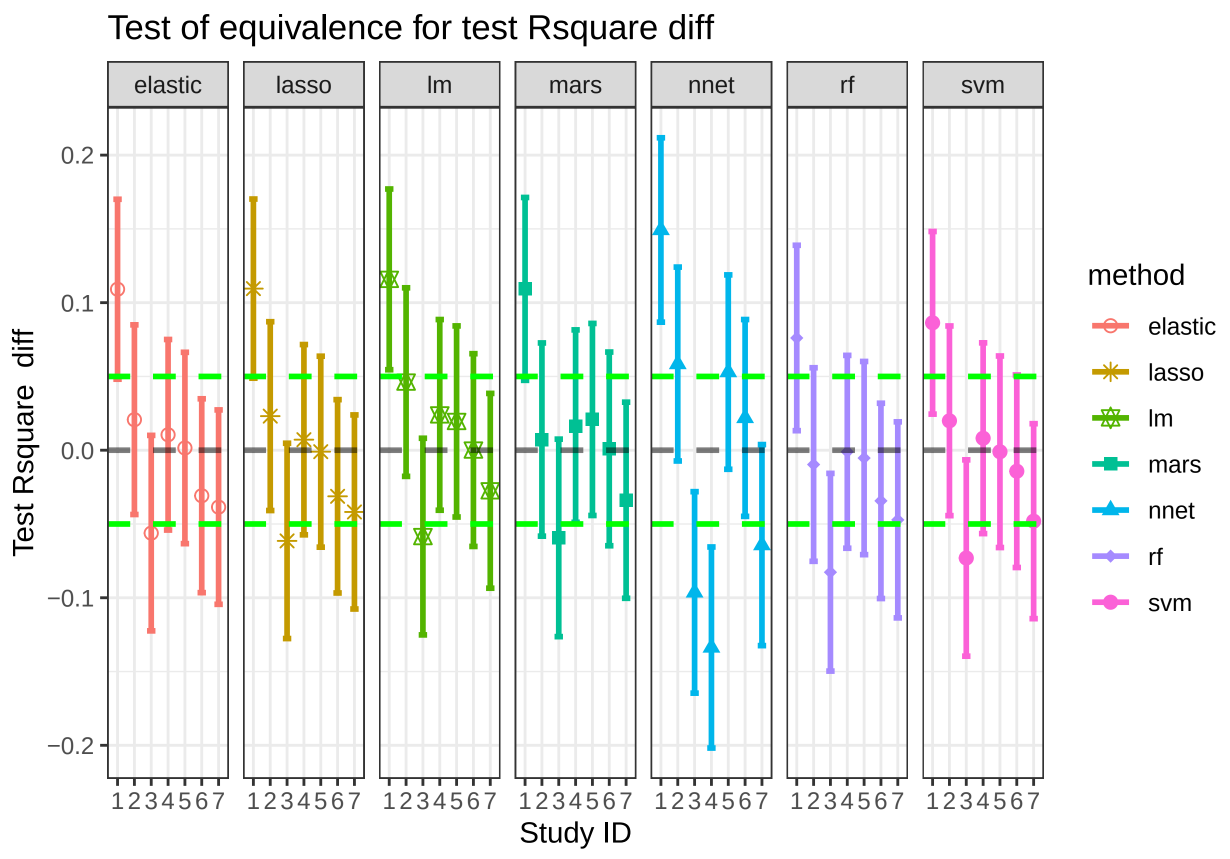
```
p.alpha = 0.975
# Rsq.diff$upper <- Rsq.diff$d + qnorm(p.alpha) * Rsq.diff$se
# Rsq.diff$lower <- Rsq.diff$d - qnorm(p.alpha) * Rsq.diff$se


ggplot( Rsq.diff, aes(study, d, group = method
                     , color = method, shape = method ) ) +
  facet_grid(~ method) +
  geom_point(size = 2) + # geom_line(size = 1) +
  geom_errorbar(aes(ymin = d - qnorm(p.alpha) * se,
                    ymax = d + qnorm(p.alpha) * se), width=.5, size = 1 ) +
  scale_shape_manual(values=c(1, 8, 11, 15, 17, 18, 19)) +
  geom_hline(yintercept = 0, color = "black", linetype="dashed", size = 1, alpha = 0.5) +
  # geom_hline(yintercept = +tost.value, color = "green", linetype="dashed", size = 1, alpha = 1) +
  # geom_hline(yintercept = -tost.value, color = "green", linetype="dashed", size = 1, alpha = 1) +
  ylab (paste("Test", measure," diff",  sep = " ")) + xlab("Study ID") +
  theme_bw()+
  # stat_summary(fun.y = mean, geom = "line") +
  # stat_summary(fun.y = mean, geom = "point", size = 2) +
  # # scale_x_discrete(
  # #   breaks = levels( ) ) +
  # stat_summary(fun.data = mean_se, geom = "errorbar", width = 0.3) +
  # theme_bw() +
  # # theme(legend.justification = c(1, 0), legend.position = c(0.95, 0.05)) +
  ggtitle(paste("Test Rsquare diff (Replication - Original)")  )
```



