The R codes are structured to mirror the experimental setup (see Figure ??). The source codes are divided into the following self-explanatory scripts. These scripts read and train the models on the training sets with various feature selection methods and evaluate their performances on the test sets in an automatical fashion.

R Codes for Regression Tasks

Baseline Learner for Regression

```
# Baseline learners without interactions: regression tasks ----
# Compare baseline learners with other methods
# 0. Load packages ----
library(mlr)
library(tidyverse)

# 1. Run baseline learners ----
# Initialise the data frame to store results,
# Prepare data sets and target continuous features
result      <- data.frame()

data_files  <- c('BostonHousing', 'Ailerons', 'Elevators')
targets     <- c('medv', 'V41', 'V19')

# Common learner: linear regression
wrapper     <- makeLearner('regr.lm')

# Loop for each dataset
for(j in 1:length(data_files) ){

  train_data  <- read.csv(paste0(data_files[j], '_train.csv'))
  test_data   <- read.csv(paste0(data_files[j],'_test.csv'))

  train_task  <- makeRegrTask(data = train_data, target = targets[j])
  test_task   <- makeRegrTask(data = test_data, target = targets[j])

  baseModel   <- train(wrapper, train_task)
  train_pred  <- predict(baseModel, train_task)
  test_pred   <- predict(baseModel, test_task)

  result[j, 'Data'] <- data_files[j]
  result[j, 'AIC']  <- AIC(baseModel$learner.model)
  result[j, 'BIC']  <- BIC(baseModel$learner.model)
  result[j, 'Train_RMSE'] <- mlr::performance(train_pred, mlr::rmse)
  result[j, 'Test_RMSE']  <- mlr::performance(test_pred, mlr::rmse)

}

# Save the result as a csv file
write.csv(result, 'baselearner_regr.csv', row.names = FALSE)
```

Modified Make Regression Task Function

```
#' @description Generating pairwise interaction terms of a data and
# '             pass in to a regression task
#'
```

```r
#' @param data A data frame containing the features and target variable(s)
#' @param target Name of the target variable
#' @param order Interaction order. The allowed values are 1, 2, and 3.
#'              Default is 2L

modifiedMakeRegrTask <- function(data, target, order = 2,
                                 remove.constant = TRUE,...){

  # Argument check on "order"
  # It is not necessary to check on "data" or "target" as they will
  # be handled by makeClassifTask function
  if( !order %in% c(1, 2, 3) | !inherits(order, 'integer')){
    stop('order must be 1, 2, or 3.')
  }

  # create the classification task

  task   <- mlr::makeRegrTask(data = data, target = target, ...)


  if( order %in% c(2, 3)){
    # extract target and descriptive features

    y   <- task$env$data[, target]

    if( order == 2){
      fml <- formula(paste0(target, "~.*."))
    }else{
      fml <- formula(paste0(target, "~.^3"))
    }

    X   <- data.frame(model.matrix(fml, task$env$data))

    # Remove constant terms
    if( remove.constant){
      X   <- mlr::removeConstantFeatures(X)
    }


    # bind the data
    newdata <- cbind(X,y)
    colnames(newdata)[ncol(newdata)] <- target

    # prompt warning message if p > n
    p <- ncol(X)
    if( p > nrow(X)){
      warning('More features than number of observations.')
    }

    # reconfigure the task
    task   <- mlr::makeRegrTask(data = newdata, target = target, ...)
  }
```

```r
  return(task)

}
```

Automatic Feature Selection Script for Regression Tasks

```r
# 0. Preliminary ----
# Load the relevant packages
library(mlr)
library(spFSR)
library(jsonlite)
library(glinternet)

# Source the modificed modified make-regression task with higher orders
source('modifiedMakeRegrTask.R')

# Common learner: linear regression
wrapper     <- makeLearner('regr.lm')

# Datasets and number of CV
data_files  <- c('BostonHousing', 'Ailerons', 'Elevators')
targets     <- c('medv', 'V41', 'V19')
numberCV    <- c(4, 10, 10)

# 1. Learning interactions using SFS for linear regresssion task ----
# Define SFS control
SFSctrl    <- makeFeatSelControlSequential(method = 'sfs',
                                           alpha = 0.001, beta = -0.0005)

for(j in 1:length(data_files)){

  # Initialise the all_result data.frame to store the result
  m <- 0
  results_summary   <- data.frame()

  data_name   <- data_files[j]
  target <- targets[j]

  try(
    {
      data        <- read.csv(paste0(data_files[j], '_train.csv'))
      test_data   <- read.csv(paste0(data_files[j],'_test.csv'))
    }
  )

  task <- makeRegrTask(data = data, target = target)

  # Create a vector of random seeds for reproducibility
  if( data_name == 'BostonHousing'){
    seed_vector <- c(1, 1010, 4, 781, 9990, 9999, 81, 46, 67, 2, 3, 1017)
  }else{
    seed_vector <- c(1, 4, 781)
  }
```

```r
for(i in 1:length(seed_vector)){

  seed.number <- seed_vector[i]
  set.seed(seed.number)

  m <- m + 1

  # Resampling
  rdesc       <- makeResampleDesc(method = 'CV' , iters = numberCV[j])

  # Extract main effects
  main_sfeats <- selectFeatures(wrapper, task = task, resampling = rdesc,
                               control = SFSctrl, show.info = TRUE,
                               measures = mlr::rmse )

