Single-Atom Catalysis Data Analysis

Package and Data Loading

The single-atom catalysis data is stored in `data/single_atom_catalysis.RData`, and the raw data is available at this Github repo. In this vignette, we will demonstrate how to use **iBART** in a real data application and reproduce Figure 7 of the paper. Before loading the **iBART** package, we must allocate enough memory for Java to avoid out of memory errors.

```
# Allocate 10GB of memory for Java. Must be called before library(iBART)
options(java.parameters = "-Xmx10g")
library(iBART)
```

Next, we load the real data set and examine what data are needed to run iBART.

```
load("../data/single_atom_catalysis.RData")
ls()
#> [1] "head" "unit" "X"    "y"
```

The data set consists of 4 objects:

- `y`: a `numeric` vector of binding energy of metal-support pairs. This is our response variable.
- `X`: a `matrix` of physical properties of the metal-support pairs. This is our primary features (predictors).
- `head`: a `character` vector storing the column names of `X`.
- `unit`: a (optional) `list` of named numeric vectors. This stores the unit information of the primary features `X`. This can be generated using the helper function `generate_unit(unit, dimension)`. See `?iBART::generate_unit` for more detail.

iBART

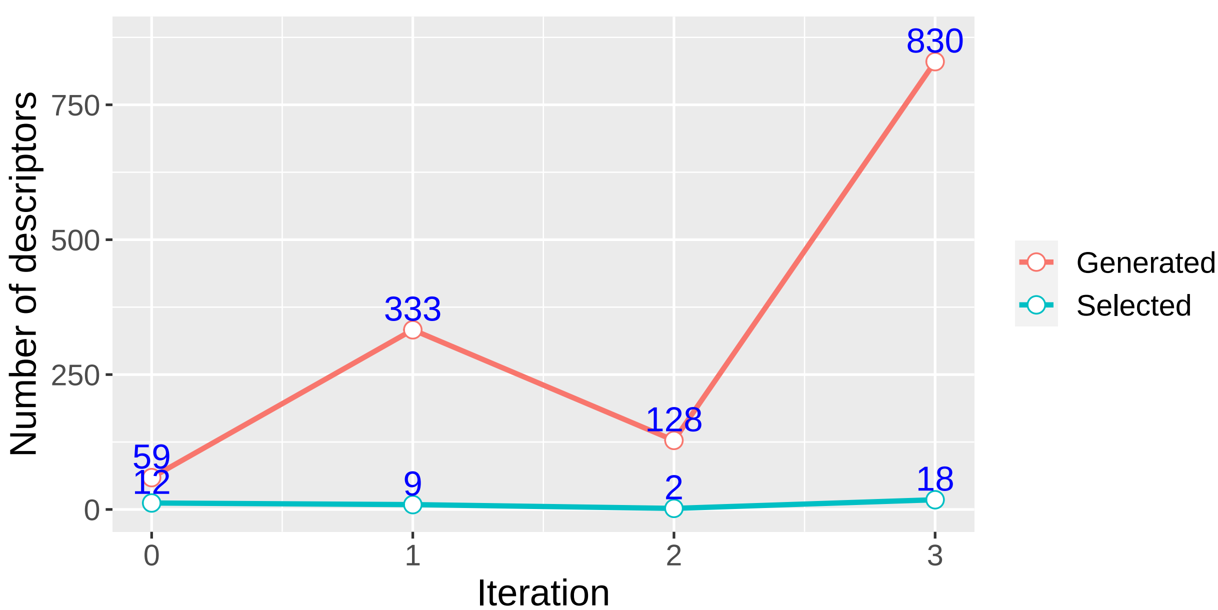
Now let's apply iBART to this data set. Besides the usual regression data `(X,y)`, we need to specify the descriptor generating strategy through `opt`. Here we specify `opt = c("binary", "unary", "binary")`, meaning there will be 3 iterations and we want to alternate between binary and unary operators, starting with binary operators $\mathcal{O}_b$. We can also use all operators O in an iteration. For example, `opt = c("all", "all")` will apply all operators O for 2 iterations.

```
iBART_results <- iBART(X = X, y = y,
                       head = head,  # colnames of X
                       unit = unit,  # units of X
                       opt = c("binary", "unary", "binary"), # binary operator first
                       out_sample = FALSE,
                       Lzero = TRUE,
                       K = 5, # maximum descriptors in l-zero model
                       standardize = FALSE,
                       seed = 888)
#> Start iBART descriptor generation and selection...
#> Iteration 1
#> iBART descriptor selection...
#> avg.........null..................................................
#> Constructing descriptors using binary operators...
#> Iteration 2
```

```
#> iBART descriptor selection...
#> avg.........null.....................................................
#> Constructing descriptors using unary operators...
#> Iteration 3
#> iBART descriptor selection...
#> avg.........null.....................................................
#> Constructing descriptors using binary operators...
#> BART iteration done!
#> LASSO descriptor selection...
#> L-zero regression...
#> Total time: 149.783242940903 secs
```

