

synthetic datasets

Warning: package 'bartMachine' was built under R version 4.3.3

Warning: package 'randomForest' was built under R version 4.3.3

Warning: package 'missForest' was built under R version 4.3.3

Warning: package 'dbarts' was built under R version 4.3.3

Warning: package 'BART' was built under R version 4.3.3

Warning: package 'bench' was built under R version 4.3.3

Warning: package 'ggplot2' was built under R version 4.3.3

create dataset

```
linear_dgp_fun <- function(n_train, n_test, p, beta, noise_sd) {  
  n <- n_train + n_test  
  X <- matrix(rnorm(n * p), nrow = n, ncol = p)  
  y <- X %*% beta + rnorm(n, sd = noise_sd)  
  data_list <- list(  
    X_train = X[1:n_train, , drop = FALSE],  
    y_train = y[1:n_train],  
    X_test = X[(n_train + 1):n, , drop = FALSE],  
    y_test = y[(n_train + 1):n]  
  )  
  return(data_list)  
}  
linear_dgp <- create_dgp(
```

```

.dgp_fun = linear_dgp_fun, .name = "Linear DGP",
# additional named parameters to pass to .dgp_fun()
n_train = 350, n_test = 120, p = 4, beta = c(1,2,1.5,3), noise_sd = 1
)

dataset_dgp_fun <- function(datasetname){

  address <- "C:/Users/pyk/Desktop/nus/RA/project/imodels-data-master/data_cleaned/"
  file <- paste0(datasetname, ".csv")
  file_path <- paste0(address, file)
  df <- read.csv(file_path)
  x <- df[, -ncol(df)]
  y <- df[, ncol(df)]

  train_indices <- createDataPartition(y, p = 0.8, list = FALSE)

  data_list <- list(
    X_train <- x[train_indices, ],
    y_train <- y[train_indices],
    X_test <- x[-train_indices, ],
    y_test <- y[-train_indices]
  )
  return(data_list)
}
dataset_dgp <- create_dgp(.dgp_fun = dataset_dgp_fun, .name = 'heart',
                          datasetname = "heart")

```

```
linear_dgp
```

DGP Name: Linear DGP

Function: function (n_train, n_test, p, beta, noise_sd)

Parameters: List of 5

\$ n_train : num 350

\$ n_test : num 120

\$ p : num 4

\$ beta : num [1:4] 1 2 1.5 3

\$ noise_sd: num 1

build BART model

```
BART_fun <- function(X_train, y_train, X_test, y_test, df,k,q) {
  train_X <- data.frame(X_train)
  test_X <- data.frame(X_test)
  t <- bench::mark(fit <- wbart(x.train = train_X,
                                y.train = y_train,
                                x.test = test_X,
                                k = k,
                                sigdf = df,
                                sigquant = q
                                ))

  time <- mean(t$time[[1]])
  predictions <- colMeans(fit$yhat.test)
  mse_score <- mean((y_test - predictions)^2)

  return(list(time = time, mse=mse_score))
}

dbarts_fun <- function(X_train, y_train, X_test, y_test, df,k,q){
  train_X <- data.frame(X_train)
  test_X <- data.frame(X_test)
  t <- bench::mark(bart_model <- bart(x.train = train_X,
                                       y.train = y_train,
                                       x.test = test_X,
                                       k = k,
                                       sigdf = df,
                                       sigquant = q))

  time <- mean(t$time[[1]])
  predictions <- colMeans(bart_model$yhat.test)
  mse_score <- mean((y_test - predictions)^2)

  return(list(time = time, mse=mse_score))
}

bartMachine_fun <- function(X_train, y_train, X_test,y_test,df,k,q){
  train_X <- data.frame(X_train)
  test_X <- data.frame(X_test)
  t <- bench::mark(bart_model <- bartMachine(
    X = train_X,
    y = y_train,
    k = k,
```

```

        nu = df,
        q=q))
    # The value of calculating the time required for modeling
    time <- mean(t$time[[1]])
    predictions <- predict(bart_model,test_X,type = "prob")
    mse_score <- mean((y_test - predictions)^2)

    return(list(time = time, mse=mse_score))
}

SoftBart_fun<- function(X_train, y_train, X_test,y_test,num_trees,alpha,beta){
  train_X <- data.frame(X_train)
  test_X <- data.frame(X_test)
  t <- bench::mark({bart_model <- softbart(X = train_X, Y = y_train, X_test = test_X, hyper
    #print(t)
    time <- mean(t$time[[1]])
    predictions <- bart_model$y_hat_test_mean
    mse_score <- mean((y_test - predictions)^2)

    return(list(time = time, mse=mse_score))
  })

RF_fun <- function(X_train, y_train, X_test,y_test){
  train_X <- data.frame(X_train)
  test_X <- data.frame(X_test)
  t <- bench::mark({rf_model <- randomForest(x=train_X, y=y_train)})
  time <- mean(t$time[[1]])
  predictions <- predict(rf_model, test_X)
  mse_score <- mean((y_test - predictions)^2)
  return(list(time = time, mse=mse_score))
}

