

python in R

python environment setting

```
use_condaenv("final")
bartpy <- import("bartpy2.sklearnmodel")
#time_py <- import("time")
numpy <- import("numpy")

# unnormalize function from [-0.5,0.5]
unnormalize_x <- function(y_train,y_new){
  x <- data.frame()
  y_min <- min(y_train)
  y_max <- max(y_train)
  for (i in 1:nrow(y_new)) {
    for (j in 1:ncol(y_new)) {
      x[i,j] <- (y_max-y_min)*(y_new[i,j]+0.5)+y_min
    }
  }
  return(x)
}
```

create dataset

```
linear_dgp_fun <- function(ratio,n, p, noise_sd) {
  set.seed(123)
  n_train <- n*ratio
  beta <- sample(1:100, p, replace = FALSE)

  #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()
  ratio = 0.8, n = 500, p = 4, 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")

```

build BART model

```

BART_fun <- function(X_train, y_train, X_test, y_test, df,k,q,nchain,budget) {
  train_X <- data.frame(X_train)
  test_X <- data.frame(X_test)
  time <- 0
  posterior <- data.frame()
  for (i in 1:nchain) {
    t <- bench::mark(fit <- wbart(x.train = train_X,
                                y.train = y_train,
                                x.test = test_X,
                                k = k,
                                sigdf = df,
                                sigquant = q,
                                ndpost = budget/nchain
                                ))
    time <- time+mean(t$time[[1]])
    posterior <- rbind(posterior, fit$yhat.test)
  }

  #predictions <- colMeans(fit$yhat.test)
  predictions <- colMeans(posterior)
  mse_score <- mean((y_test - predictions)^2)

  lower_bounds <- apply(posterior, 2, quantile, probs = 0.025)
  upper_bounds <- apply(posterior, 2, quantile, probs = 0.975)

  coverage <- mean(y_test >= lower_bounds & y_test <= upper_bounds)
  return(list(time = time, mse=mse_score,coverage = coverage))
}

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

  time <- mean(t$time[[1]])

```

```

predictions <- colMeans(bart_model$yhat.test)
mse_score <- mean((y_test - predictions)^2)

lower_bounds <- apply(bart_model$yhat.test, 2, quantile, probs = 0.025)
upper_bounds <- apply(bart_model$yhat.test, 2, quantile, probs = 0.975)

coverage <- mean(y_test >= lower_bounds & y_test <= upper_bounds)

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

bartMachine_fun <- function(X_train, y_train, X_test, y_test, df, k, q, nchain, budget){
  train_X <- data.frame(X_train)
  test_X <- data.frame(X_test)
  posterior <- data.frame()
  time <- 0
  CI <- matrix(0, nrow = nrow(test_X), ncol = 2)
  ndpost <- budget/nchain

  t <- bench::mark(bart_model <- bartMachine(
    X = train_X,
    y = y_train,
    k = k,
    nu = df,
    q=q,
    num_burn_in = 100,
    num_iterations_after_burn_in = ndpost))
  # The value of calculating the time required for modeling
  time <- time+mean(t$time[[1]])
  posterior <- rbind(posterior, predict(bart_model, test_X, type = "prob"))
  CI <- CI+calc_credible_intervals(bart_model, test_X)

  predictions <- colMeans(posterior)
  #predictions <- predict(bart_model, test_X, type = "prob")
  mse_score <- mean((y_test - predictions)^2)

  #CI <- calc_credible_intervals(bart_model, test_X)
  coverage <- mean(y_test >= CI[,1]/nchain & y_test <= CI[,2]/nchain)

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

```

```

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, hyperp

  time <- mean(t$time[[1]])
  predictions <- bart_model$y_hat_test_mean
  mse_score <- mean((y_test - predictions)^2)

  lower_bounds <- apply(bart_model$y_hat_test, 2, quantile, probs = 0.025)
  upper_bounds <- apply(bart_model$y_hat_test, 2, quantile, probs = 0.975)

  coverage <- mean(y_test >= lower_bounds & y_test <= upper_bounds)

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

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))
}

bartpy_fun <- function(X_train, y_train, X_test,y_test,nchain,budget){
  train_x <- numpy$array(X_train)
  train_y <- numpy$array(y_train)
  test_x <- numpy$array(X_test)
  test_y <- numpy$array(y_test)

  bart_model <- bartpy$SklearnModel(n_jobs=1,n_chains = as.integer(nchain),
                                     n_samples = as.integer(budget/nchain) )

  #start_time <- time_py$time()
  t <- bench::mark({yk <- bart_model$fit(train_x,train_y)})
  #time <- time_py$time-start_time
  time <- mean(t$time[[1]])
  predictions <- yk$predict(test_x)
  mse_score <- mean((test_y - predictions)^2)

