Multifunctional peptide engineering

Renee Ti Chou and Henry T. Hsueh

2023-03-12

Contents

Preface

The research notebook contains the code of the machine learning algorithms used to generate the results in the paper "*Machine learning-driven multifunctional peptide engineering for sustained ocular drug delivery.*" The research involves a super learner-based methodology to improve multi-functional peptide engineering. The aim is to impart high melanin binding, high cell-penetration, and low cytotoxicity to ocular drugs through peptide-drug conjugation, with the ultimate goal of enhancing the sustained delivery of the drug to maintain the therapeutic level in the eye for a prolonged period.

Citation:

Chou RT, *et al.* Supplementary materials for machine learning-driven multifunctional peptide engineering for sustained ocular drug delivery. https://doi.org/10.13016/0jck-hnnv, (2023).

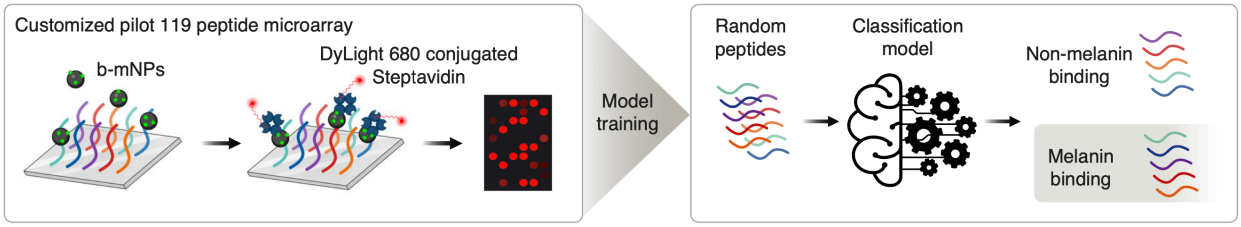
```
@misc{https://doi.org/10.13016/0jck-hnnv,
  doi = {10.13016/0JCK-HNNV},
  url = {https://drum.lib.umd.edu/handle/1903/29529},
  author = {Chou,  Renee Ti and Hsueh,  Henry T. and Rai,  Usha and Liyanage,  Wathsala and Kim,  Yoo
   ↪  Chun and Appell,  Matthew B. and Pejavar,  Jahnavi and Leo,  Kirby T. and Davison,  Charlotte and
   ↪  Kolodziejski,  Patricia and Mozzer,  Ann and Kwon,  HyeYoung and Sista,  Maanasa and Anders,
   ↪  Nicole M. and Hemingway,  Avelina and Rompicharla,  Sri Vishnu Kiran and Edwards,  Malia and
   ↪  Pitha,  Ian and Hanes,  Justin and Cummings,  Michael P. and Ensign,  Laura M.},
  keywords = {machine learning,  drug delivery},
  language = {en},
  title = {Supplementary materials for machine learning-driven multifunctional peptide engineering for
   ↪  sustained ocular drug delivery},
  publisher = {Digital Repository at the University of Maryland},
  year = {2023}
}
```

Section 1

Melanin binding pilot peptide array

```
library(seqinr)
library(Peptides)
library(stringr)
library(protr)
library(plyr)
library(randomForest)
library(ggplot2)
```

