Single-Atom Catalysis Data Analysis

Data Description

The single-atom catalysis data is stored in `data/single_atom_catalysis.RData`, and the raw data is available at this GitHub repo. Here we model the metal/oxide binding energy (the response variable y) using $p = 59$ physical properties of the transition metals and the oxide supports (the primary features X). The response variable y and the primary features X are treated as continuous variables, and we aim to use **iBART** to find an interpretable model with high predictive performance for the metal/oxide binding energy.

A total of 13 transition metals (Cu, Ag, Au, Ni, Pd, Pt, Co, Rh, Ir, Fe, Ru, Mn, V) and 7 oxide supports (CeO_2(111), MgO(100), CeO_2(110), TbO_2(111), ZnO(100), TiO_2(011), α-Al_2O_3(0001)) were studied in the dataset, making a total of $n = 13 \times 7 = 91$ metal/oxide pairs. The primary feature matrix X contains various physical properties of the transition metals and the oxide supports including Pauling Electronegativity (χ_P), $(n-1)^{\text{th}}$ and n^{th} Ionization Energies (IE_{n-1}, IE_n), Electron Affinity (EA), HOMO Energy, LUMO Energy, Heat of Sublimation (ΔH_{sub}), Oxidation Energy of oxide support ($\Delta H_{\text{f,ox,bulk}}$), Oxide Formation Enthalpy ($\Delta H_{\text{f,ox}}$), Zunger Orbital Radius (r), Atomic Number (Z), Meidema Parameters of metal atoms ($\eta^{1/3}, \varphi$), Valance Electron (N_{val}), Oxygen Vacancy Energy of oxide support ($\Delta\text{E}_{\text{vac}}$), Workfunction of oxide support (WF), Surface Energy (γ), Coordination Number (CN), and Bond Valence of surface metal atom (BV). Most of these physical properties are defined for both the transition metals and the oxide supports, while a few are only defined for the transition metals or the oxide supports. A detailed description of the 59 primary features X can be found on pages 11–14 of the data supplementary materials published by O'Connor et al.