```

```

## calculate coverage
extract <- yk$extract
model_samples <- extract[[1]][[1]]
a <- data.frame()
for (model in model_samples) {
  a <- rbind(a,model$predict(test_x))
}
a_new <- unnormailize_x(train_y,a)

lower_bounds <- apply(a_new, 2, quantile, probs = 0.025)
upper_bounds <- apply(a_new, 2, quantile, probs = 0.975)

coverage <- mean(test_y >= lower_bounds & test_y <= upper_bounds)

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

stochtree_fun <- function(X_train, y_train, X_test, y_test, q,nchain,budget){
  train_X <- data.frame(X_train)
  test_X <- data.frame(X_test)
  posterior <- matrix(0, nrow = nrow(test_X), ncol = 1)
  time <- 0
  for (i in 1:nchain) {
    t <- bench::mark(bart_model <- stochtree::bart(X_train = train_X,
                                                    y_train = y_train,
                                                    X_test = test_X,
                                                    q = q,
                                                    num_burnin = 100))

    time <- time+mean(t$time[[1]])
    posterior <- cbind(posterior,bart_model$y_hat_test)
  }
  posterior <- posterior[,-1]
  predictions <- rowMeans(posterior)
  mse_score <- mean((y_test - predictions)^2)

  lower_bounds <- apply(posterior, 1, quantile, probs = 0.025)
  upper_bounds <- apply(posterior, 1, quantile, probs = 0.975)

  coverage <- mean(y_test >= lower_bounds & y_test <= upper_bounds)

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

```

```

}
xgb_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({xgb_model <- xgboost(data=as.matrix(train_X), label =y_train,nrounds = 1000)})
  time <- mean(t$time[[1]])
  predictions <- predict(xgb_model, as.matrix(test_X))
  mse_score <- mean((y_test - predictions)^2)
  return(list(time = time, mse=mse_score))
}

```

create evaluation

```

plot_mse <- function(fit_results){
  fit_results$time_numeric <- as.numeric(fit_results$time)

# Calculate MSE for each group
summary <- fit_results %>%
  group_by(fit_results$.dgp_name, fit_results$.method_name,n,p,noise_sd,nchain) %>%
  summarise(
    Mean_MSE = mean(mse),
    Var_MSE = sd(mse),

    .groups = 'keep')
plt <- ggplot(summary, aes(x = `fit_results$.method_name`, y = Mean_MSE
  #fill = Category
)) +
  geom_bar(stat = "identity") +
  facet_grid(~n+p+noise_sd+nchain)+
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))+

  labs(y = "MSE", x = "method")
  facet_wrap(~ `fit_results$.dgp_name`)
return(plt)
}

plot_time <- function(fit_results){
  fit_results$time_numeric <- as.numeric(fit_results$time)

```

```

# Calculate MSE for each group
summary <- fit_results %>%
  group_by(fit_results$.dgp_name, fit_results$.method_name,n,p,noise_sd,nchain) %>%
  summarise(
    Mean_time = mean(time_numeric),
    Var_time = sd(time_numeric),
  )

plt <- ggplot(summary, aes(x = `fit_results$.method_name`, y = Mean_time
  #fill = Category
)) +
  geom_bar(stat = "identity") +

  facet_wrap(~n+p+noise_sd+nchain) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))+
  labs(y = "time", x = "method")
  facet_wrap(~ `fit_results$.dgp_name`)
return(plt)
}

plot_coverage <- function(fit_results){
  fit_results$time_numeric <- as.numeric(fit_results$time)

# Calculate MSE for each group
summary <- fit_results %>%
  group_by(fit_results$.dgp_name, fit_results$.method_name,n,p,noise_sd,nchain) %>%
  summarise(

    Mean_coverage=mean(coverage),
    SD_coverage = sd(coverage))

plt <- ggplot(summary, aes(x = `fit_results$.method_name`, y = Mean_coverage)) +
  geom_bar(stat = "identity") +

  facet_grid(~n+p+noise_sd+nchain)+
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))+
  labs(y = "coverage", x = "method")
  facet_wrap(~ `fit_results$.dgp_name`)
return(plt)
}