  # Obtain the performance with main effects only
  reduced_main_task <- makeRegrTask(id = 'reduced main',
                                   data = data[, c(main_sfeats$x , targets[j])],
                                   target = targets[j])

  # Add interactions
  second_task  <- modifiedMakeRegrTask(id = 'interaction',
                                      data = data[, c(main_sfeats$x , target)],
                                      target = target, order = 2L)

  # Specify the maximum number of trials
  z <- 1
  success <- FALSE

  tryCatch({
    second_sfeats <- selectFeatures(wrapper, task = second_task,
                                   resampling = rdesc,
                                   control = SFSctrl, show.info = TRUE,
                                   measures = mlr::rmse)
    success <- TRUE

  },
  error = function(e){conditionMessage(e)}
  )


  results_summary[m, 'dataset']      <- data_name
  results_summary[m, 'seed_number']  <- seed.number
  results_summary[m, 'p_0']          <- length(main_sfeats$x)

  if(exists('second_sfeats')){
    signif_terms     <- as.character(unique(c(main_sfeats$x , second_sfeats$x)))
    results_summary[m, 'p_1'] <- length(signif_terms) - length(main_sfeats$x)

    # Add the omitted main effects
    third_task <- makeRegrTask(id = 'strong hierarchy',
                              data =
                               second_task$env$data[, c(signif_terms, targets[j])],
```

```r
                                target = targets[j])

      constrainedMod   <- train( wrapper, third_task )
      constrainedPred  <- predict( constrainedMod, third_task)

      results_summary[m, 'train_rmse'] <- performance( constrainedPred,
                                                       measure = mlr::rmse)
      results_summary[m, 'train_AIC']  <- AIC( constrainedMod$learner.model)
      results_summary[m, 'train_BIC']  <- BIC( constrainedMod$learner.model)


      sub_test_task  <- modifiedMakeRegrTask(data = test_data, target = targets[j],
                                             order = 2L, remove.constant = FALSE)

      sub_test_task <- makeRegrTask(data =
                                     sub_test_task$env$data[, c(signif_terms, targets[j])],
                                    target = targets[j], id = 'test_subset')

      sub_pred_test                  <- predict(constrainedMod , sub_test_task)
      results_summary[m, 'test_rmse']<- performance( sub_pred_test, mlr::rmse)

      output <- list(coeff  = constrainedMod$learner.model$coefficients,
                     result = results_summary[m, ],
                     features = signif_terms)

      write(toJSON(output, auto_unbox = TRUE, pretty = TRUE, factor = 'string'),
            paste0('GA_',data_name,'_',seed.number,'.txt'))

    }else{
      results_summary[m, 'p_1']        <- NA
      results_summary[m, 'train_rmse'] <- NA
      results_summary[m, 'train_AIC']  <- NA
      results_summary[m, 'train_BIC']  <- NA
      results_summary[m, 'test_rmse']  <- NA

    }

  }

  write.csv(results_summary, paste0('SFS_regr_',data_files[j],'.csv'),
            row.names = FALSE)

}


# 2. Learning interactions via hierarchical group-lasso regularization ----
for(j in 1:length(data_files)){

  # Initialise the all_result data.frame to store the result
  m <- 0
  data_name  <- data_files[j]
  all_result <- data.frame()
```

```r
if( data_name == 'BostonHousing'){

  data  <- read.csv('BostonHousing_train.csv')
  data  <- removeConstantFeatures(data)
  Y     <- data$medv
  X     <- data[, c(1:ncol(data)-1)]

  test_data <- read.csv('BostonHousing_test.csv')
  test_Y    <- test_data$medv
  test_X    <- test_data[, c(1:ncol(test_data)-1)]

}else if( data_name == 'Ailerons'){

  data  <- read.csv('Ailerons_train.csv')
  data  <- removeConstantFeatures(data)
  Y     <- data$V41
  X     <- data[, c(1:ncol(data)-1)]

  test_data <- read.csv('Ailerons_test.csv')
  test_Y    <- test_data$V41
  test_X    <- test_data[, c(1:ncol(test_data)-1)]


}else if( data_name == 'Elevators'){

  data  <- read.csv('Elevators_train.csv')
  data  <- removeConstantFeatures(data)
  Y     <- data$V19
  X     <- data[, c(1:ncol(data)-1)]

  test_data <- read.csv('Elevators_test.csv')
  test_Y    <- test_data$V19
  test_X    <- test_data[, c(1:ncol(test_data)-1)]


}else{
  stop('No dataset found')
}

# Create a vector of random seeds for reproducibility
# Note: Boston Housing has more random seeds as its data size is small
if( data_name == 'BostonHousing'){
  seed_vector <- c(1, 1010, 4, 781, 9990, 9999, 81, 46, 67, 2, 3, 1017)
}else{
  seed_vector <- c(1, 4, 781)
}


for(i in 1:length(seed_vector)){

  m <- m + 1

  seed.number <- seed_vector[i]
```

```r
  set.seed( seed.number )

  startTime <- Sys.time()
  fit       <- glinternet.cv(X, Y, numLevels = rep(1, ncol(X)),
                             family = 'gaussian', nFolds = numberCV[j],
                             numCores = 1, verbose = TRUE)
  endTime   <- Sys.time()


  # Obtain main effects and interaction
  coeff <- coef(fit)
  p_0   <- length(coeff$mainEffects$cat) + length(coeff$mainEffects$cont)
  p_1   <- length(coeff$interactions$catcat) +
    length(coeff$interactions$contcat) +