Test of Consistency:

```
tost.alpha = 0.95
tost.width = .05

ggplot( Rsq.diff, aes(study, d, group = method
                      , color = method, shape = method ) ) +
  facet_grid(~ method) +
  geom_point(size = 2) + # geom_line(size = 1) +
  geom_errorbar(aes(ymin = d - qnorm(tost.alpha) * se,
                    ymax = d + qnorm(tost.alpha) * se),
                width=.5, size = 1 ) +
  scale_shape_manual(values=c(1, 8, 11, 15, 17, 18, 19)) +
  geom_hline(yintercept = 0, color = "black", linetype="dashed", size = 1, alpha = 0.5) +
  geom_hline(yintercept = +tost.width, color = "green", linetype="dashed", size = 1, alpha = 1) +
  geom_hline(yintercept = -tost.width, color = "green", linetype="dashed", size = 1, alpha = 1) +
  ylab (paste("Test", measure," diff",  sep = " ")) + xlab("Study ID") +
  theme_bw()+
  # stat_summary(fun.y = mean, geom = "line") +
  # stat_summary(fun.y = mean, geom = "point", size = 2) +
  # # scale_x_discrete(
  # #   breaks = levels( ) ) +
  # stat_summary(fun.data = mean_se, geom = "errorbar", width = 0.3) +
  # theme_bw() +
```

```r
# # theme(legend.justification = c(1, 0), legend.position = c(0.95, 0.05)) +
ggtitle(paste("Test of equivalence for test Rsquare diff")  )
```



3) Estimating population R^2 by combining studies:

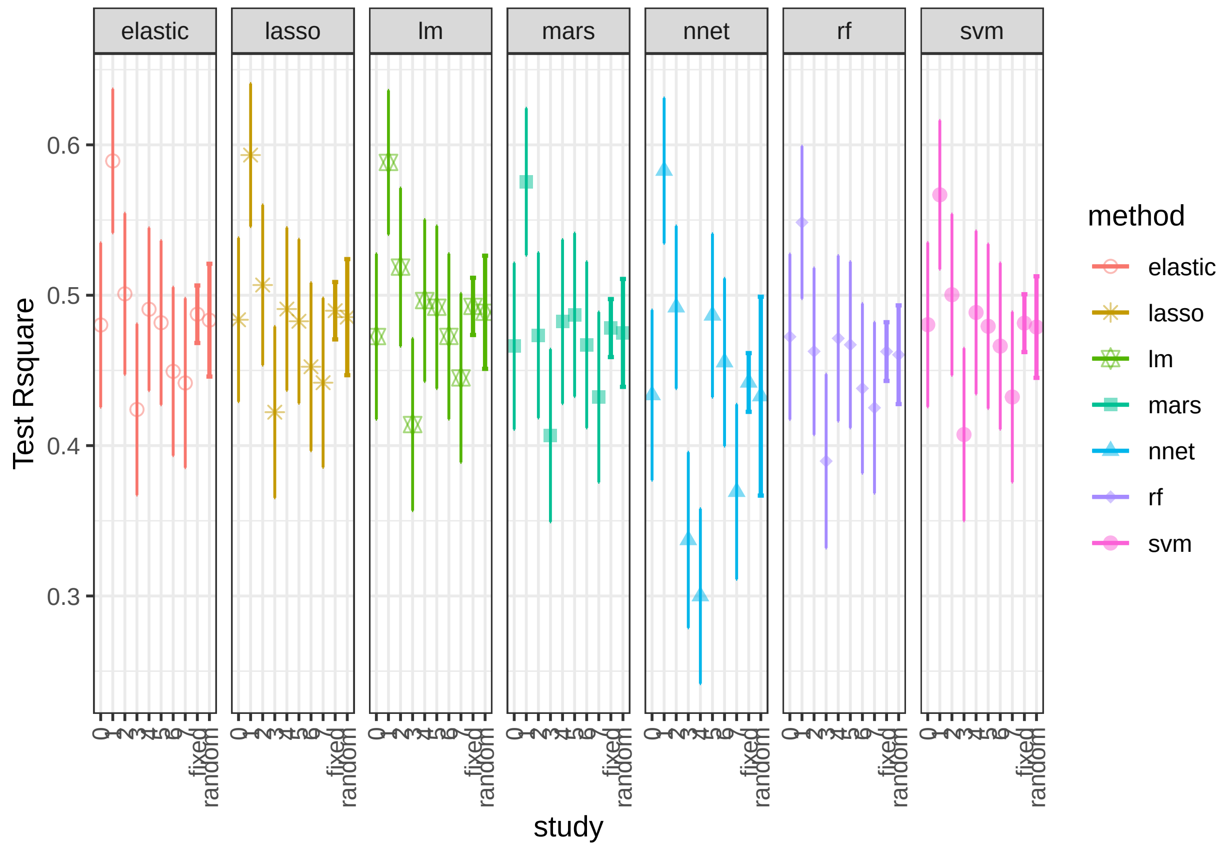
```r
#(i) get SE of each study's test Rsquare using Olkin and FInns equation 1:
head(Rsq.dat)
```

```
##   study  method   Rsquare test.N study.type
## 1     0    nnet 0.4335675    667     Target
## 2     0     svm 0.4804074    667     Target
## 3     0      rf 0.4723777    667     Target
## 4     0    mars 0.4661091    667     Target
## 5     0 elastic 0.4801965    667     Target
## 6     0   lasso 0.4836806    667     Target
```

```r
Rsq.dat <-   ddply( Rsq.dat, c("study", "method"),
                    function(x) {
  df1 <- olkin.se( x$Rsquare, x$test.N )
  df1$study.type <- x$study.type
  df1$test.N <- x$test.N
  df1
} )

head(Rsq.dat)
```

```
##   study  method   Rsquare         se study.type test.N
## 1     0 elastic 0.4801965 0.02789432     Target    667
## 2     0   lasso 0.4836806 0.02780769     Target    667
## 3     0      lm 0.4725220 0.02807905     Target    667
## 4     0    mars 0.4661091 0.02822691     Target    667
## 5     0    nnet 0.4335675 0.02888309     Target    667
## 6     0      rf 0.4723777 0.02808244     Target    667
```

```r
# (ii) get the meta-analytic population intervals:

meta.dat <-  meta.function2(Rsq.dat,
                       TE.name = "Rsquare",
                       se.name = "se",
                       study.name = "study",
                       method.name = "method")

# meta.summary <- meta.dat[meta.dat$type == "summary",]
# meta.fixed <- meta.summary[meta.summary$study =="fixed",]
# meta.random <- meta.summary[meta.summary$study =="random",]

# (iii) plot:
measure = "Rsquare"
meta.dat$study <- factor(meta.dat$study)


ggplot(meta.dat, aes(y=TE, x=study, ymin=lower, ymax=upper, shape = method, color = method))+
  #Add data points and color them black
  # geom_point(color = 'black', size = 0.5)+
  #Add 'special' points for the summary estimates, by making them diamond shaped
  geom_point(size= 2, alpha = 0.5)+
  #add the CI error bars
  geom_errorbar(width =.1)+
  # geom_errorbarh( data = subset(meta.summary, type=='summary'),  height =.15, size = 0.75)+
  geom_errorbar( data = subset(meta.dat, type=='summary'),
                width =.5, size = 0.75)+
  scale_shape_manual(values=c(1, 8, 11, 15, 17, 18, 19))+
  ylab(paste("Test", measure, sep = " "))+
  facet_grid( ~ method) + #, scales= 'free', space='free')+
  theme_bw() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))
```