1.1 Overview



Created with BioRender.com

1.2 Generating ML input

```
# --------------
# Peptides_2.4.4
# --------------
# read in fasta file
peptides <- read.fasta("./other_data/mb_pilot_peptide_array.fasta", seqtype = "AA", as.string = TRUE,
↪  set.attributes = FALSE)
# Molecular weight
weights <- as.matrix(sapply(peptides, function(x) mw(x, monoisotopic = FALSE)))
colnames(weights) <- "weight"
```

```
# Amino acid composition
aa.comp <- sapply(peptides, function(x) aaComp(x))
aa.comp.matrix <- t(sapply(aa.comp, function(x) x[, "Mole%"]))
# Isoelectric point
pI.values <- as.matrix(sapply(peptides, function(x) pI(x, pKscale = "EMBOSS")))
colnames(pI.values) <- "pI.value"
# Hydrophobicity
hydrophobicity.values <- as.matrix(sapply(peptides, function(x) hydrophobicity(x, scale =
↪  "KyteDoolittle")))
colnames(hydrophobicity.values) <- "hydrophobicity.value"
# Net charge at pH 7
net.charges <- as.matrix(sapply(peptides, function(x) charge(x, pH = 7, pKscale = "EMBOSS")))
colnames(net.charges) <- "net.charge"
# Boman index
boman.indices <- as.matrix(sapply(peptides, function(x) boman(x)))
colnames(boman.indices) <- "boman.index"
# combine peptides results
peptides_res <- data.frame(cbind(weights, aa.comp.matrix, pI.values, hydrophobicity.values,
↪  net.charges, boman.indices))

# -----------
# protr_1.6-2
# -----------
length <- 7
# read in FASTA file
sequences <- readFASTA("./other_data/mb_pilot_peptide_array.fasta")
sequences <- sequences[unlist(lapply(sequences, function(x) nchar(x) > 1))]
# calculate amino acid composition descriptors (dim = 20)
x1 <- t(sapply(sequences, extractAAC))
colnames(x1)[colnames(x1) == "Y"] <- "Y_tyrosine"
# calculate dipeptide composition descriptors (dim = 400)
x2 <- t(sapply(sequences, extractDC))
colnames(x2)[colnames(x2) == "NA"] <- "NA_dipeptide"
# calculate Moreau-Broto autocorrelation descriptors (dim = 8 * (length - 1))
x3 <- t(sapply(sequences, extractMoreauBroto, nlag = length - 1L))
colnames(x3) <- paste("moreau_broto", colnames(x3), sep = "_")
# calculate composition descriptors (dim = 21)
x4 <- t(sapply(sequences, extractCTDC))
# calculate transition descriptors (dim = 21)
x5 <- t(sapply(sequences, extractCTDT))
# calculate distribution descriptors (dim = 105)
```

```
x6 <- t(sapply(sequences, extractCTDD))
# calculate conjoint triad descriptors (dim = 343)
x7 <- t(sapply(sequences, extractCTriad))
# calculate sequence-order-coupling numbers (dim = 2 * (length - 1))
x8 <- t(sapply(sequences, extractSOCN, nlag = length - 1L))
# calculate quasi-sequence-order descriptors (dim = 40 + 2 * (length - 1))
x9 <- t(sapply(sequences, extractQSO, nlag = length - 1L))
# calculate pseudo-amino acid composition (dim = 20 + (length - 1))
x10 <- t(sapply(sequences, extractPAAC, lambda = length - 1L))
# calculate amphiphilic pseudo-amino acid composition (dim = 20 + 2 * (length - 1))
x11 <- t(sapply(sequences, extractAPAAC, lambda = length - 1L))
# combine all of the result datasets
protr_res <- data.frame(cbind(x1, x2, x3, x4, x5, x6, x7, x8, x9, x10, x11))

labels <- read.csv("./other_data/mb_pilot_peptide_array_labels.csv", row.names = 1)
merge.all <- function(x, ..., by = "row.names") {
  L <- list(...)
  for (i in seq_along(L)) {
    x <- merge(x, L[[i]], by = by)
    rownames(x) <- x$Row.names
    x$Row.names <- NULL
  }
  return(x)
}
data <- merge.all(peptides_res, protr_res, labels)
write.csv(data, file = "./data/mb_pilot_peptide_array_ml_input.csv")
```