    length(coeff$interactions$contcont)

  k <- p_0 + p_1

  # calculate RMSE
  prediction <- data.frame(truth = Y, pred = fit$fitted)
  train_rmse <- mean((prediction$truth-prediction$pred)^2)^0.5

  # predict on test data
  test_prediction <- data.frame(truth = test_Y, pred = predict(fit, test_X))
  test_rmse <- mean((test_prediction$truth-test_prediction$pred)^2)^0.5

  # Export the granular info as JSON
  result <- list( colnames = colnames(X),
                  mainEffects  = coeff$mainEffects,
                  interactions = coeff$interactions,

                  train_rmse  = train_rmse,
                  test_rmse  = test_rmse,

                  lambda = fit$lambdaHat,
                  runtime = as.numeric(endTime - startTime ))

  write(toJSON(result, auto_unbox = TRUE, pretty = TRUE, factor = 'string'),
        paste0('glinternet_', data_name, '_', seed.number, '.txt'))

  # Append the all result summary
  all_result[m, 'dataset']   <- data_files[j]
  all_result[m, 'seed']      <- seed.number
  all_result[m, 'lambda']    <- result$lambda

  all_result[m, 'p_0']       <- p_0
  all_result[m, 'p_1']       <- p_1
  all_result[m, 'train_rmse']<- train_rmse
  all_result[m, 'test_rmse'] <- test_rmse

}
# Save the result summary
write.csv(all_result,
```

```r
            paste0('glinternet_regr_',data_files[j],'.csv'),
            row.names = FALSE)
}

# 3. Learning interactions using SP-FSR for linear regresssion task ----
for(j in 1:length(data_files)){

  # Initialise the all_result data.frame to store the result
  m <- 0
  results_summary   <- data.frame()

  data_name   <- data_files[j]
  target      <- targets[j]

  try(
    {
      data        <- read.csv(paste0(data_files[j], '_train.csv'))
      test_data   <- read.csv(paste0(data_files[j],'_test.csv'))
    }
  )

  task <- makeRegrTask(data = data, target = target)

  # Create a vector of random seeds for reproducibility
  # Note: Boston Housing has more random seeds as its data size is small
  if( data_name == 'BostonHousing'){
    seed_vector <- c(1, 1010, 4, 781, 9990, 9999, 81, 46, 67, 2, 3, 1017)
  }else{
    seed_vector <- c(1, 4, 781)
  }


  for(i in 1:length(seed_vector)){

    seed.number <- seed_vector[i]
    set.seed(seed.number)

    # Auto feature selection of main effects ----
    spsaMod <- spFeatureSelection(  task = task,
                                    wrapper = wrapper,
                                    measure = mlr::rmse,
                                    num.features.selected = 0,
                                    norm.method = NULL,
                                    cv.stratify = FALSE,
                                    num.cv.folds = numberCV[j])

    # Store the result
    m <- m + 1
    results_importance            <- list()
    est_coeff                     <- list()

    results_summary[m, 'dataset']     <- data_name
    results_summary[m, 'seed']        <- seed.number
```

```r
results_summary[m, 'p_0']         <- length(spsaMod$features)
results_summary[m, 'mean_0']      <- spsaMod$best.value
results_summary[m, 'std_0']       <- spsaMod$best.std
results_summary[m, 'runtime_0']   <- spsaMod$run.time

k <- 1
results_importance[[k]]   <- getImportance(spsaMod)
features.to.keep <- as.character(results_importance[[k]]$features)
target           <- task$task.desc$target

fittedTask     <- makeRegrTask(data[, c(features.to.keep, target)],
                               target = target, id = 'subset')

fittedMod      <- train(wrapper, fittedTask)
pred           <- predict(fittedMod, fittedTask)
fittedMod      <- fittedMod$learner.model
est_coeff[[k]] <- data.frame(coefficient = fittedMod$coefficients)


# Select interactions ----
sub_task  <- modifiedMakeRegrTask(data = data[, c(features.to.keep, target)],
                                  target = target, order = 2L)

sub_spsaMod <- spFeatureSelection(  task = sub_task,
                                    wrapper = wrapper,
                                    measure = mlr::rmse,
                                    num.features.selected = 0,
                                    norm.method = NULL,
                                    features.to.keep = features.to.keep,
                                    cv.stratify = FALSE,
                                    num.cv.folds = numberCV[j])

k <- k + 1
results_summary[m, 'p_1']       <- length(sub_spsaMod$features)
results_summary[m, 'mean_1']    <- sub_spsaMod$best.value
results_summary[m, 'std_1']     <- sub_spsaMod$best.std
results_summary[m, 'runtime_1'] <- sub_spsaMod$run.time
results_importance[[k]]         <- getImportance(sub_spsaMod)

new_features   <- as.character(results_importance[[k]]$features)
sub_fittedtask <- makeRegrTask(sub_task$env$data[, c(new_features, target)],
                               target = target, id = 'subset')

sub_fittedMod  <- train(wrapper, sub_fittedtask)
sub_pred       <- predict(sub_fittedMod, sub_fittedtask)

est_coeff[[k]]  <- data.frame(coefficient = sub_fittedMod$learner.model$coefficients)

results_summary[m, 'train_AIC']     <- AIC( sub_fittedMod$learner.model)
results_summary[m, 'train_BIC']     <- BIC( sub_fittedMod$learner.model )