elastic
lasso
lm
mars
nnet
rf
svm
Test Rsquare
0.6
0.5
0.4
0.3
0
1
2
3
4
5
6
7
fixed
random
study
method
elastic
lasso
lm
mars
nnet
rf
svm

5. Study1 training script

load caret functions script and R libraries:

```
##########################################################################################
#--------------libraries----------------------------##
libs2use <- c("caret","caretEnsemble", "reshape2", "ggplot2"
              , "randomForest", "e1071",  "glmnet", "nnet", "earth")
libs2install<- libs2use[!libs2use %in% rownames(installed.packages()  ) ]
if (length(libs2install)) {  install.packages(libs2install, dependencies = TRUE) }
sapply(libs2use, require, character.only = TRUE)
```

```
##         caret caretEnsemble      reshape2       ggplot2  randomForest
##          TRUE          TRUE          TRUE          TRUE          TRUE
##         e1071        glmnet          nnet         earth
##          TRUE          TRUE          TRUE          TRUE
```

```
##------------------------------------------------------------------------
##########################################################################################
```

```
#load source with wrapper functions written with caret functions:
source("caret_MLfunctions.R")
```

function to run training/cv and test and get outputs for all methods:

```
# function that trains and gets cv and test measures for each study sample.
# it takes in the dv string; the row-index of ESSdata used for training of a study sample;
# the caret machine-learning functions; names of machine-learning methods; num of folds;
# study number
study.pred.function <- function( dv, sample.index
                                 ,functions.list, method.names, CVfolds
                                 , study.ref){

  study.ref = study.ref
  # predictor.vars <- names(ESS.ger)[!names(ESS.ger) %in% c("essround", "idno", "study", dv)]

  # data.train <- ESS.ger[ESS.ger$idno %in% sample.index[[as.character(study.ref)]]$train.index ,
  #                        c(dv, predictor.vars) ]
  # data.test <- ESS.ger[ESS.ger$idno %in% sample.index[[as.character(study.ref)]]$test.index ,
  #                       c("idno",  dv, predictor.vars) ]


    predictor.vars <- names(ESS.ger)[!names(ESS.ger) %in% c("essround", "idno", "study", "row.num", dv)

  data.train <- ESS.ger[ESS.ger$row.num %in% sample.index[[as.character(study.ref)]]$train.index ,
                        c(dv, predictor.vars) ]
  data.test <- ESS.ger[ESS.ger$row.num %in% sample.index[[as.character(study.ref)]]$test.index ,
                       c(dv, predictor.vars) ]
  test.id.no <-  ESS.ger[ESS.ger$row.num %in% sample.index[[as.character(study.ref)]]$test.index ,
                         c("idno") ]

  formula1 <- as.formula(paste(dv,"~ . "))
  set.seed(12345)
  model.results <- train_model.function(formula1, data.train, dv,
                                        CVfolds, functions.list)
  names(model.results) <- method.names

  #combine all models into a caretlist object for analysis:
  multimodel <- lapply( model.results, function(x) x[[1]]  )