1.3 Training initial RF model

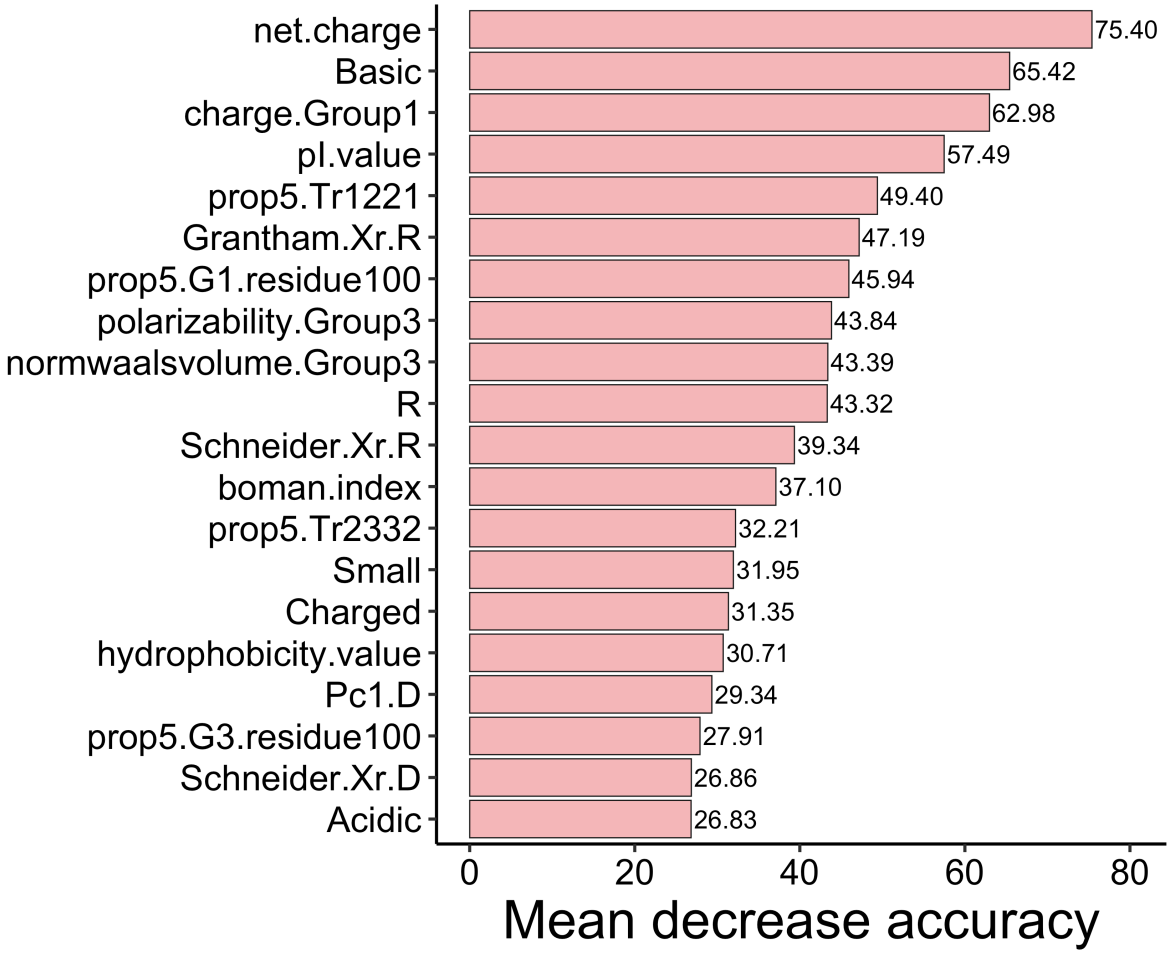
```
data <- read.csv("./data/mb_pilot_peptide_array_ml_input.csv", check.names = FALSE, row.names = 1)
predictor_variables <- subset(data, select = -category)
response_variable <- factor(data$category, levels = c("non-bind", "bind"))
# impute missing values
response_variable <- na.roughfix(response_variable)
# perform balance sampling
samp_size <- min(table(response_variable))
```

```
# build RF model
set.seed(22)
ntree <- 100000
rf <- randomForest(
  x = predictor_variables, y = response_variable, ntree = ntree, sampsize = rep(samp_size, 2),
  importance = TRUE, proximity = TRUE, do.trace = 0.1 * ntree
)
save(rf, file = "./rdata/mb_pilot_rf.RData")
```

1.4 Melanin binding variable importance

```
load(file = "./rdata/mb_pilot_rf.RData")
imp_ds <- as.data.frame(importance(rf)[, "MeanDecreaseAccuracy", drop = FALSE])
imp_ds$`Feature` <- rownames(imp_ds)
colnames(imp_ds) <- c("Mean decrease accuracy", "Feature")
imp_ds <- imp_ds[order(-imp_ds$`Mean decrease accuracy`), ][1:20, ]

p <- ggplot(imp_ds, aes(x = reorder(`Feature`, `Mean decrease accuracy`), y = `Mean decrease
↪  accuracy`)) +
  geom_col(color = "grey10", fill = "#f5b8b8", size = 0.2) +
  geom_text(aes(label = sprintf("%.2f", `Mean decrease accuracy`)), nudge_y = 4.2, size = 3) +
  coord_flip() +
  ylim(0, max(imp_ds$`Mean decrease accuracy`) + 5) +
  theme_bw() +
  theme(
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    strip.background = element_blank(),
    panel.border = element_blank(),
    axis.line = element_line(color = "black"),
    legend.text = element_text(colour = "black", size = 10),
    plot.title = element_text(hjust = 0.5, size = 18),
    plot.margin = ggplot2::margin(10, 10, 10, 0, "pt"),
    axis.title.x = element_text(colour = "black", size = 18),
    axis.title.y = element_text(colour = "black", size = 18),
    axis.text.x = element_text(colour = "black", size = 12),
    axis.text.y = element_text(colour = "black", size = 12),
    legend.title = element_text(size = 14),
```

```
  legend.position = c(0.85, 0.5)
) +
xlab("") +
ggtitle("")
```




```
sessionInfo()
```

```
## R version 4.2.2 (2022-10-31)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Big Sur ... 10.16
```

```
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] ggplot2_3.3.6        randomForest_4.7-1.1 plyr_1.8.7
## [4] protr_1.6-2          stringr_1.4.1        Peptides_2.4.4
## [7] seqinr_4.2-16
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.9         pillar_1.8.1       compiler_4.2.2     R.methodsS3_1.8.2
##  [5] R.utils_2.12.0     tools_4.2.2        digest_0.6.29      evaluate_0.16
##  [9] lifecycle_1.0.1    tibble_3.1.8       gtable_0.3.0       R.cache_0.16.0
## [13] pkgconfig_2.0.3    rlang_1.0.4        DBI_1.1.3          cli_3.3.0
## [17] rstudioapi_0.14    yaml_2.3.5         xfun_0.32          fastmap_1.1.0
## [21] withr_2.5.0        dplyr_1.0.9        styler_1.8.0       knitr_1.40
## [25] generics_0.1.3     vctrs_0.4.1        tidyselect_1.1.2   ade4_1.7-19
## [29] grid_4.2.2         glue_1.6.2         R6_2.5.1           fansi_1.0.3
## [33] rmarkdown_2.16     bookdown_0.28      purrr_0.3.4        magrittr_2.0.3
## [37] codetools_0.2-18   scales_1.2.1       htmltools_0.5.3    MASS_7.3-58.1
## [41] assertthat_0.2.1   colorspace_2.0-3   utf8_1.2.2         stringi_1.7.8
## [45] munsell_0.5.0      R.oo_1.25.0
```