results_summary[m, 'train_rmse']    <- mlr::performance(sub_pred, mlr::rmse)
```

```r
    # Predict on the test data
    sub_test_task <- modifiedMakeRegrTask(data =
                                            test_data[, c(features.to.keep, target)],
                                          target = target, order = 2L,
                                          remove.constant = FALSE)

    sub_test_task <- makeRegrTask(data =
                                    sub_test_task$env$data[, c(new_features, target)],
                                  target = target, id = 'test_subset')

    sub_pred_test  <- predict(sub_fittedMod, sub_test_task)

    results_summary[m, 'test_rmse']     <- mlr::performance( sub_pred_test, mlr::rmse)

    # Prediction w/o interaction terms
    test_task        <- makeRegrTask(data = test_data, target = target)
    test_pred_noint  <- predict(spsaMod$best.model, test_task)
    train_pred_noint <- predict(spsaMod$best.model, task)

    results_summary[m, 'test_rmse']       <- mlr::performance( sub_pred_test, mlr::rmse)
    results_summary[m, 'test_rmse_noint'] <- mlr::performance(test_pred_noint, mlr::rmse)
    results_summary[m, 'train_rmse_noint']<- mlr::performance(train_pred_noint, mlr::rmse)

    final_result   <- list(features = new_features,
                          est_coeff = est_coeff,
                          results_importance,
                          result = results_summary[m, ])

    write(toJSON(final_result, auto_unbox = TRUE, pretty = TRUE, factor = 'string'),
          paste0('spfsr_',data_name,'_',seed.number,'.txt'))
  }

  write.csv(results_summary,
            paste0('spfsr_regr_',data_files[j],'.csv'),
            row.names = FALSE)

}
# 4. Learning interactions using GA for linear regresssion task ----
# Define GA control
GActrl <-  makeFeatSelControlGA(maxit = 100)

for(j in 1:length(data_files)){
  # Initialise the all_result data.frame to store the result
  m <- 0
  results_summary   <- data.frame()
  data_name   <- data_files[j]

  try(
    {
      data        <- read.csv(paste0(data_files[j], '_train.csv'))
      test_data   <- read.csv(paste0(data_files[j],'_test.csv'))
    }
  )
```

```r
task <- makeRegrTask(data = data, target = targets[j])

# Create a vector of random seeds for reproducibility
# Note: Boston Housing has more random seeds as its data size is small
if( data_name == 'BostonHousing'){
  seed_vector <- c(1, 1010, 4, 781, 9990, 9999, 81, 46, 67, 2, 3, 1017)
}else{
  seed_vector <- c(1, 4, 781)
}

for(i in 1:length(seed_vector)){

  seed.number <- seed_vector[i]
  set.seed(seed.number)

  m <- m + 1

  # Resampling
  rdesc       <- makeResampleDesc(method = 'CV' , iters = numberCV[j])

  # Extract main effects
  main_sfeats <- selectFeatures(wrapper, task = task, resampling = rdesc,
                                control = GActrl, show.info = TRUE,
                                measures = mlr::rmse )

  # Obtain the performance with main effects only
  reduced_main_task <- makeRegrTask(id = 'reduced main',
                                    data = data[, c(main_sfeats$x , targets[j])],
                                    target = targets[j])

  # Add interactions
  second_task  <- modifiedMakeRegrTask(id = 'interaction',
                                       data = data[, c(main_sfeats$x , targets[j])],
                                       target = targets[j], order = 2L)

  second_sfeats <- selectFeatures(wrapper, task = second_task,
                                  resampling = rdesc,
                                  control = GActrl, show.info = TRUE,
                                  measures = mlr::rmse )

  results_summary[m, 'dataset']      <- data_name
  results_summary[m, 'seed_number']  <- seed.number
  results_summary[m, 'p_0']          <- length(main_sfeats$x)

  signif_terms     <- as.character(unique(c(main_sfeats$x , second_sfeats$x)))
  results_summary[m, 'p_1'] <- length(signif_terms) - length(main_sfeats$x)

  # Add the omitted main effects
  third_task <- makeRegrTask(id = 'strong hierarchy',
                             data = second_task$env$data[, c(signif_terms, targets[j])],
                             target = targets[j])

  constrainedMod   <- train( wrapper, third_task )
```

```
    constrainedPred  <- predict( constrainedMod, third_task)


    results_summary[m, 'train_rmse'] <- performance( constrainedPred , measure = mlr::rmse)
    results_summary[m, 'train_AIC'] <- AIC( constrainedMod$learner.model)
    results_summary[m, 'train_BIC'] <- BIC( constrainedMod$learner.model)

    sub_test_task  <- modifiedMakeRegrTask(data = test_data, target = targets[j],
                                           order = 2L, remove.constant = FALSE)

    sub_test_task  <- makeRegrTask(sub_test_task$env$data[, c(signif_terms, targets[j])],
                                   target = targets[j], id = 'test_subset')

    sub_pred_test         <- predict(constrainedMod , sub_test_task)
    results_summary[m, 'test_rmse'] <- performance( sub_pred_test, measure = mlr::rmse)


    output <- list(coeff  = constrainedMod$learner.model$coefficients,
                   result = results_summary[m, ],
                   features = signif_terms)