Package and Data Loading

Before loading the **iBART** package, we must allocate enough memory to 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")    # load data
summary(catalysis)
#>      Length Class  Mode
#> X    5369   -none- numeric
#> y      91   -none- numeric
#> head   59   -none- character
#> unit   59   -none- list
```

The data set consists of 4 objects:

- `X`: a `matrix` of physical properties of the transition metals and the oxide supports described in Data Description. These are our primary features (predictors).
- `y`: a `numeric` vector of metal/oxide binding energy described in Data Description. This is our response variable.
- `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 the data set. Besides the usual regression data (X,y), we need to specify the descriptor generating strategy through opt. Here we set opt = c("binary", "unary", "binary"), the descriptor generating strategy described in (8) of our paper. That is, we let iBART run for 3 descriptor generating iterations, where binary operators $\mathcal{O}_b$ are used in the 1st iteration, unary operators $\mathcal{O}_u$ in the 2nd iteration, and binary operators $\mathcal{O}_b$ in the 3rd iteration.

We can also define other descriptor generating strategies. The available operator sets at each iteration are

- all: all operators $\mathcal{O} = \{+, -, \times, /, |-|, I, \exp, \log, |\cdot|, \sqrt{}, ^{-1}, ^2, \sin(\pi\cdot), \cos(\pi\cdot)\}$
- binary: binary operators $\mathcal{O}_b = \{+, -, \times, /, |-|\}$
- unary: unary operators $\mathcal{O}_u = \{I, \exp, \log, |\cdot|, \sqrt{}, ^{-1}, ^2, \sin(\pi\cdot), \cos(\pi\cdot)\}$

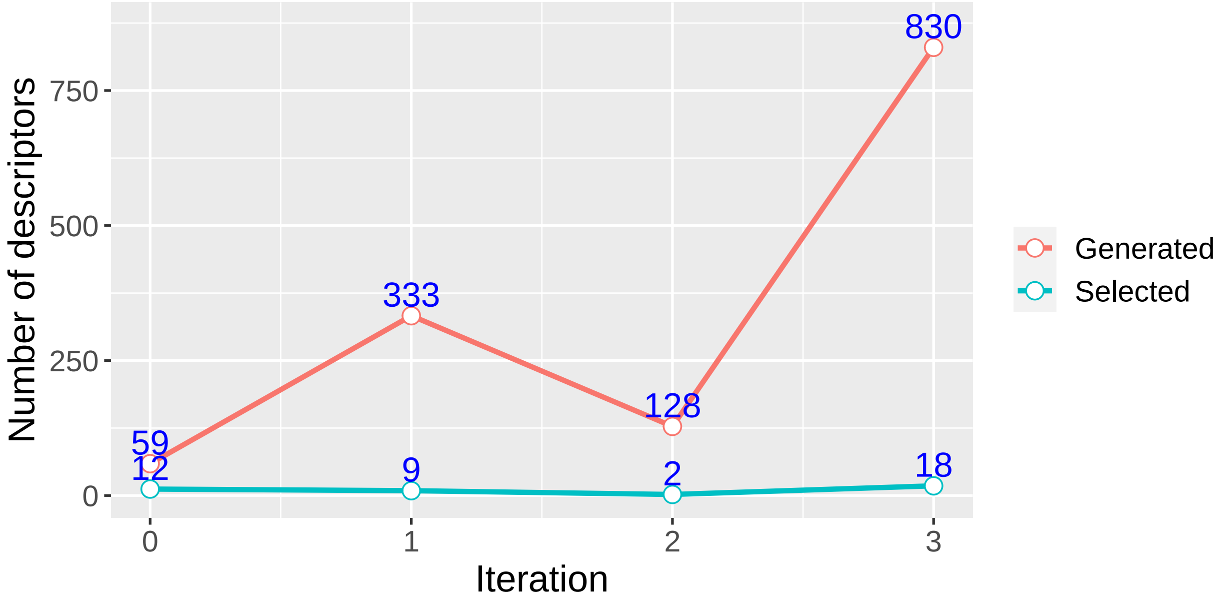
For example, opt = c("all", "all") will apply all operators $\mathcal{O}$ for 2 iterations. Here we use the same tuning parameters discussed in Section 3.4 of our paper.

```r
iBART_real_data <- 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 number of 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: 153.510401010513 secs
```

Dimension Reduction

iBART() returns many interesting outputs. For example, iBART_real_data$iBART_gen_size and iBART_real_data$iBART_sel_size return the dimensions of the generated and selected descriptor space for each iteration, respectively. Let's plot them and see how iBART uses nonparametric variable selection to achieve effective dimension reduction.

```r
library(ggplot2)
df_dim <- data.frame(dim = c(iBART_real_data$iBART_sel_size, iBART_real_data$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(linewidth = 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))
```



Due to the iterative nonparametric screening framework of iBART, the dimension of the selected space is significantly smaller than that of the generated space at each iteration. This ensures that only a sparse subset of the intermediate descriptors is used to generate the consecutive descriptor space, and thus achieving a progressive dimension reduction.

iBART model

The full selected model is stored in `iBART_real_data$iBART_model`, which is a `cv.glmnet` object since LASSO is used the last iteration. This means that we can use all of the `glmnet` functionality. For instance, we can obtain the model coefficients at $\lambda =$`lambda.min` using

```
coef(iBART_real_data$iBART_model, s = iBART_real_data$iBART_model$lambda.min)
```

To view the selected descriptors only, we can do the following instead

```
iBART_real_data$descriptor_names   # Symbolic syntax of the selected descriptors
#>  [1] "(s_EA*Hf)"                             "(m_n13*m_Chi_MB_un)"
#>  [3] "(Hf*Oxv)"                              "((Hfo*s_EA)*abs((Oxv-s_ion_4)))"
#>  [5] "(Hfo/Hf)"                              "(m_n13/s_phi)"
#>  [7] "(Hf/Oxv)"                              "((Hfo*s_EA)/(m_N_val/CMS))"
#>  [9] "(s_ion_3/Hf)"                          "((m_n13/m_N_val)/(m_N_val/CMS))"
#> [11] "((m_n13/m_N_val)/(m_N_val/s_ion_4))" "abs((Hfo/Oxv))"
#> [13] "abs((m_n13/Oxv))"                      "abs((Hf/Oxv))"
#> [15] "abs((CMS/Hf))"                         "abs((s_ion_3/Hf))"
#> [17] "abs(((Hfo*s_EA)/Oxv))"                 "abs(((m_n13/m_N_val)/Oxv))"