Section 2

Variable reduction

```
library(Peptides)
library(protr)
library(rlist)
library(ranger)
library(ggplot2)
library(cowplot)
library(scales)
```

2.1 Methods

- Formula of AIC

$$AIC = -2\ln(\hat{L}) + 2k, \tag{2.1}$$

$$\begin{aligned} &\text{where } \hat{L} \text{ is the maximum likelihood value,} \\ &\text{and } k \text{ is the number of parameters.} \end{aligned} \tag{2.2}$$

2.1.1 Regression

- Likelihood of normal distribution

  Assume residuals are normally distributed, then

$$L(\theta) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(\hat{y}_i - y_i)^2}{2\sigma^2}} \tag{2.3}$$

$$\begin{aligned} \ln L(\theta) &= \sum_{i=1}^{N} [\ln(\frac{1}{\sqrt{2\pi\sigma^2}}) - \frac{(\hat{y}_i - y_i)^2}{2\sigma^2}] \\ &= -\frac{N}{2}\ln(2\pi) - \frac{N}{2}\ln(\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2. \end{aligned} \tag{2.4}$$

Since $\hat{\sigma}^2 = \frac{\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}{N} = MSE$,

$$\ln L(\theta) = -\frac{N}{2}\ln(MSE) + C, \text{where } C \text{ is a constant.} \tag{2.5}$$

- Derived regression AIC

$$AIC_{reg} = N\ln(MSE) + 2k \tag{2.6}$$

2.1.2 Classification

- Likelihood of Bernoulli distribution

$$L(\theta) = \prod_{i=1}^{N}[q_\theta(y_i = 1)^{y_i} q_\theta(y_i = 0)^{1-y_i}] \tag{2.7}$$

$$\ln L(\theta) = \sum_{i=1}^{N}[y_i \ln q_\theta(y_i = 1) + (1 - y_i)\ln q_\theta(y_i = 0)], \tag{2.8}$$

$$\text{where } q_\theta \text{ is the estimated probability with parameters } \quad \text{and } \ y_i \in \{0, 1\} \tag{2.9}$$

- Binary cross-entropy

$$\begin{aligned} H_p(q_\theta) &= -\frac{1}{N}\sum_{i=1}^{N}[y_i \log_2 q_\theta(y_i = 1) + (1 - y_i)\log_2 q_\theta(y_i = 0)] \\ &= -\frac{1}{N\cdot\ln 2}\sum_{i=1}^{N}[y_i \ln q_\theta(y_i = 1) + (1 - y_i)\ln q_\theta(y_i = 0)] \end{aligned} \tag{2.10}$$

- Derived classification AIC

$$AIC_{clf} = 2\cdot\ln 2\cdot N\cdot H_p(q_\theta) + 2k \tag{2.11}$$

2.1.2.1 Extension to multi-class classification

- Generalized likelihood estimation

  Assume the classes are one-hot encoded, then we can generalize equation (2.11) to multi-class problems.

$$L(\theta) = \prod_{i=1}^{N} \prod_{j=1}^{M} \hat{y}_{ij}^{y_{ij}} \tag{2.12}$$

$$\ln L(\theta) = \sum_{i=1}^{N} \sum_{j=1}^{M} y_{ij} \ln \hat{y}_{ij}, \tag{2.13}$$

where $\hat{y}_{ij}$ is the estimated probability of class j of sample i and $y_{ij} \in \{0, 1\}$ (2.14)

- Categorical cross-entropy

$$H_p(q_\theta) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{M} y_{ij} \log_2 \hat{y}_{ij} \tag{2.15}$$