    write(toJSON(output, auto_unbox = TRUE, pretty = TRUE, factor = 'string'),
          paste0('GA_',data_name,'_',seed.number,'.txt'))

  }

  write.csv(results_summary,
            paste0('GA_regr_',data_files[j],'.csv'),
            row.names = FALSE)

}
```

R Codes for Classification Tasks

Baseline Learner for Classification

```
# Baseline learners without interactions: classification tasks ----
# Compare baseline learners with other methods
# 0, Load packages ----
library(mlr)
library(tidyverse)

# 1. Run baseline learners ----
# Initialise the data frame to store results,
# Prepare data sets and target continuous features
result      <- data.frame()

data_files  <- c('Sonar', 'ionosphere')
targets     <- c('Class', 'class')

# Common learner: logistic regression
wrapper <- makeLearner('classif.logreg', predict.type = 'prob')
```

```r
# 2. Loop for each dataset ----
for(j in 1:length(data_files) ){

  train_data  <- read.csv(paste0(data_files[j], '_train.csv'))
  test_data   <- read.csv(paste0(data_files[j],'_test.csv'))

  train_task  <- makeClassifTask(data = train_data, target = targets[j])
  test_task   <- makeClassifTask(data = test_data, target = targets[j])

  baseModel   <- train(wrapper, train_task)
  train_pred  <- predict(baseModel, train_task)
  test_pred   <- predict(baseModel, test_task)

  result[j, 'Data'] <- data_files[j]
  result[j, 'AIC']  <- AIC(baseModel$learner.model)
  result[j, 'BIC']  <- BIC(baseModel$learner.model)
  result[j, 'Train_logLoss'] <- mlr::performance(train_pred, mlr::logloss)
  result[j, 'Test_logLoss']  <- mlr::performance(test_pred, mlr::logloss)
  result[j, 'Train_auc']     <- mlr::performance(train_pred, mlr::auc)
  result[j, 'Test_auc']      <- mlr::performance(test_pred, mlr::auc)
}


# Save the result as a csv file
write.csv(result, 'baselearner_classif.csv', row.names = FALSE)
```

Modified Make Classification Task Function

```r
#' @description Generating pairwise interaction terms of a data
#'               and pass in to a task
#' @param data A data frame containing the features and target variable(s)
#' @param target Name of the target variable
#' @param order Interaction order. The allowed values are 1, 2, and 3.
#'              Default is 2L

modifiedMakeClassifTask <- function(data, target, order = 2,
                                    remove.constant = FALSE,...){

  # Argument check on "order"
  # It is not necessary to check on "data" or "target" as they will
  # be handled by makeClassifTask function
  if( !order %in% c(1, 2, 3) | !inherits(order, 'integer')){
    stop('order must be 1, 2, or 3.')
  }

  # create the classification task

  task   <- mlr::makeClassifTask(data = data, target = target, ...)

  if( order %in% c(2, 3)){
    # extract target and descriptive features

    y   <- task$env$data[, target]
```

```r
    if( order == 2){
      fml <- formula(paste0(target, "~.*."))
    }else{
      fml <- formula(paste0(target, "~.^3"))
    }

    X   <- data.frame(model.matrix(fml, task$env$data))

    # Remove constant terms
    if(remove.constant){
      X   <- mlr::removeConstantFeatures(X)
    }

    # bind the data
    newdata <- cbind(X,y)
    colnames(newdata)[ncol(newdata)] <- target

    # prompt warning message if p > n
    p <- ncol(X)
    if( p > nrow(X)){
      warning('More features than number of observations.')
    }

    # reconfigure the task
    task   <- mlr::makeClassifTask(data = newdata, target = target, ...)
  }

  return(task)

}
```

Automatic Feature Selection Script for Classification Tasks

```r
# 0. Preliminary ----
# Load the relevant packages
library(mlr)
library(spFSR)
library(jsonlite)
library(glinternet)
library(pROC)
library(MASS)
library(ROCR)

source('modifiedMakeClassifTask.R')

# Create a vector of random seeds for reproducibilty
seed_vector <- c(1, 1010, 4, 781, 9990, 9999, 81, 46, 67, 2, 3, 1017)
measure     <- mlr::auc

# Configuration of common task, wrapper and control
wrapper  <- makeLearner('classif.logreg', predict.type = 'prob')
rdesc    <- makeResampleDesc(method = 'CV', stratify = TRUE, iters = 5)

# 1. Learning interactions with SFFS ----
```

```
ctrl     <- makeFeatSelControlSequential(method = 'sffs',
                                         alpha = 0.001,
                                         beta = 0.0005)

m <- 0
result   <- data.frame()
datasets <- c('ionosphere', 'Sonar')

for( j in 1:length(datasets)){

  data_name <- datasets[j]
  if( data_name == 'ionosphere'){

    data      <- read.csv('ionosphere_train.csv')
    target    <- 'class'
    test_data <- read.csv('ionosphere_test.csv')

  }else if( data_name == 'Sonar'){

    data      <- read.csv('Sonar_train.csv')
    target    <- 'Class'
    test_data <- read.csv('Sonar_test.csv')

  }else{
    stop('No dataset found')
  }

  data  <- removeConstantFeatures(data)
  data  <- na.omit(data)

  test_data <- removeConstantFeatures(test_data)
  task      <- makeClassifTask(data = data, target = target)

  for(i in 1:length(seed_vector)){

    seed.number <- seed_vector[i]
    set.seed(seed.number)

    m <- m + 1

    # Extract main effects
    main_sfeats   <- selectFeatures(wrapper, task = task, resampling = rdesc,