  models.accuracy <- caretmodel_accuracy.function2(multimodel)  #CV accuracy measures
  models.accuracy$study = study.ref #adding study ID


  #prediction using multimodels:

  #CV measures:
  write.csv(models.accuracy, paste("CVaccuracy.study.", study.ref,"_", dv, ".csv", sep = "") )


  #collect test predictions for all ess rounds into a dataframe
  models.pred <- as.data.frame( sapply(multimodel, predict, newdata = data.test))
  colnames(models.pred) <- names(multimodel)
  models.pred$idno <- test.id.no
  models.pred$y.obs <- data.test[, dv]
  models.pred


  head(models.pred)
  tail(models.pred)

  #write test predictios to csv files:
  write.csv( models.pred, paste("pred.model_study.", study.ref,"_", dv,".csv", sep = "" )  )



} # end of study pred function

###-----------------------------------------------------------
###-----------------------------------------------------------
```

ESSdata input:

```
#load ESS germany dataset with all rounds:
ESS.ger <- read.csv( "ESS_Ger_selVars_uniqueID_analysis_data_final.csv"  )
ESS.ger <- ESS.ger[, -1]
str(ESS.ger)
```

```
## 'data.frame':    17427 obs. of  38 variables:
##  $ essround: int  1 1 1 1 1 1 1 1 1 1 ...
##  $ idno    : int  101114 101120 101126 101322 101405 101417 101429 101512 101601 101625 ...
##  $ ppltrst : int  0 6 0 0 4 0 7 3 4 0 ...
##  $ pplfair : int  5 6 0 7 4 4 6 4 4 5 ...
##  $ pplhlp  : int  4 4 0 5 2 0 4 4 3 4 ...
##  $ lrscale : int  8 3 10 8 2 5 6 3 3 7 ...
##  $ stflife : int  0 8 3 3 7 7 7 7 7 8 ...
##  $ stfeco  : int  0 4 5 0 3 4 0 5 7 6 ...
##  $ stfgov  : int  0 7 5 2 6 2 3 4 4 4 ...
##  $ stfdem  : int  8 10 5 6 9 5 8 8 7 9 ...
##  $ stfedu  : int  9 8 7 6 7 2 2 9 5 2 ...
##  $ stfhlth : int  10 10 7 6 10 2 5 9 8 9 ...
##  $ happy   : int  2 8 7 6 8 9 7 8 9 8 ...
##  $ sclmeet : int  6 6 4 5 6 5 5 4 3 5 ...
##  $ eduyrs  : int  10 11 9 9 13 12 11 12 10 10 ...
##  $ sclact  : int  4 2 3 3 3 3 3 3 2 3 ...
##  $ health  : int  1 1 3 2 2 3 3 3 3 3 ...
##  $ ipcrtiv : int  3 6 2 2 6 3 4 4 5 3 ...
##  $ imprich : int  5 5 2 4 4 3 4 4 5 3 ...
##  $ ipeqopt : int  6 6 5 3 6 5 6 5 6 6 ...
##  $ ipshabt : int  4 5 3 6 4 3 3 4 6 3 ...
##  $ impsafe : int  2 5 6 6 4 6 6 6 6 6 ...
##  $ impdiff : int  6 5 6 4 6 4 4 6 3 6 ...
##  $ ipfrule : int  1 5 4 2 4 5 6 5 5 5 ...
##  $ ipudrst : int  6 6 4 2 6 6 6 6 6 5 ...
##  $ ipmodst : int  2 3 6 2 3 5 6 5 4 4 ...
##  $ ipgdtim : int  5 6 5 3 6 4 5 6 4 5 ...
##  $ impfree : int  6 5 5 6 6 5 4 5 6 6 ...
##  $ iphlppl : int  2 5 5 5 5 5 6 5 5 6 ...
##  $ ipsuces : int  2 5 6 5 6 4 6 5 5 3 ...
##  $ ipstrgv : int  3 5 6 6 6 6 6 6 6 6 ...
##  $ ipadvnt : int  6 4 3 3 4 2 3 3 3 3 ...
##  $ ipbhprp : int  2 5 5 3 4 5 6 6 5 6 ...
##  $ iprspot : int  6 6 5 5 3 4 6 5 5 4 ...
##  $ iplylfr : int  4 5 6 5 6 6 6 6 5 6 ...
##  $ impenv  : int  5 6 6 5 5 6 6 6 5 5 ...
##  $ imptrad : int  1 6 6 2 2 6 6 6 2 6 ...
##  $ impfun  : int  6 6 4 2 6 3 6 5 2 6 ...
```

```r
table(ESS.ger$essround)
```

```
##
##    1    2    3    4    5    6    7    8
## 2325 2100 1773 1013 2494 2577 2625 2520
```