- Derived generalized classification AIC

$$AIC_{clf} = 2 \cdot \ln 2 \cdot N \cdot H_p(q_\theta) + 2k \tag{2.16}$$

2.2 ML input generating function

```
generate_ml_input <- function(fasta_file, label_file, output_file) {
  # --------------
  # Peptides_2.4.4
  # --------------
  # read in fasta file
  peptides <- read.fasta(fasta_file, seqtype = "AA", as.string = TRUE, set.attributes = FALSE)
  # Molecular weight
  weights <- as.matrix(sapply(peptides, function(x) mw(x, monoisotopic = FALSE)))
  colnames(weights) <- "weight"
  # Amino acid composition
  aa.comp <- sapply(peptides, function(x) aaComp(x))
  aa.comp.matrix <- t(sapply(aa.comp, function(x) x[, "Mole%"]))
  # Isoelectric point
  pI.values <- as.matrix(sapply(peptides, function(x) pI(x, pKscale = "EMBOSS")))
  colnames(pI.values) <- "pI.value"
  # Hydrophobicity
```

```r
  hydrophobicity.values <- as.matrix(sapply(peptides, function(x) hydrophobicity(x, scale =
↪  "KyteDoolittle")))
  colnames(hydrophobicity.values) <- "hydrophobicity.value"
  # Net charge at pH 7
  net.charges <- as.matrix(sapply(peptides, function(x) charge(x, pH = 7, pKscale = "EMBOSS")))
  colnames(net.charges) <- "net.charge"
  # Boman index
  boman.indices <- as.matrix(sapply(peptides, function(x) boman(x)))
  colnames(boman.indices) <- "boman.index"
  # combine peptides results
  peptides_res <- data.frame(cbind(weights, aa.comp.matrix, pI.values, hydrophobicity.values,
↪  net.charges, boman.indices))

  # -----------
  # protr_1.6-2
  # -----------
  length <- 7
  # read in FASTA file
  sequences <- readFASTA(fasta_file)
  sequences <- sequences[unlist(lapply(sequences, function(x) nchar(x) > 1))]
  # calculate amino acid composition descriptors (dim = 20)
  x1 <- t(sapply(sequences, extractAAC))
  colnames(x1)[colnames(x1) == "Y"] <- "Y_tyrosine"
  # calculate dipeptide composition descriptors (dim = 400)
  x2 <- t(sapply(sequences, extractDC))
  colnames(x2)[colnames(x2) == "NA"] <- "NA_dipeptide"
  # calculate Moreau-Broto autocorrelation descriptors (dim = 8 * (length - 1))
  x3 <- t(sapply(sequences, extractMoreauBroto, nlag = length - 1L))
  colnames(x3) <- paste("moreau_broto", colnames(x3), sep = "_")
  # calculate composition descriptors (dim = 21)
  x4 <- t(sapply(sequences, extractCTDC))
  # calculate transition descriptors (dim = 21)
  x5 <- t(sapply(sequences, extractCTDT))
  # calculate distribution descriptors (dim = 105)
  x6 <- t(sapply(sequences, extractCTDD))
  # calculate conjoint triad descriptors (dim = 343)
  x7 <- t(sapply(sequences, extractCTriad))
  # calculate sequence-order-coupling numbers (dim = 2 * (length - 1))
  x8 <- t(sapply(sequences, extractSOCN, nlag = length - 1L))
  # calculate quasi-sequence-order descriptors (dim = 40 + 2 * (length - 1))
  x9 <- t(sapply(sequences, extractQSO, nlag = length - 1L))
```

```
  # calculate pseudo-amino acid composition (dim = 20 + (length - 1))
  x10 <- t(sapply(sequences, extractPAAC, lambda = length - 1L))
  # calculate amphiphilic pseudo-amino acid composition (dim = 20 + 2 * (length - 1))
  x11 <- t(sapply(sequences, extractAPAAC, lambda = length - 1L))
  # combine all of the result datasets
  protr_res <- data.frame(cbind(x1, x2, x3, x4, x5, x6, x7, x8, x9, x10, x11))

  labels <- read.csv(label_file, row.names = 1)
  merge.all <- function(x, ..., by = "row.names") { # https://stackoverflow.com/a/22618297
    L <- list(...)
    for (i in seq_along(L)) {
      x <- merge(x, L[[i]], by = by)
      rownames(x) <- x$Row.names
      x$Row.names <- NULL
    }
    return(x)
  }
  data <- merge.all(peptides_res, protr_res, labels)
  write.csv(data, file = output_file)
}
```