                                    control = ctrl, show.info = TRUE, measures = measure )


    # Obtain the performance with main effects only
    reduced_main_task <- makeClassifTask(id = 'reduced main',
                                         data = data[, c(main_sfeats$x , target)],
                                         target = target)


    # Add interactions
    second_task <- modifiedMakeClassifTask(id = 'interaction',
```

```r
                                              data = data[, c(main_sfeats$x , target)],
                                              target = target, order = 2L)

    second_sfeats <- selectFeatures(wrapper, task = second_task, resampling = rdesc,
                                   control = ctrl, show.info = TRUE, measures = measure )

    result[m, 'dataset']      <- data_name
    result[m, 'seed_number']  <- seed.number
    result[m, 'p_0']          <- length(main_sfeats$x)

    signif_terms     <- as.character(unique(c(main_sfeats$x , second_sfeats$x)))
    result[m, 'p_1'] <- length(signif_terms) - length(main_sfeats$x)

    # Add the omitted main effects
    third_task <- makeClassifTask( id = 'strong hierarchy',
                                   data = second_task$env$data[, c(signif_terms, target)],
                                   target = target)

    constrainedMod   <- train( wrapper, third_task )
    constrainedPred  <- predict( constrainedMod, third_task)


    result[m, 'train_auc'] <- performance( constrainedPred , measure = measure)
    result[m, 'train_AIC'] <- AIC( constrainedMod$learner.model)
    result[m, 'train_BIC'] <- BIC( constrainedMod$learner.model)

    sub_test_task  <- modifiedMakeClassifTask(data = test_data, target = target, order = 2L)

    sub_test_task  <- makeClassifTask(sub_test_task$env$data[, c(signif_terms, target)],
                                     target = target, id = 'test_subset')

    sub_pred_test         <- predict(constrainedMod , sub_test_task)
    result[m, 'test_auc'] <- performance( sub_pred_test, measure)

    output <- list(coeff   = constrainedMod$learner.model$coefficients,
                   result = result[m, ],
                   features = signif_terms)

    write(toJSON(output, auto_unbox = TRUE, pretty = TRUE, factor = 'string'),
          paste0('SFFS_',data_name,'_',seed.number,'.txt'))
  }

}

write.csv(result, 'SFFS_results.csv', row.names = FALSE)



# 2. Learning interactions with two-step SP-FSR ----
# Reset the results summary
m <- 0
results_summary   <- data.frame()
```

```
for(j in 1:length(datasets)){

  data_name  <- datasets[j]

  if( data_name == 'ionosphere'){

    data      <- read.csv('ionosphere_train.csv')
    target    <- 'class'
    test_data <- read.csv('ionosphere_test.csv')
    data      <- removeConstantFeatures(data)
    test_data <- removeConstantFeatures(test_data)

  }else if( data_name == 'Sonar'){

    data      <- read.csv('Sonar_train.csv')
    target    <- 'Class'
    test_data <- read.csv('Sonar_test.csv')
    data      <- removeConstantFeatures(data)
    test_data <- removeConstantFeatures(test_data)

  }else{
    stop('No dataset found')
  }

  task      <- makeClassifTask(data = data, target = target)


  for(i in 1:length(seed_vector)){

    seed.number <- seed_vector[i]
    set.seed(seed.number)

    # Auto feature selection of main effects ----
    spsaMod <- spFeatureSelection(  task = task,
                                    wrapper = wrapper,
                                    measure = measure,
                                    num.features.selected = 0,
                                    norm.method = NULL)

    # Store the result
    m <- m + 1
    results_importance            <- list()
    est_coeff                     <- list()

    results_summary[m, 'dataset']     <- data_name
    results_summary[m, 'seed']        <- seed.number
    results_summary[m, 'p_0']         <- length(spsaMod$features)
    results_summary[m, 'mean_0']      <- spsaMod$best.value
    results_summary[m, 'std_0']       <- spsaMod$best.std
    results_summary[m, 'runtime_0']   <- spsaMod$run.time

    k <- 1
    results_importance[[k]]   <- getImportance(spsaMod)
```

```r
features.to.keep <- as.character(results_importance[[k]]$features)
target           <- task$task.desc$target

fittedTask     <- makeClassifTask(data[, c(features.to.keep, target)],
                                  target = target, id = 'subset')

fittedMod      <- train(wrapper, fittedTask)
pred           <- predict(fittedMod, fittedTask)
fittedMod      <- fittedMod$learner.model
est_coeff[[k]] <- data.frame(coefficient = fittedMod$coefficients)


# Select interactions ----
sub_task  <- modifiedMakeClassifTask(data = data[, c(features.to.keep, target)],
                                     target = target, order = 2L)

sub_spsaMod <- spFeatureSelection(  task = sub_task,
                                    wrapper = wrapper,
                                    measure = measure,
                                    num.features.selected = 0,
                                    norm.method = NULL,
                                    features.to.keep = features.to.keep)