```

create evaluation

```

posterior_mse <- function(fit_results,truth_col,estimate_col){
  y_test = fit_results$truth_col
  pred = fit_results$estimate_col
  return(mean((y_test - pred)^2))
}

pred_err <- create_evaluator(

```

```

    .eval_fun = posterior_mse, .name = 'Posterior MSE',
    # additional named parameters to pass to .eval_fun()
    truth_col = "y_test", estimate_col = "predictions"
  )

BART <- create_method(
  .method_fun = BART_fun, .name = "BART",
  # additional named parameters to pass to .method_fun()
  k=2.5,q=0.95,df=4
)
dbarts <- create_method(.method_fun = dbarts_fun,.name = "dbarts",
                        k=2.5,q=0.95,df=4)
bartMachine <- create_method(.method_fun = bartMachine_fun,.name = "bartMachine",
                             k=2.5,q=0.95,df=4)
SoftBart <- create_method(.method_fun = SoftBart_fun,.name = "SoftBart",
                          num_trees=50,alpha=0.95,beta=2)
RF <- create_method(.method_fun = RF_fun,.name = "RandomForest")
# Create experiment
experiment <- create_experiment(name = "Test Experiment") %>%
  add_dgp(linear_dgp) %>%
  add_dgp(dataset_dgp) %>%
  add_method(dbarts) %>%
  add_method(BART) %>%
  add_method(bartMachine) %>%
  add_method(SoftBart) %>%
  add_method(RF)%>%
  add_evaluator(pred_err)

results <- run_experiment(experiment, n_reps = 4, save = TRUE)

```

Fitting Test Experiment...

Warning: Some expressions had a GC in every iteration; so filtering is disabled.

Warning: Some expressions had a GC in every iteration; so filtering is disabled.

Warning: Some expressions had a GC in every iteration; so filtering is disabled.

Warning in randomForest.default(x = train_X, y = y_train): The response has five or fewer unique values. Are you sure you want to do regression?
Warning in randomForest.default(x = train_X, y = y_train): The response has five or fewer unique values. Are you sure you want to do regression?
Warning in randomForest.default(x = train_X, y = y_train): The response has five or fewer unique values. Are you sure you want to do regression?
Warning in randomForest.default(x = train_X, y = y_train): The response has five or fewer unique values. Are you sure you want to do regression?
Warning in randomForest.default(x = train_X, y = y_train): The response has five or fewer unique values. Are you sure you want to do regression?

Warning: Some expressions had a GC in every iteration; so filtering is disabled.

Warning in randomForest.default(x = train_X, y = y_train): The response has five or fewer unique values. Are you sure you want to do regression?
Warning in randomForest.default(x = train_X, y = y_train): The response has five or fewer unique values. Are you sure you want to do regression?
Warning in randomForest.default(x = train_X, y = y_train): The response has five or fewer unique values. Are you sure you want to do regression?
Warning in randomForest.default(x = train_X, y = y_train): The response has five or fewer unique values. Are you sure you want to do regression?
Warning in randomForest.default(x = train_X, y = y_train): The response has five or fewer unique values. Are you sure you want to do regression?