`iBART()` returns many interesting outputs. For example, `iBART_results$iBART_gen_size` and `iBART_results$iBART_sel_size` store dimension of the newly generated / selected descriptor space for each iteration. Let's plot them and see how **iBART** use nonparametric variable selection for dimension reduction.

```
library(ggplot2)
df_dim <- data.frame(dim = c(iBART_results$iBART_sel_size, iBART_results$iBART_gen_size),
                     iter = rep(0:3, 2),
                     type = rep(c("Selected", "Generated"), each = 4))
ggplot(df_dim, aes(x = iter, y = dim, colour = type, group = type)) +
  theme(text = element_text(size = 15), legend.title = element_blank()) +
  geom_line(size = 1) +
  geom_point(size = 3, shape = 21, fill = "white") +
  geom_text(data = df_dim, aes(label = dim, y = dim + 40, group = type),
            position = position_dodge(0), size = 5, colour = "blue") +
  labs(x = "Iteration", y = "Number of descriptors") +
  scale_x_continuous(breaks = c(0, 1, 2, 3))
#> Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
#> i Please use `linewidth` instead.
#> This warning is displayed once every 8 hours.
#> Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
#> generated.
```



We can access the selected k-descriptor via `iBART_results$Lzero_names` and the corresponding regression model in `iBART_results$Lzero_models`. For instance, the selected 3-descriptor model is

```
iBART_results$Lzero_names[[3]]
#> [1] "(s_EA*Hf)"                   "abs((Hfo/Oxv))"
#> [3] "abs(((m_n13/m_N_val)/Oxv))"
summary(iBART_results$Lzero_models[[3]])
#>
#> Call:
#> lm(formula = y_train ~ ., data = dat_train)
#>
#> Residuals:
#>      Min       1Q   Median       3Q      Max
#> -1.70871 -0.42326  0.05825  0.44715  1.97315
#>
#> Coefficients:
#>                              Estimate Std. Error t value Pr(>|t|)
#> (Intercept)                  -0.01707    0.12675  -0.135    0.893
#> `(s_EA*Hf)`                   0.40427    0.04441   9.104 2.75e-14 ***
#> `abs((Hfo/Oxv))`             -0.58838    0.09857  -5.969 5.05e-08 ***
#> `abs(((m_n13/m_N_val)/Oxv))` -19.62963    4.25098  -4.618 1.33e-05 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 0.6378 on 87 degrees of freedom
#> Multiple R-squared:  0.9534, Adjusted R-squared:  0.9518
#> F-statistic: 593.9 on 3 and 87 DF,  p-value: < 2.2e-16
```

OIS vs non-OIS

The OIS model differs from the non-OIS model in that the former builds on nonlinear descriptors (composition of $\mathcal{O}$ on X) while the latter builds on the primary features X. The OIS model has many advantages. In particular, it reveals interpretable nonlinear relationship between y and X, and improves prediction accuracy over a simple linear regression model (or non-OIS model). We showcase the improved accuracy over non-OIS model using Figure 7 in the paper.