k <- k + 1
results_summary[m, 'p_1']       <- length(sub_spsaMod$features)
results_summary[m, 'mean_1']    <- sub_spsaMod$best.value
results_summary[m, 'std_1']     <- sub_spsaMod$best.std
results_summary[m, 'runtime_1'] <- sub_spsaMod$run.time
results_importance[[k]]         <- getImportance(sub_spsaMod)

new_features   <- as.character(results_importance[[k]]$features)
sub_fittedtask <- makeClassifTask(sub_task$env$data[, c(new_features, target)],
                                  target = target, id = 'subset')
sub_fittedMod  <- train(wrapper, sub_fittedtask)
sub_pred       <- predict(sub_fittedMod, sub_fittedtask)

est_coeff[[k]]  <- data.frame(coefficient = sub_fittedMod$learner.model$coefficients)

results_summary[m, 'train_AIC']     <- AIC( sub_fittedMod$learner.model)
results_summary[m, 'train_BIC']     <- BIC( sub_fittedMod$learner.model )
results_summary[m, 'train_auc']     <- mlr::performance(sub_pred, mlr::auc)
results_summary[m, 'train_logloss'] <- mlr::performance(sub_pred, mlr::logloss)


# Predict on the test data
sub_test_task  <- modifiedMakeClassifTask(data =
                                            test_data[, c(features.to.keep, target)],
                                          target = target, order = 2L)

sub_test_task  <- makeClassifTask(sub_test_task$env$data[, c(new_features, target)],
                                  target = target, id = 'test_subset')

sub_pred_test  <- predict(sub_fittedMod, sub_test_task)
```

```
    results_summary[m, 'test_auc']     <- mlr::performance( sub_pred_test, measure)
    results_summary[m, 'test_logloss'] <- mlr::performance( sub_pred_test, mlr::logloss)

    # Prediction w/o interaction terms
    test_task        <- makeClassifTask(data = test_data, target = target)
    test_pred_noint  <- predict(spsaMod$best.model, test_task)
    train_pred_noint <- predict(spsaMod$best.model, task)

    results_summary[m, 'test_auc']     <- mlr::performance( sub_pred_test, measure)
    results_summary[m, 'test_logloss'] <- mlr::performance( sub_pred_test, mlr::logloss)

    results_summary[m, 'test_auc_noint']     <- mlr::performance(test_pred_noint,
                                                                  mlr::auc)
    results_summary[m, 'test_logloss_noint'] <- mlr::performance(test_pred_noint,
                                                                  mlr::logloss)

    results_summary[m, 'train_auc_noint']     <- mlr::performance(train_pred_noint,
                                                                   mlr::auc)
    results_summary[m, 'train_logloss_noint'] <- mlr::performance(train_pred_noint,
                                                                   mlr::logloss)

    final_result   <- list(features = new_features,
                           est_coeff = est_coeff,
                           results_importance,
                           result = results_summary[m, ])

    write(toJSON(final_result, auto_unbox = TRUE, pretty = TRUE, factor = 'string'),
          paste0('spfsr_',data_name,'_',seed.number,'.txt'))
  }
}

write.csv(results_summary, 'spfsr_results.csv', row.names = FALSE)


# 3. Learning interactions with glinternet ----
# Reset the data frame
m <- 0
all_result <- data.frame()

for(j in 1:length(datasets)){

  data_name  <- datasets[j]

  if( data_name == 'ionosphere'){

    data <- read.csv('ionosphere_train.csv')
    Y    <- ifelse(data$class == 'b', 1, 0)
    data  <- removeConstantFeatures(data)
    X     <- data[, c(1:ncol(data)-1)]

    test_data <- read.csv('ionosphere_test.csv')
    test_data <- removeConstantFeatures(test_data)
    test_Y    <- ifelse(test_data$class == 'b', 1, 0)
```

```r
  test_X    <- test_data[, c(1:ncol(test_data)-1)]

}else if( data_name == 'Sonar'){

  data <- read.csv('Sonar_train.csv')
  Y    <- ifelse(data$Class == 'R', 1, 0)
  data  <- removeConstantFeatures(data)
  X     <- data[, c(1:ncol(data)-1)]

  test_data <- read.csv('Sonar_test.csv')
  test_data <- removeConstantFeatures(test_data)
  test_Y    <- ifelse(test_data$Class == 'R', 1, 0)
  test_X    <- test_data[, c(1:ncol(test_data)-1)]

}else{
  stop('No dataset found')
}

for(i in 1:length(seed_vector)){

  m <- m + 1

  seed.number <- seed_vector[i]
  set.seed( seed.number )


  # hierarchical group-lasso regularization (CV = 10)
  startTime <- Sys.time()
  fit       <- glinternet.cv(X, Y, numLevels = rep(1, ncol(X)),
                             family = 'binomial', nFolds = 5)
  endTime   <- Sys.time()

  # Obtain main effects and interaction
  coeff <- coef(fit)
  p_0 <- length(coeff$mainEffects$cat) +
    length(coeff$mainEffects$cont)
  p_1 <- length(coeff$interactions$catcat) +
    length(coeff$interactions$contcat) +
    length(coeff$interactions$contcont)

  k <- p_0 + p_1

  # calculate AIC and BIC
  prediction <- data.frame(truth = Y, pred = fit$fitted)
  logLik     <- sum(prediction$truth*log(prediction$pred) + (1-prediction$truth)*log(1-prediction$pred
  AIC        <- -2*logLik + 2*k
  BIC        <- -2*logLik + log(nrow(X))*k

  # predict on test data
  test_prediction <- data.frame(truth = test_Y,
                                pred = predict(fit, test_X))