2.3 AIC functions

```
#' Mean squared error function
#'
#' @param y_true a factor vector representing true values.
#' @param y_pred a numeric vector or a matrix of predicted values.
#'
#' @return MSE
#'
mse <- function(y_true, y_pred) {
  return(mean((y_true - y_pred)^2))
}
```

```
#' Regression AIC
#'
#' @param y_true a factor vector representing true values.
```

```r
#' @param y_pred a numeric vector or a matrix of predicted values.
#' @param k number of parameters/variables.
#' @param eps a very small value to avoid negative infinitives generated by log(p=0).
#'
#' @return AIC
#'
aic_reg <- function(y_true, y_pred, k, eps = 1e-15) {
  mserr <- mse(y_true, y_pred)
  if (mserr == 0) mserr <- eps
  AIC <- length(y_true) * log(mserr) + 2 * k
  return(AIC)
}
```

```r
#' Cross-entropy function
#'
#' @param y_true a factor vector representing true labels.
#' @param y_pred a numeric vector (probabilities of the second class/factor level)
#' or a matrix of predicted probabilities.
#' @param eps a very small value to avoid negative infinitives generated by log(p=0).
#' If the function returns NaN, then increasing eps may solve the issue. Alternatively,
#' As NaN is usually generated in the case of p = 1 - eps = 1, resulting in log(1 - p)
#' = log(0) from binary cross-entropy, the user can pass prediction values as a matrix
#' to calculate categorical cross-entropy instead.
#'
#' @return H, cross-entropy (mean loss per sample)
#'
crossEntropy <- function(y_true, y_pred, eps = 1e-15) {
  stopifnot(is.factor(y_true))
  y_pred <- pmax(pmin(y_pred, 1 - eps), eps)
  n_levels <- nlevels(y_true)
  H <- NULL
  # Binary classification
  if (n_levels == 2 && is.vector(y_pred)) {
    y_true <- as.numeric(y_true == levels(y_true)[-1])
    H <- -mean(y_true * log2(y_pred) + (1 - y_true) * log2(1 - y_pred))
  }
  # Multi-class classification
  else if (n_levels == ncol(y_pred)) {
    y_true <- as.numeric(y_true)
```

```
    y_true_encoded <- t(sapply(y_true, function(x) {
      tmp <- rep(0, n_levels)
      tmp[x] <- 1
      return(tmp)
    }))
    H <- -mean(sapply(1:nrow(y_true_encoded), function(x) sum(y_true_encoded[x, ] * log2(y_pred[x,
↪  ])))))
  }
  return(H)
}
```

```
#' Classification AIC
#'
#' @param y_true a factor vector representing true labels.
#' @param y_pred a numeric vector or a matrix of predicted probabilities.
#' @param k number of parameters/variables.
#' @param ... other parameters to be passed to `crossEntropy()`
#'
#' @return AIC
#'
aic_clf <- function(y_true, y_pred, k, ...) {
  H <- crossEntropy(y_true, y_pred, ...)
  AIC <- 2 * log(2) * length(y_true) * H + 2 * k
  return(AIC)
}
```

2.4 Melanin binding (regression)

2.4.1 Generating ML input

```
generate_ml_input(
  fasta_file = "./other_data/mb_second_peptide_array.fasta",
  label_file = "./other_data/mb_second_peptide_array_labels.csv",
  output_file = "./data/mb_second_peptide_array_ml_input.csv"
)
```

2.4.2 Variable importance

```
set.seed(12)
mb_data <- read.csv("./data/mb_second_peptide_array_ml_input.csv", check.names = FALSE, row.names = 1)
# random shuffle
mb_data <- mb_data[sample(1:nrow(mb_data), replace = FALSE), ]

# train random forest using ranger
ntree <- 100000
predictor_variables <- subset(mb_data, select = -log_intensity)
response_variable <- mb_data$log_intensity
rf <- ranger(
  x = predictor_variables, y = response_variable, num.trees = ntree, importance = "permutation",
  ↪  verbose = TRUE,
  scale.permutation.importance = TRUE, seed = 3, keep.inbag = TRUE
)
save(rf, file = "./rdata/var_reduct_mb_rf.RData")
```