create 8 random samples to serve as studies 0 (original) to 7:

```r
#select an original study
original.N <- 2000
num_of_reps <- 7
reps.N <- 2000
not_used_N <- nrow(ESS.ger ) - (original.N + num_of_reps* reps.N )
```

```r
#construct a vector to index studies 0 [original] to 7, and randomly allot that to ESS.ger:
reps.int <- c( rep(0, original.N), rep(1:7, each = reps.N), rep(NA, not_used_N) )
set.seed(1234)
reps.int <- sample(reps.int, replace = FALSE) #randomize the study rows
ESS.ger$study <- reps.int
# str(ESS.ger)

#remove the rows not used in this replication analysis:
ESS.ger <- na.omit(ESS.ger)

#check:-----
table(ESS.ger$study)
```

```
##
##    0    1    2    3    4    5    6    7
## 2000 2000 2000 2000 2000 2000 2000 2000
```

divide all studies into training and test sets:

```r
#getting training and test sizes for each study.-------------------
N.study <- table(ESS.ger$study)
#ensure that test set has size >=500
# test.N <- apply(N.rounds, 1, function(x) round ( max(1/3 *x, 500) ) )
test.N <- apply(N.study, 1, function(x) round ( 1/3 *x ) )
train.N <- N.study - test.N

#get sampling indices for train/test for each round:
library(plyr)
ESS.ger$row.num <- 1:nrow(ESS.ger)
str(ESS.ger)
```

```
## 'data.frame':    16000 obs. of  40 variables:
##  $ essround: int  1 1 1 1 1 1 1 1 1 1 ...
##  $ idno    : int  101114 101120 101126 101322 101405 101417 101429 101512 101601 101625 ...
##  $ ppltrst : int  0 6 0 0 4 0 7 3 4 0 ...
##  $ pplfair : int  5 6 0 7 4 4 6 4 4 5 ...
##  $ pplhlp  : int  4 4 0 5 2 0 4 4 3 4 ...
##  $ lrscale : int  8 3 10 8 2 5 6 3 3 7 ...
##  $ stflife : int  0 8 3 3 7 7 7 7 7 8 ...
##  $ stfeco  : int  0 4 5 0 3 4 0 5 7 6 ...
##  $ stfgov  : int  0 7 5 2 6 2 3 4 4 4 ...
##  $ stfdem  : int  8 10 5 6 9 5 8 8 7 9 ...
##  $ stfedu  : int  9 8 7 6 7 2 2 9 5 2 ...
##  $ stfhlth : int  10 10 7 6 10 2 5 9 8 9 ...
##  $ happy   : int  2 8 7 6 8 9 7 8 9 8 ...
##  $ sclmeet : int  6 6 4 5 6 5 5 4 3 5 ...
##  $ eduyrs  : int  10 11 9 9 13 12 11 12 10 10 ...
##  $ sclact  : int  4 2 3 3 3 3 3 3 2 3 ...
##  $ health  : int  1 1 3 2 2 3 3 3 3 3 ...
##  $ ipcrtiv : int  3 6 2 2 6 3 4 4 5 3 ...
##  $ imprich : int  5 5 2 4 4 3 4 4 5 3 ...
##  $ ipeqopt : int  6 6 5 3 6 5 6 5 6 6 ...
##  $ ipshabt : int  4 5 3 6 4 3 3 4 6 3 ...
##  $ impsafe : int  2 5 6 6 4 6 6 6 6 6 ...
##  $ impdiff : int  6 5 6 4 6 4 4 6 3 6 ...
##  $ ipfrule : int  1 5 4 2 4 5 6 5 5 5 ...
##  $ ipudrst : int  6 6 4 2 6 6 6 6 6 5 ...
##  $ ipmodst : int  2 3 6 2 3 5 6 5 4 4 ...
##  $ ipgdtim : int  5 6 5 3 6 4 5 6 4 5 ...
##  $ impfree : int  6 5 5 6 6 5 4 5 6 6 ...
##  $ iphlppl : int  2 5 5 5 5 5 6 5 5 6 ...
##  $ ipsuces : int  2 5 6 5 6 4 6 5 5 3 ...
##  $ ipstrgv : int  3 5 6 6 6 6 6 6 6 6 ...
##  $ ipadvnt : int  6 4 3 3 4 2 3 3 3 3 ...
##  $ ipbhprp : int  2 5 5 3 4 5 6 6 5 6 ...
##  $ iprspot : int  6 6 5 5 3 4 6 5 5 4 ...
##  $ iplylfr : int  4 5 6 5 6 6 6 6 5 6 ...
##  $ impenv  : int  5 6 6 5 5 6 6 6 5 5 ...
##  $ imptrad : int  1 6 6 2 2 6 6 6 2 6 ...
##  $ impfun  : int  6 6 4 2 6 3 6 5 2 6 ...
##  $ study   : num  3 4 3 4 4 0 7 5 0 6 ...
##  $ row.num : int  1 2 3 4 5 6 7 8 9 10 ...
##  - attr(*, "na.action")= 'omit' Named int  19 22 24 36 71 74 75 96 99 118 ...
##   ..- attr(*, "names")= chr  "19" "22" "24" "36" ...
```

```
sample.index <- dlply(ESS.ger, "study", function(x) {

  study.name <- as.character(unique(x$study)) #get round number as character

  # train.index <- sample(idno, size = train.N[study.name], replace = FALSE )
  train.index.1 <- sample(1:nrow(x), size = train.N[study.name], replace = FALSE )
  train.index <- x$row.num[ train.index.1 ]
  test.index <- x$row.num[ -train.index.1 ]
  # test.index <- sample ( idno.test, size = test.N[study.name], replace = FALSE   )
  list( train.index = train.index, test.index = test.index )
})
```