```



```

coverage_plot <- create_visualizer(
  .viz_fun = plot_coverage, .name = 'coverage Plot',
  # additional named parameters to pass to .viz_fun()
)

time_plot <- create_visualizer(
  .viz_fun = plot_time, .name = 'time Plot',
  # additional named parameters to pass to .viz_fun()
)

mse_plot <- create_visualizer(
  .viz_fun = plot_mse, .name = 'MSE Plot',
  # additional named parameters to pass to .viz_fun()
)

```

model fitting

```

BART <- create_method(
  .method_fun = BART_fun, .name = "BART",
  k=2.5,q=0.95,df=4,nchain = 1,budget=1200
)
dbarts <- create_method(.method_fun = dbarts_fun,.name = "dbarts",
  k=2.5,q=0.95,df=4,nchain = 1,budget=1200)
bartMachine <- create_method(.method_fun = bartMachine_fun,.name = "bartMachine",
  k=2.5,q=0.95,df=4,budget=1200)
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")
bartpy2 <- create_method(.method_fun = bartpy_fun,.name = "bartpy",budget=1200)
stochtree <- create_method(.method_fun = stochtree_fun, .name = "stochtree",q=0.95,budget=1200)
XGB <- create_method(.method_fun = xgb_fun, .name = "XGBoost")
# 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_method(bartpy2)%>%
add_method(stochtree)%>%
add_method(XGB)%>%
add_visualizer(mse_plot)%>%
add_visualizer(time_plot)%>%
add_visualizer(coverage_plot)%>%
add_vary_across(
  .dgp = "Linear DGP",
  noise_sd = c(0.5),
  n=c(200),
  p=c(4,6)
) %>%
add_vary_across(
  .method = c("BART","dbarts","bartMachine","stochtree","bartpy"),
  nchain=c(1,3)
)
#add_evaluator(pred_err)

results <- run_experiment(experiment, n_reps = 2, save = TRUE)

```

Fitting Test Experiment...

Saving fit results...

Fit results saved | time taken: 0.128387 seconds

2 reps completed (totals: 2/2) | time taken: 5.804757 minutes

=====

No evaluators to evaluate. Skipping evaluation.

=====

Visualizing Test Experiment...

`summarise()` has grouped output by 'fit_results\$.dgp_name', 'fit_results\$.method_name', 'n'

`summarise()` has grouped output by 'fit_results\$.dgp_name', 'fit_results\$.method_name', 'n'

Visualization completed | time taken: 0.002618 minutes

Saving viz results...

Viz results saved | time taken: 0.100657 seconds

=====

```
# Render automated documentation and view results
```

```
#render_docs(experiment)
```

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

```
# A tibble: 52 x 11
```