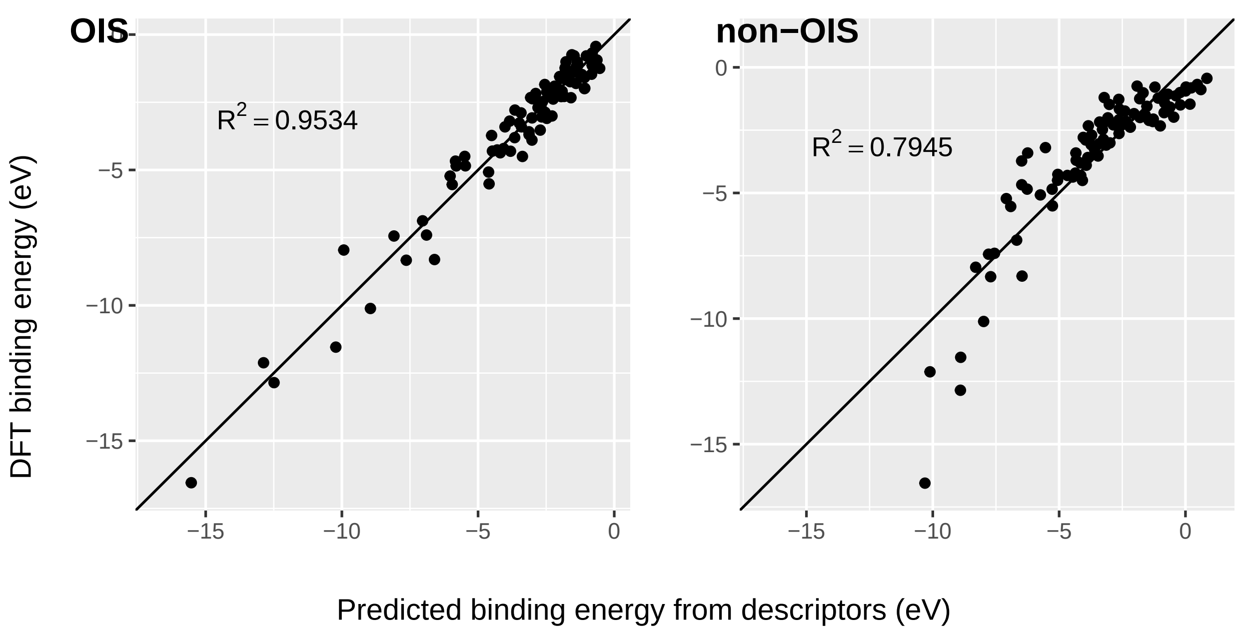
```
# Train a non-OIS model with 3 predictors
set.seed(123)
model_no_OIS <- k_var_model(X_train = X, y_train = y, k = 3, parallel = FALSE)

#### Figure 7 ####
library(ggpubr)
model_OIS <- iBART_results$Lzero_model[[3]]

# Prepare data for plotting
data_OIS <- data.frame(y = y, y_hat = model_OIS$fitted.values)
data_no_OIS <- data.frame(y = y, y_hat = model_no_OIS$models$fitted.values)

p1 <- ggplot(data_OIS, aes(x = y_hat, y = y)) +
  geom_point() +
  geom_abline() +
  xlim(c(min(data_OIS$y_hat, data_OIS$y) - 0.2, max(data_OIS$y_hat, data_OIS$y) + 0.2)) +
  ylim(c(min(data_OIS$y_hat, data_OIS$y) - 0.2, max(data_OIS$y_hat, data_OIS$y) + 0.2)) +
  xlab("") +
  ylab("") +
```

```r
  annotate("text", x = -12, y = -3, parse = TRUE,
           label = paste("R^{2} ==", round(summary(model_OIS)$r.squared, 4)))
p2 <- ggplot(data_no_OIS, aes(x = y_hat, y = y)) +
  geom_point() +
  geom_abline() +
  xlim(c(min(data_no_OIS$y_hat, data_no_OIS$y) - 0.2, max(data_no_OIS$y_hat, data_no_OIS$y) + 0.2)) +
  ylim(c(min(data_no_OIS$y_hat, data_no_OIS$y) - 0.2, max(data_no_OIS$y_hat, data_no_OIS$y) + 0.2)) +
  xlab("") +
  ylab("") +
  annotate("text", x = -12, y = -3, parse = TRUE,
           label = paste("R^{2} ==", round(summary(model_no_OIS$models)$r.squared, 4)))
fig <- ggarrange(p1, p2,
                 labels = c("OIS", "non-OIS"),
                 ncol = 2, nrow = 1)
annotate_figure(fig,
                bottom = text_grob("Predicted binding energy from descriptors (eV)"),
                left = text_grob("DFT binding energy (eV)", rot = 90))
```