2.4.3 Variable reduction

```
load(file = "./rdata/var_reduct_mb_rf.RData")
var_imp <- ranger::importance(rf)
var_imp <- var_imp[order(-var_imp)]
var_imp <- var_imp[var_imp >= 0]

set.seed(12)
mb_data <- read.csv("./data/mb_second_peptide_array_ml_input.csv", check.names = FALSE, row.names = 1)
# random shuffle
mb_data <- mb_data[sample(1:nrow(mb_data), replace = FALSE), ]

ntree <- 1000
var_subset_res <- list()
for (i in 1:length(var_imp)) {
  cat(paste0(i, "\n"))
  predictor_variables <- mb_data[, colnames(mb_data) %in% names(var_imp)[1:i], drop = FALSE]
  response_variable <- mb_data$log_intensity
  rf <- ranger(x = predictor_variables, y = response_variable, num.trees = ntree, importance = "none",
↪  verbose = TRUE, seed = 3)
  var_subset_res <- list.append(var_subset_res, list(
```

```
    rsq = rf$r.squared,
    aic = aic_reg(
      y_true = response_variable,
      y_pred = rf$predictions,
      k = i
    )
  ))
}
save(var_subset_res, file = "./rdata/var_reduct_mb_var_subset_res.RData")
```

```
load(file = "./rdata/var_reduct_mb_var_subset_res.RData")
rsq_vec <- unlist(lapply(var_subset_res, function(x) x$rsq))
aic_vec <- unlist(lapply(var_subset_res, function(x) x$aic))
rsq_cutoff <- which.max(rsq_vec)
aic_cutoff <- which.min(aic_vec)

p1_1 <- ggplot(data = data.frame(x = 1:length(rsq_vec), y = rsq_vec), aes(x = x, y = y)) +
  geom_point(colour = "black", alpha = 0.3, size = 0.5) +
  geom_vline(xintercept = rsq_cutoff, colour = "red", linetype = "dashed") +
  scale_x_continuous(breaks = c(1, rsq_cutoff, 200, 400, 600, 735)) +
  scale_y_continuous(
    breaks = c(0.40, 0.45, 0.50, 0.55),
    labels = c("0.40", "0.45", "0.50", "0.55")
  ) +
  theme_bw() +
  theme(
    plot.margin = unit(c(10, 10, -10, 10), "pt"),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    strip.background = element_blank(),
    panel.border = element_blank(),
    axis.line = element_line(color = "black"),
    plot.title = element_text(hjust = 0.5),
    axis.title = element_text(colour = "black"),
    axis.text = element_text(colour = "black")
  ) +
  xlab("") +
  ylab(expression(italic(R^2))) +
  ggtitle("Melanin binding")
```

```
p1_2 <- ggplot(data = data.frame(x = 1:length(aic_vec), y = aic_vec), aes(x = x, y = y)) +
  geom_point(colour = "black", alpha = 0.3, size = 0.5) +
  geom_vline(xintercept = aic_cutoff, colour = "red", linetype = "dashed") +
  scale_x_continuous(breaks = c(1, aic_cutoff, 200, 400, 600, 735)) +
  scale_y_continuous(
    breaks = c(-500, 0, 500, 1000),
    labels = c("-500", "0", "500", "1,000")
  ) +
  theme_bw() +
  theme(
    plot.margin = unit(c(-15, 10, 10, 10), "pt"),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    strip.background = element_blank(),
    panel.border = element_blank(),
    axis.line = element_line(color = "black"),
    plot.title = element_text(hjust = 0.5),
    axis.title = element_text(colour = "black"),
    axis.text = element_text(colour = "black")
  ) +
  xlab("Number of variables") +
  ylab("AIC") +
  ggtitle("")

cat(paste0("Maximum R-squared: ", formatC(rsq_vec[rsq_cutoff], format = "f", digits = 3), "\n"))
cat(paste0("R-squared at AIC cutoff: ", formatC(rsq_vec[aic_cutoff], format = "f", digits = 3), "\n"))
```

2.4.4 Generating train-test splits

```
set.seed(12)
mb_data <- read.csv("./data/mb_second_peptide_array_ml_input.csv", check.names = FALSE, row.names = 1)
# random shuffle
mb_data <- mb_data[sample(1:nrow(mb_data), replace = FALSE), ]

cutoff <- aic_cutoff
load(file = "./rdata/var_reduct_mb_rf.RData")
var_imp <- ranger::importance(rf)
var_imp <- var_imp[order(-var_imp)][1:cutoff]
```

```
mb_data <- mb_data[, colnames(mb_data) %in% c(names(var_imp), "log_intensity")]

outer_splits <- list()
n_train <- round(nrow(mb_data) * 0.9)
set.seed(22)
for (i in 1:10) {
  # random shuffle
  mb_data_ <- mb_data[sample(1:nrow(mb_data), replace = FALSE), ]
  outer_train <- mb_data_[1:n_train, ]
  outer_test <- mb_data_[(n_train + 1):nrow(mb_data_), ]
  outer_splits <- list.append(outer_splits, list(train = outer_train, test = outer_test))
}
save(mb_data, outer_splits, file = "./rdata/var_reduct_mb_train_test_splits.RData")
```

2.5 Cell-penetration (classification)

2.5.1 Generating ML input

```
generate_ml_input(
  fasta_file = "./other_data/CPP924.fasta",
  label_file = "./other_data/cpp_labels.csv",
  output_file = "./data/cpp_ml_input.csv"
)
```

2.5.2 Variable importance

```
set.seed(12)
cpp_data <- read.csv("./data/cpp_ml_input.csv", check.names = FALSE, row.names = 1)
# random shuffle
cpp_data <- cpp_data[sample(1:nrow(cpp_data), replace = FALSE), ]

# train random forest using ranger
ntree <- 100000
predictor_variables <- subset(cpp_data, select = -category)
response_variable <- factor(cpp_data$category, levels = c("non-penetrating", "penetrating"))
rf <- ranger(
  x = predictor_variables, y = response_variable, num.trees = ntree, sample.fraction = rep(0.5, 2),
```

```
  importance = "permutation", verbose = TRUE, scale.permutation.importance = TRUE, seed = 3,
  ↪  keep.inbag = TRUE
)
save(rf, file = "./rdata/var_reduct_cpp_rf.RData")
```