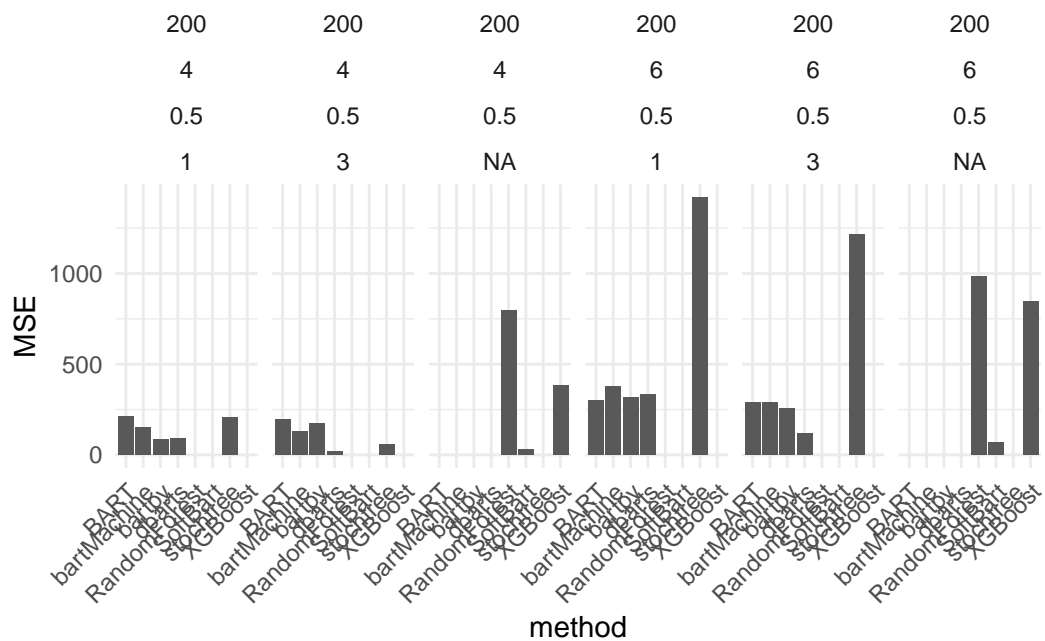
	.rep	.dgp_name	.method_name	noise_sd	n	p	nchain	time	mse
	<chr>	<chr>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<list>	<dbl>
1	1	Linear	DGP BART	0.5	200	4	1	<bench_tm>	214.
2	1	Linear	DGP BART	0.5	200	4	3	<bench_tm>	198.
3	1	Linear	DGP BART	0.5	200	6	1	<bench_tm>	300.
4	1	Linear	DGP BART	0.5	200	6	3	<bench_tm>	288.
5	1	Linear	DGP RandomForest	0.5	200	4	NA	<bench_tm>	827.
6	1	Linear	DGP RandomForest	0.5	200	6	NA	<bench_tm>	981.
7	1	Linear	DGP SoftBart	0.5	200	4	NA	<bench_tm>	38.8
8	1	Linear	DGP SoftBart	0.5	200	6	NA	<bench_tm>	70.6
9	1	Linear	DGP XGBoost	0.5	200	4	NA	<bench_tm>	381.
10	1	Linear	DGP XGBoost	0.5	200	6	NA	<bench_tm>	847.

```
# i 42 more rows
```

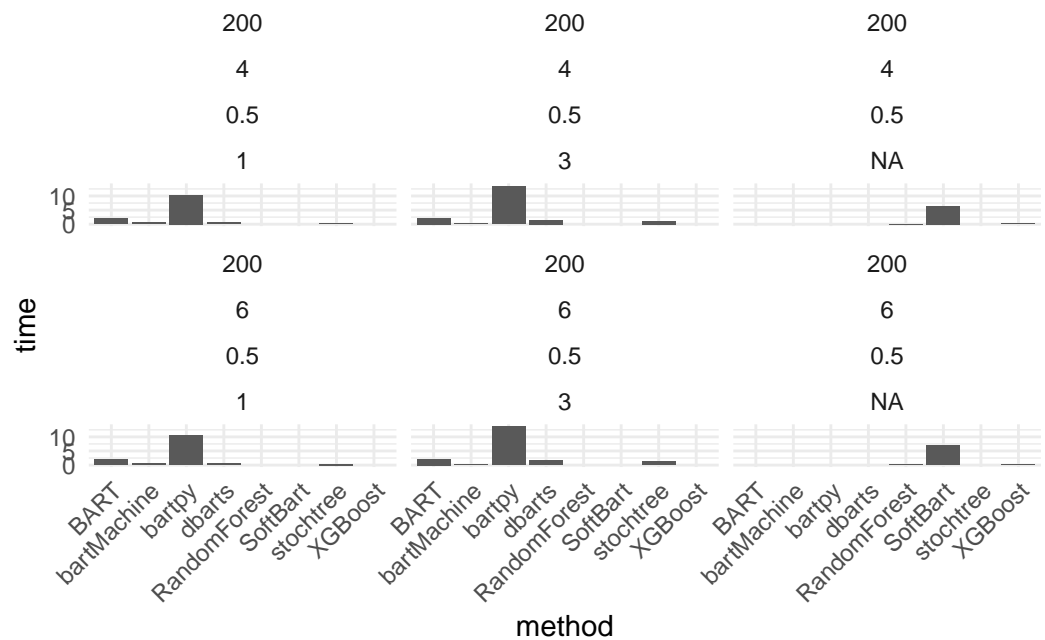
```
# i 2 more variables: coverage <dbl>, time_numeric <dbl>
```

```
results$viz_results
```

```
$`MSE Plot`
```



\$`time Plot`



\$`coverage Plot`

Warning: Removed 4 rows containing missing values or values outside the scale range
(`geom_bar()`).