  test_logLik    <- sum(test_prediction$truth*log(test_prediction$pred) +
```

```r
                            (1-test_prediction$truth)*log(1-test_prediction$pred))

    test_AIC        <- -2*test_logLik + 2*k
    test_BIC        <- -2*test_logLik + log(nrow(test_X))*k


    # Store the result
    result <- list( colnames = colnames(X),
                    mainEffects  = coeff$mainEffects,
                    interactions = coeff$interactions,

                    train_auc  = auc(Y, fit$fitted),
                    AIC = AIC,
                    BIC = BIC,

                    test_auc = auc(test_prediction$truth, test_prediction$pred),
                    test_AIC = test_AIC,
                    test_BIC = test_BIC,


                    train_logloss = -logLik/nrow(data),
                    test_logloss  = -test_logLik/nrow(test_data),
                    lambda = fit$lambdaHat,
                    runtime = as.numeric(endTime - startTime ))

    write(toJSON(result, auto_unbox = TRUE, pretty = TRUE, factor = 'string'),
          paste0('glinternet_', data_name, '_', seed.number, '.txt'))

    # Store the all result summary

    all_result[m, 'dataset']   <- datasets[j]
    all_result[m, 'seed']      <- seed.number
    all_result[m, 'lambda']    <- result$lambda
    all_result[m, 'train_auc'] <- result$train_auc
    all_result[m, 'train_AIC'] <- result$AIC
    all_result[m, 'train_BIC'] <- result$BIC
    all_result[m, 'train_logloss'] <- result$train_logloss
    all_result[m, 'p_0']       <- p_0
    all_result[m, 'p_1']       <- p_1
    all_result[m, 'test_auc']  <- result$test_auc
    all_result[m, 'test_AIC']  <- result$test_AIC
    all_result[m, 'test_BIC']  <- result$test_BIC
    all_result[m, 'test_logloss'] <- result$test_logloss

  }

}

# Save the result summary
write.csv(all_result, 'glinternet_all_result.csv', row.names = FALSE)

# 4. Learning interactions with GA ----
m         <- 0
```

```r
result   <- data.frame()
ctrl     <- makeFeatSelControlGA( maxit = 20)

for( j in 1:length(datasets)){

  data_name <- datasets[j]
  if( data_name == 'ionosphere'){

    data      <- read.csv('ionosphere_train.csv')
    target    <- 'class'
    test_data <- read.csv('ionosphere_test.csv')

  }else if( data_name == 'Sonar'){

    data      <- read.csv('Sonar_train.csv')
    target    <- 'Class'
    test_data <- read.csv('Sonar_test.csv')

  }else{
    stop('No dataset found')
  }

  data  <- removeConstantFeatures(data)
  data  <- na.omit(data)

  test_data <- removeConstantFeatures(test_data)
  task      <- makeClassifTask(data = data, target = target)

  for(i in 1:length(seed_vector)){

    seed.number <- seed_vector[i]
    set.seed(seed.number)

    m <- m + 1

    # Extract main effects
    main_sfeats   <- selectFeatures(wrapper, task = task, resampling = rdesc,
                                    control = ctrl, show.info = TRUE, measures = measure )


    # Obtain the performance with main effects only
    reduced_main_task <- makeClassifTask(id = 'reduced main',
                                         data = data[, c(main_sfeats$x , target)],
                                         target = target)


    # Add interactions
    second_task <- modifiedMakeClassifTask(id = 'interaction',
                                           data = data[, c(main_sfeats$x , target)],
                                           target = target, order = 2L)

    second_sfeats <- selectFeatures(wrapper, task = second_task, resampling = rdesc,
                                    control = ctrl, show.info = TRUE, measures = measure )
```

```r
    result[m, 'dataset']      <- data_name
    result[m, 'seed_number']  <- seed.number
    result[m, 'p_0']          <- length(main_sfeats$x)

    signif_terms     <- as.character(unique(c(main_sfeats$x , second_sfeats$x)))
    result[m, 'p_1'] <- length(signif_terms) - length(main_sfeats$x)

    # Add the omitted main effects
    third_task <- makeClassifTask( id = 'strong hierarchy',
                                  data = second_task$env$data[, c(signif_terms, target)],
                                  target = target)

    constrainedMod   <- train( wrapper, third_task )
    constrainedPred  <- predict( constrainedMod, third_task)


    result[m, 'train_auc'] <- performance( constrainedPred , measure = measure)
    result[m, 'train_AIC'] <- AIC( constrainedMod$learner.model)
    result[m, 'train_BIC'] <- BIC( constrainedMod$learner.model)

    sub_test_task  <- modifiedMakeClassifTask(data = test_data, target = target, order = 2L)

    sub_test_task  <- makeClassifTask(sub_test_task$env$data[, c(signif_terms, target)],
                                  target = target, id = 'test_subset')

    sub_pred_test         <- predict(constrainedMod , sub_test_task)
    result[m, 'test_auc'] <- performance( sub_pred_test, measure)

    output <- list(coeff  = constrainedMod$learner.model$coefficients,
                   result = result[m, ],
                   features = signif_terms)

    write(toJSON(output, auto_unbox = TRUE, pretty = TRUE, factor = 'string'),
          paste0('GA_',data_name,'_',seed.number,'.txt'))
  }

}

write.csv(result, 'GA_result.csv', row.names = FALSE)
```