Warning: Some expressions had a GC in every iteration; so filtering is disabled.

Warning in randomForest.default(x = train_X, y = y_train): The response has five or fewer unique values. Are you sure you want to do regression?
Warning in randomForest.default(x = train_X, y = y_train): The response has five or fewer unique values. Are you sure you want to do regression?
Warning in randomForest.default(x = train_X, y = y_train): The response has five or fewer unique values. Are you sure you want to do regression?
Warning in randomForest.default(x = train_X, y = y_train): The response has five or fewer unique values. Are you sure you want to do regression?

Warning: Some expressions had a GC in every iteration; so filtering is disabled.

```
Warning in randomForest.default(x = train_X, y = y_train): The response has
five or fewer unique values. Are you sure you want to do regression?
Warning in randomForest.default(x = train_X, y = y_train): The response has
five or fewer unique values. Are you sure you want to do regression?
Warning in randomForest.default(x = train_X, y = y_train): The response has
five or fewer unique values. Are you sure you want to do regression?
Warning in randomForest.default(x = train_X, y = y_train): The response has
five or fewer unique values. Are you sure you want to do regression?
```

```
Saving fit results...
```

```
Fit results saved | time taken: 0.087900 seconds
```

```
4 reps completed (totals: 4/4) | time taken: 11.333286 minutes
```

```
=====
```

```
Evaluating Test Experiment...
```

```
Warning: Unknown or uninitialised column: `truth_col`.
```

```
Warning: Unknown or uninitialised column: `estimate_col`.
```

```
Evaluation completed | time taken: 0.000297 minutes
```

```
Saving eval results...
```

```
Eval results saved | time taken: 0.077636 seconds
```

```
=====
```

```
No visualizers to visualize. Skipping visualization.
```

```
=====
```

```
# Render automated documentation and view results
#render_docs(experiment)
```

```
result <- results$fit_results
result
```

```
# A tibble: 40 x 5
```

	.rep	.dgp_name	.method_name	time	mse
	<chr>	<chr>	<chr>	<list>	<dbl>
1	1	Linear DGP	BART	<bench_tm [1]>	1.41
2	1	Linear DGP	RandomForest	<bench_tm [1]>	2.98
3	1	Linear DGP	SoftBart	<bench_tm [1]>	1.15
4	1	Linear DGP	bartMachine	<bench_tm [1]>	1.36
5	1	Linear DGP	dbarts	<bench_tm [1]>	1.42

```

6 1 heart BART <bench_tm [1]> 0.123
7 1 heart RandomForest <bench_tm [1]> 0.127
8 1 heart SoftBart <bench_tm [1]> 0.118
9 1 heart bartMachine <bench_tm [1]> 0.120
10 1 heart dbarts <bench_tm [1]> 1.52
# i 30 more rows

```

```
result$time_numeric <- as.numeric(result$time)
```

```
result$Resource <- paste(result$.dgp_name, result$.method_name, sep="_")
```

```

# Calculate MSE for each group
summary <- result %>%
  group_by(Resource) %>%
  summarise(
    Mean_MSE = mean(mse),
    Var_MSE = sd(mse),
    Mean_time = mean(time_numeric),
    Var_time = sd(time_numeric))

print(summary)

```

```
# A tibble: 10 x 5
```

	Resource	Mean_MSE	Var_MSE	Mean_time	Var_time
	<chr>	<dbl>	<dbl>	<dbl>	<dbl>
1	Linear DGP_BART	1.22	0.174	3.96	0.287
2	Linear DGP_RandomForest	2.87	0.186	0.208	0.0404
3	Linear DGP_SoftBart	1.09	0.110	40.8	2.46
4	Linear DGP_bartMachine	1.27	0.143	1.32	0.0737
5	Linear DGP_dbarts	1.16	0.210	0.922	0.186
6	heart_BART	0.130	0.0106	3.91	0.487
7	heart_RandomForest	0.141	0.0126	0.179	0.0191
8	heart_SoftBart	0.132	0.0124	26.8	1.04
9	heart_bartMachine	0.130	0.0102	1.07	0.0895
10	heart_dbarts	1.45	0.108	0.993	0.137