variable details for use in training loop:

```
CVfolds = 10
method.names <- c("nnet", "svm", "rf" , "mars", "elastic", "lasso" , "lm") #***P.S: this should match
study.ref <- 0:7
dv <- c( "happy")

#get all caret_ML functions into a list for use in actual training loop later:
functions.list <- list( nnet.function, svm.function, rf.function, mars.function
                        , elastic.function,  lasso.function, lm.function )
```

The actual training loop to run machine-learning for all studies:

```r
  for (study in study.ref){
    system.time({
      study.pred.function( dv, sample.index
                           ,functions.list, method.names, CVfolds
                           , study)

    })
  }



##-----------------------------------------------------
```

6. Study 2 training script

R libraries:

```
##########################################################################################
#--------------libraries----------------------------##
libs2use <- c("tidyr",  "caret","caretEnsemble",  "reshape2", "plyr", "ggplot2"
              , "randomForest", "e1071", "glmnet", "nnet", "earth"
              , "mvtnorm", "tmvtnorm")
libs2install<- libs2use[!libs2use %in% rownames(installed.packages()  ) ]
if (length(libs2install)) {  install.packages(libs2install, dependencies = TRUE) }
sapply(libs2use, require, character.only = TRUE)
```

```
## Loading required package: mvtnorm

## Loading required package: tmvtnorm

## Loading required package: stats4

## Loading required package: gmm

## Loading required package: sandwich

##         tidyr         caret caretEnsemble      reshape2          plyr
##          TRUE          TRUE          TRUE          TRUE          TRUE
##       ggplot2  randomForest         e1071        glmnet          nnet
##          TRUE          TRUE          TRUE          TRUE          TRUE
##         earth       mvtnorm      tmvtnorm
##          TRUE          TRUE          TRUE
```

```
##-----------------------------------------------------------------------------
##########################################################################################
```