iBART_real_data$coefficients       # Coefficients of the selected descriptors
#>                            Intercept                           (s_EA*Hf)
#>                         2.2667558260                        0.0641410077
#>                 (m_n13*m_Chi_MB_un)                            (Hf*Oxv)
#>                       -0.2171094963                       -0.0102489240
#>     ((Hfo*s_EA)*abs((Oxv-s_ion_4)))                            (Hfo/Hf)
#>                       -0.0005014602                        0.0015104408
#>                       (m_n13/s_phi)                            (Hf/Oxv)
#>                       -3.3557729563                       -0.2006426371
#>          ((Hfo*s_EA)/(m_N_val/CMS))                        (s_ion_3/Hf)
#>                       -0.1284775976                        0.0002178855
#>     ((m_n13/m_N_val)/(m_N_val/CMS)) ((m_n13/m_N_val)/(m_N_val/s_ion_4))
#>                       -1.0673897941                       -0.7633565895
#>                      abs((Hfo/Oxv))                    abs((m_n13/Oxv))
#>                       -0.3578988249                       -0.8954807952
#>                       abs((Hf/Oxv))                       abs((CMS/Hf))
#>                       -0.1567861860                        0.0052516576
#>                   abs((s_ion_3/Hf))               abs(((Hfo*s_EA)/Oxv))
#>                        0.0010605538                       -0.0133682249
#>          abs(((m_n13/m_N_val)/Oxv))
#>                       -1.7113215006
```

iBART+ℓ_0

If `Lzero = TRUE`, iBART+ℓ_0 will be run and `K` controls the maximum number of descriptors in a model. Here we set `K=5` so iBART+ℓ_0 will return 5 models: the best 1-descriptor model, 2-descriptor model, and so on. We can access the best k-descriptor via `iBART_real_data$Lzero_names`, and their corresponding regression models using `iBART_real_data$Lzero_models`. For instance, the best 3-descriptor model is

```
iBART_real_data$Lzero_names[[3]]
#> [1] "(s_EA*Hf)"                  "abs((Hfo/Oxv))"
#> [3] "abs(((m_n13/m_N_val)/Oxv))"
summary(iBART_real_data$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

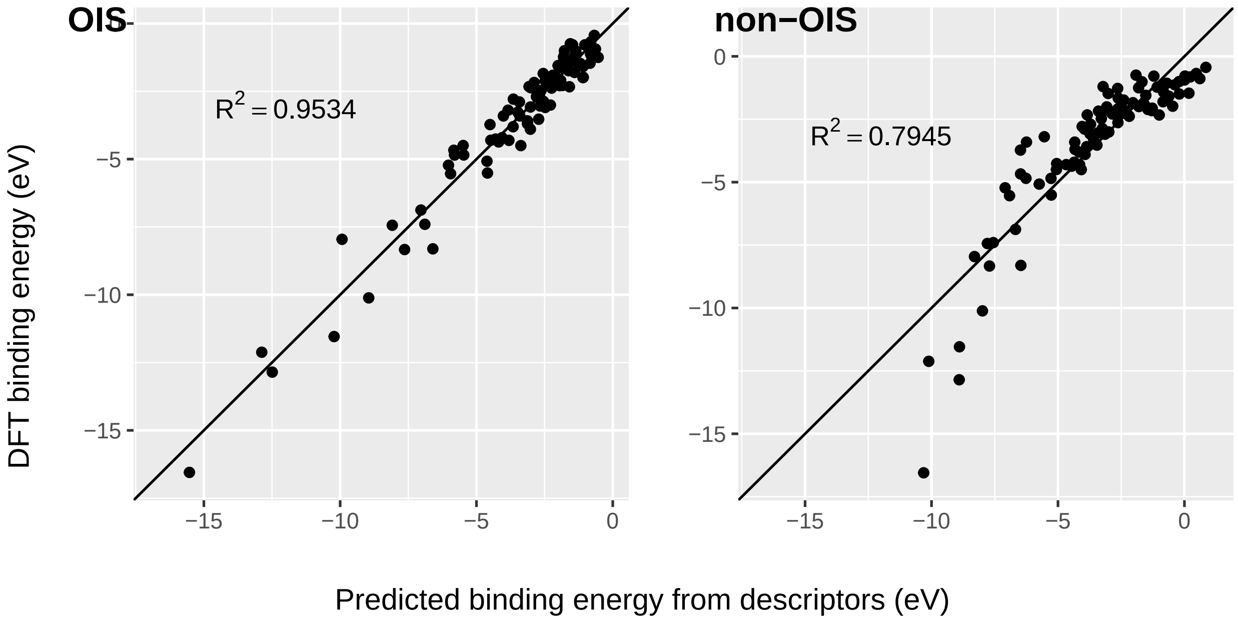
The OIS model differs from the non-OIS model in that the former builds on nonlinear descriptors (composition of $\mathcal{O}$ on X) while the latter builds on the primary features X. The OIS model has many advantages. In particular, it reveals an interpretable nonlinear relationship between y and X,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

```r
# 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_real_data$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("") +
  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