R Session Info

```r
sessionInfo()
#> R version 4.0.5 (2021-03-31)
#> Platform: x86_64-w64-mingw32/x64 (64-bit)
#> Running under: Windows 10 x64 (build 22621)
#>
#> Matrix products: default
#>
#> locale:
#> [1] LC_COLLATE=English_United States.1252
#> [2] LC_CTYPE=English_United States.1252
#> [3] LC_MONETARY=English_United States.1252
#> [4] LC_NUMERIC=C
#> [5] LC_TIME=English_United States.1252
```

```
#>
#> attached base packages:
#> [1] stats     graphics  grDevices utils     datasets  methods   base
#>
#> other attached packages:
#> [1] ggpubr_0.6.0  ggplot2_3.4.4 iBART_0.0.3.3
#>
#> loaded via a namespace (and not attached):
#>  [1] shape_1.4.6         tidyselect_1.2.0    xfun_0.40
#>  [4] purrr_1.0.2         splines_4.0.5       rJava_1.0-4
#>  [7] lattice_0.20-44     carData_3.0-5       colorspace_2.0-3
#> [10] vctrs_0.6.4         generics_0.1.3      htmltools_0.5.6.1
#> [13] yaml_2.3.5          utf8_1.2.2          survival_3.2-11
#> [16] rlang_1.1.1         pillar_1.9.0        glue_1.6.2
#> [19] withr_2.5.1         foreach_1.5.1       lifecycle_1.0.3
#> [22] munsell_0.5.0       ggsignif_0.6.4      gtable_0.3.4
#> [25] codetools_0.2-18    evaluate_0.22       labeling_0.4.3
#> [28] knitr_1.44          fastmap_1.1.1       parallel_4.0.5
#> [31] fansi_1.0.3         itertools_0.1-3     broom_1.0.5
#> [34] bartMachine_1.2.6   scales_1.2.1        backports_1.4.1
#> [37] abind_1.4-5         farver_2.1.0        gridExtra_2.3
#> [40] digest_0.6.33       rstatix_0.7.2       dplyr_1.1.3
#> [43] cowplot_1.1.1       grid_4.0.5          cli_3.6.1
#> [46] tools_4.0.5         magrittr_2.0.3      missForest_1.4
#> [49] glmnet_4.1-1        tibble_3.2.1        randomForest_4.6-14
#> [52] crayon_1.5.2        tidyr_1.3.0         car_3.1-2
#> [55] pkgconfig_2.0.3     Matrix_1.3-4        bartMachineJARs_1.1
#> [58] rmarkdown_2.25      rstudioapi_0.15.0   iterators_1.0.13
#> [61] R6_2.5.1            compiler_4.0.5
```