user-defined functions:

```
#################--------------------------------------------#######################
#################-----function to get sample of given R-square-------#################

get_sample_given_rsquare <- function( pop.data, sample.data = pop.data,
                                      dv, formula1, R.square.change ){

  reg.model <- lm(formula1, data = pop.data)
  R.square.old = summary(reg.model)$r.squared
  R.square.new = R.square.old - R.square.change
  if(R.square.new > R.square.old){
    message("new Rsquare ", R.square.new, " > original Rsquare: ", round(R.square.old, 4) )
    message("function aborted; please input Rsquare less than ",  round(R.square.old, 4) )
    return(NULL)
```

```r
  }

  #get variance of residuals, regression, and total:
  var.err <- var(reg.model$ residuals)
  var.fitted <- var( reg.model$fitted.values )
  var.tot <- var(pop.data[, dv] )

  #get the variance to add to the DV error term
  var.add = (var.fitted/R.square.new) - var.fitted - var.err

  #add the new error variance to the data dv:
  sample.data[, dv] <- sample.data[, dv] + rnorm(length(sample.data[, dv] ),
                                        mean = 0, sd = sqrt(var.add) )

  #return data with new DV scores
  return(sample.data)

}
# #################------------end of function-----------------########################

#################------------------------------------------------------------------####
#wrapper function specific to  ESS data:
#uses get_sample_given_Rsquare to return ESSdata with Rsq of specified studies changed.
changing_ESS_studies_Rsq <- function(pop.dat, sample.dat, studies_to_change, Rsq.change){
  library(plyr)

  sample.dat <-  ddply(sample.dat, "study", function(df){
    if (unique(df$study) %in% studies_to_changeRsq){
      df <- get_sample_given_rsquare(pop.dat, sample.data = df, dv, formula1, Rsq.change )
    } else df

  })

  return(sample.dat)
}


#######----------function to run training for all methods------------------##########


#Note: Rsq.num is used for purpose of write.csv: This argument is diff from
#earlier versions of this function used in Study 1
study.pred.function <- function( mydata, dv, xname, sample.index
                                 ,functions.list, method.names, CVfolds
                                 , study.ref, Rsq.num){

  library(caret)


  study.ref = study.ref
  # predictor.vars <- names(mydata)[!names(mydata) %in% c("essround", "idno", "study", dv)]
  #
  # data.train <- mydata[mydata$idno %in% sample.index[[as.character(study.ref)]]$train.index ,
  #                      c(dv, predictor.vars) ]
  # data.test <- mydata[mydata$idno %in% sample.index[[as.character(study.ref)]]$test.index ,
  #                     c("idno",  dv, predictor.vars) ]

    predictor.vars <- names(mydata)[!names(mydata) %in% c("essround", "idno", "study", "row.num", dv)]

  data.train <- mydata[mydata$row.num %in% sample.index[[as.character(study.ref)]]$train.index ,
                       c(dv, predictor.vars) ]
  data.test <- mydata[mydata$row.num %in% sample.index[[as.character(study.ref)]]$test.index ,
                      c(dv, predictor.vars) ]
  test.id.no <-  mydata[mydata$row.num %in% sample.index[[as.character(study.ref)]]$test.index ,
                        c("idno") ]

  formula1 <- as.formula(paste(dv,"~ . "))

  set.seed(12345)
  model.results <- train_model.function(formula1, data.train, dv,
                                        CVfolds, functions.list)
  names(model.results) <- method.names

  #combine all models into a caretlist object for analysis:
  multimodel <- lapply( model.results, function(x) x[[1]]  )
  models.accuracy <- caretmodel_accuracy.function2(multimodel)  #CV accuracy measures
  models.accuracy$study = study.ref #adding study ID


  #prediction using multimodels:
  #CV measures:
  write.csv(models.accuracy, paste("CVaccuracy.study.", study.ref,"_",
                                   dv,".Rsq.",Rsq.num ,".csv", sep = "") )


  #collect test predictions for all 8 studies into a dataframe
  models.pred <- as.data.frame( sapply(multimodel, predict, newdata = data.test))
  colnames(models.pred) <- names(multimodel)
  models.pred$idno <- test.id.no
  models.pred$y.obs <- data.test[, dv]
  models.pred


  head(models.pred)
  tail(models.pred)

  write.csv( models.pred, paste("pred.model_study.", study.ref,"_",
                                dv,".Rsq.", Rsq.num ,".csv", sep = "" )  )


} # end of study pred function

###-------------------------------------------------------
```

Training main script:

get source code for caret functions and initialize variables:

```
source("caret_MLfunctions.R")
library(caret)
#step 1a:train models on one round/studysample. This study is the reference study

CVfolds = 10
method.names <- c("nnet", "svm", "rf" , "mars", "elastic", "lasso" , "lm") #***P.S: this should match f
study.ref <- 0:7
xname <- "stflife"
dv <- c( "happy")
Rsq.change = c( 0.05, 0.1, 0.15, 0.2, 0.3, 0.4)
studies_to_changeRsq <- 4:7

#get all caret_ML functions into a list for use in next line:
functions.list <- list( nnet.function, svm.function, rf.function, mars.function
                        , elastic.function,  lasso.function, lm.function )


##---------------------------------------------------------
```