```
sessionInfo()
#> R version 4.3.2 (2023-10-31 ucrt)
#> Platform: x86_64-w64-mingw32/x64 (64-bit)
#> Running under: Windows 11 x64 (build 22631)
#>
#> Matrix products: default
#>
#>
#> locale:
#> [1] LC_COLLATE=English_United States.utf8
#> [2] LC_CTYPE=English_United States.utf8
#> [3] LC_MONETARY=English_United States.utf8
#> [4] LC_NUMERIC=C
#> [5] LC_TIME=English_United States.utf8
#>
#> time zone: America/Chicago
#> tzcode source: internal
#>
#> attached base packages:
#> [1] stats     graphics  grDevices utils     datasets  methods   base
#>
#> other attached packages:
#> [1] ggpubr_0.6.0  ggplot2_3.4.4 iBART_1.0.0
#>
#> loaded via a namespace (and not attached):
#>  [1] tidyr_1.3.0          utf8_1.2.4           generics_0.1.3
#>  [4] bartMachineJARs_1.1  rstatix_0.7.2        shape_1.4.6
#>  [7] lattice_0.21-9       digest_0.6.33        magrittr_2.0.3
#> [10] evaluate_0.23        grid_4.3.2           iterators_1.0.14
#> [13] fastmap_1.1.1        foreach_1.5.2        glmnet_4.1-6
#> [16] Matrix_1.6-1.1       missForest_1.5       backports_1.4.1
#> [19] survival_3.5-7       gridExtra_2.3        bartMachine_1.2.6
#> [22] purrr_1.0.2          fansi_1.0.5          doRNG_1.8.6
#> [25] itertools_0.1-3      scales_1.2.1         codetools_0.2-19
#> [28] abind_1.4-5          cli_3.6.1            rlang_1.1.2
#> [31] cowplot_1.1.1        munsell_0.5.0        splines_4.3.2
#> [34] withr_2.5.2          yaml_2.3.7           tools_4.3.2
#> [37] parallel_4.3.2       ggsignif_0.6.4       dplyr_1.1.3
#> [40] colorspace_2.1-0     rngtools_1.5.2       broom_1.0.5
#> [43] vctrs_0.6.4          R6_2.5.1             lifecycle_1.0.4
#> [46] randomForest_4.7-1.1 car_3.1-2            pkgconfig_2.0.3
#> [49] rJava_1.0-6          pillar_1.9.0         gtable_0.3.4
#> [52] glue_1.6.2           Rcpp_1.0.11          xfun_0.41
#> [55] tibble_3.2.1         tidyselect_1.2.0     highr_0.10
#> [58] rstudioapi_0.15.0    knitr_1.45           farver_2.1.1
#> [61] htmltools_0.5.7      carData_3.0-5        rmarkdown_2.25
#> [64] labeling_0.4.3       compiler_4.3.2
```