data input:

```
#load ESS germany dataset with all rounds as population data:--------------
pop.dat <- read.csv( "ESS_Ger_selVars_uniqueID_analysis_data_final.csv"  )
pop.dat <- pop.dat[, -1]  #remove X column
pop.dat <- pop.dat [, -c(1, 2)] #remove ID and esround

formula1 = as.formula(paste(dv,"~ . "))


#get the 8 studies dataset:---------:
ESS.dat <- read.csv("ESS_Ger_8Studies.csv")

ESS.dat <- ESS.dat[, -1] # removing redundant X column
str(ESS.dat)
```

```
## 'data.frame':    16000 obs. of  39 variables:
##  $ essround: int  1 1 1 1 1 1 1 1 1 1 ...
##  $ idno    : int  101114 101120 101126 101322 101405 101417 101429 101512 101601 101625 ...
##  $ ppltrst : int  0 6 0 0 4 0 7 3 4 0 ...
##  $ pplfair : int  5 6 0 7 4 4 6 4 4 5 ...
##  $ pplhlp  : int  4 4 0 5 2 0 4 4 3 4 ...
##  $ lrscale : int  8 3 10 8 2 5 6 3 3 7 ...
##  $ stflife : int  0 8 3 3 7 7 7 7 7 8 ...
##  $ stfeco  : int  0 4 5 0 3 4 0 5 7 6 ...
##  $ stfgov  : int  0 7 5 2 6 2 3 4 4 4 ...
##  $ stfdem  : int  8 10 5 6 9 5 8 8 7 9 ...
##  $ stfedu  : int  9 8 7 6 7 2 2 9 5 2 ...
##  $ stfhlth : int  10 10 7 6 10 2 5 9 8 9 ...
##  $ happy   : int  2 8 7 6 8 9 7 8 9 8 ...
##  $ sclmeet : int  6 6 4 5 6 5 5 4 3 5 ...
##  $ eduyrs  : int  10 11 9 9 13 12 11 12 10 10 ...
##  $ sclact  : int  4 2 3 3 3 3 3 3 2 3 ...
##  $ health  : int  1 1 3 2 2 3 3 3 3 3 ...
##  $ ipcrtiv : int  3 6 2 2 6 3 4 4 5 3 ...
##  $ imprich : int  5 5 2 4 4 3 4 4 5 3 ...
##  $ ipeqopt : int  6 6 5 3 6 5 6 5 6 6 ...
##  $ ipshabt : int  4 5 3 6 4 3 3 4 6 3 ...
##  $ impsafe : int  2 5 6 6 4 6 6 6 6 6 ...
##  $ impdiff : int  6 5 6 4 6 4 4 6 3 6 ...
##  $ ipfrule : int  1 5 4 2 4 5 6 5 5 5 ...
##  $ ipudrst : int  6 6 4 2 6 6 6 6 6 5 ...
##  $ ipmodst : int  2 3 6 2 3 5 6 5 4 4 ...
##  $ ipgdtim : int  5 6 5 3 6 4 5 6 4 5 ...
##  $ impfree : int  6 5 5 6 6 5 4 5 6 6 ...
##  $ iphlppl : int  2 5 5 5 5 5 6 5 5 6 ...
##  $ ipsuces : int  2 5 6 5 6 4 6 5 5 3 ...
##  $ ipstrgv : int  3 5 6 6 6 6 6 6 6 6 ...
##  $ ipadvnt : int  6 4 3 3 4 2 3 3 3 3 ...
##  $ ipbhprp : int  2 5 5 3 4 5 6 6 5 6 ...
##  $ iprspot : int  6 6 5 5 3 4 6 5 5 4 ...
##  $ iplylfr : int  4 5 6 5 6 6 6 6 5 6 ...
##  $ impenv  : int  5 6 6 5 5 6 6 6 5 5 ...
##  $ imptrad : int  1 6 6 2 2 6 6 6 2 6 ...
##  $ impfun  : int  6 6 4 2 6 3 6 5 2 6 ...
##  $ study   : int  0 5 5 5 7 5 0 2 5 4 ...
```

training loop begins:

```
##------------------------------------------------------------------------------##
##-----------------------------loop to analyze different Rsq datasets:-----------

for(Rsq.num in Rsq.change){

  #get 4 new datasets, with studies 4 to 7 having Rsquare of 0.4, 0.3, 0.2 and 0.1 in loops
  ESS.ger <- changing_ESS_studies_Rsq( pop.dat, ESS.dat,
                                       studies_to_changeRsq, Rsq.num )
  write.csv( ESS.ger, paste("ESS.ger.Rsq_", Rsq.num, ".csv" , sep = "") )

  # str(ESS.ger)
  ################
  #print out the R^2 of samples to check if R^2 was changed correctly:-------
  ddply(ESS.ger, "study", function(df){
    df <- df[, -c(1:2)]

    lm.model <- lm(formula1, data = df)
    cat(paste( "\nstudy:", unique(df$study), "; RsqChange = ", Rsq.num, "New Rsquare: ", summary(lm.mode

  } )
```

```r
  #---------------------------------------------------------------------------
  ################


  #-------------------------------------------------------------------##
  #getting training and test sizes for each study:--------------
  N.study <- table(ESS.ger$study)
  #ensure that test set has size >=500
  # test.N <- apply(N.rounds, 1, function(x) round ( max(1/3 *x, 500) ) )
  test.N <- apply(N.study, 1, function(x) round ( 1/3 *x ) )
  train.N <- N.study - test.N

  #get sampling indices for train/test for each round:
library(plyr)
ESS.ger$row.num <- 1:nrow(ESS.ger)
str(ESS.ger)
sample.index <- dlply(ESS.ger, "study", function(x) {

  study.name <- as.character(unique(x$study)) #get round number as character

  # train.index <- sample(idno, size = train.N[study.name], replace = FALSE )
  train.index.1 <- sample(1:nrow(x), size = train.N[study.name], replace = FALSE )
  train.index <- x$row.num[ train.index.1 ]
  test.index <- x$row.num[ -train.index.1 ]
  # test.index <- sample ( idno.test, size = test.N[study.name], replace = FALSE   )
  list( train.index = train.index, test.index = test.index )
})


  #training for each machine-learning method:
  for (dv.name in dv){
    for (study in study.ref){
        #write csv files of predictions and CV preds:
        study.pred.function( ESS.ger, dv.name, xname, sample.index
                             ,functions.list, method.names, CVfolds
                             , study, Rsq.num)
    }
  }
} # end of Rsq loop


##-----------------------------------------------------
```