2.5.3 Variable reduction

```
load(file = "./rdata/var_reduct_cpp_rf.RData")
var_imp <- ranger::importance(rf)
var_imp <- var_imp[order(-var_imp)]
var_imp <- var_imp[var_imp >= 0]

set.seed(12)
cpp_data <- read.csv("./data/cpp_ml_input.csv", check.names = FALSE, row.names = 1)
# random shuffle
cpp_data <- cpp_data[sample(1:nrow(cpp_data), replace = FALSE), ]

ntree <- 1000
var_subset_res <- list()
for (i in 1:length(var_imp)) {
  cat(paste0(i, "\n"))
  predictor_variables <- cpp_data[, colnames(cpp_data) %in% names(var_imp)[1:i], drop = FALSE]
  response_variable <- factor(cpp_data$category, levels = c("non-penetrating", "penetrating"))
  rf <- ranger(
    x = predictor_variables, y = response_variable, num.trees = ntree, importance = "none", verbose =
    ↪  TRUE, seed = 3,
    probability = TRUE
  )
  var_subset_res <- list.append(var_subset_res, list(
    acc = 1 - rf$prediction.error,
    aic = aic_clf(
      y_true = response_variable,
      y_pred = rf$predictions,
      k = i
    )
  ))
}
save(var_subset_res, file = "./rdata/var_reduct_cpp_var_subset_res.RData")
```

```
load(file = "./rdata/var_reduct_cpp_var_subset_res.RData")
acc_vec <- unlist(lapply(var_subset_res, function(x) x$acc))
aic_vec <- unlist(lapply(var_subset_res, function(x) x$aic))
acc_cutoff <- which.max(acc_vec)
aic_cutoff <- which.min(aic_vec)

p2_1 <- ggplot(data = data.frame(x = 1:length(acc_vec), y = acc_vec), aes(x = x, y = y)) +
  geom_point(colour = "black", alpha = 0.3, size = 0.5) +
  geom_vline(xintercept = acc_cutoff, colour = "red", linetype = "dashed") +
  scale_x_continuous(breaks = c(1, acc_cutoff, 250, 500, 750, 955)) +
  scale_y_continuous(
    breaks = c(0.88, 0.90, 0.92),
    labels = c("0.88", "0.90", "0.92")
  ) +
  theme_bw() +
  theme(
    plot.margin = unit(c(10, 10, -10, 10), "pt"),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    strip.background = element_blank(),
    panel.border = element_blank(),
    axis.line = element_line(color = "black"),
    plot.title = element_text(hjust = 0.5),
    axis.title = element_text(colour = "black"),
    axis.text = element_text(colour = "black")
  ) +
  xlab("") +
  ylab("Accuracy") +
  ggtitle("Cell-penetration")

p2_2 <- ggplot(data = data.frame(x = 1:length(aic_vec), y = aic_vec), aes(x = x, y = y)) +
  geom_point(colour = "black", alpha = 0.3, size = 0.5) +
  geom_vline(xintercept = aic_cutoff, colour = "red", linetype = "dashed") +
  scale_x_continuous(breaks = c(aic_cutoff, 250, 500, 750, 955)) +
  scale_y_continuous(
    breaks = c(500, 1000, 1500, 2000, 2500),
    labels = c("500", "1,000", "1,500", "2,000", "2,500")
  ) +
  theme_bw() +
  theme(
    plot.margin = unit(c(-15, 10, 10, 10), "pt"),
```

```
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    strip.background = element_blank(),
    panel.border = element_blank(),
    axis.line = element_line(color = "black"),
    plot.title = element_text(hjust = 0.5),
    axis.title = element_text(colour = "black"),
    axis.text = element_text(colour = "black")
  ) +
  xlab("Number of variables") +
  ylab("AIC") +
  ggtitle("")

cat(paste0("Maximum accuracy: ", formatC(acc_vec[acc_cutoff], format = "f", digits = 3), "\n"))
cat(paste0("Accuracy at AIC cutoff: ", formatC(acc_vec[aic_cutoff], format = "f", digits = 3), "\n"))
```

2.5.4 Generating train-test splits

```
set.seed(12)
cpp_data <- read.csv("./data/cpp_ml_input.csv", check.names = FALSE, row.names = 1)
# random shuffle
cpp_data <- cpp_data[sample(1:nrow(cpp_data), replace = FALSE), ]

cutoff <- aic_cutoff
load(file = "./rdata/var_reduct_cpp_rf.RData")
var_imp <- ranger::importance(rf)
var_imp <- var_imp[order(-var_imp)][1:cutoff]
cpp_data <- cpp_data[, colnames(cpp_data) %in% c(names(var_imp), "category")]

outer_splits <- list()
n_train <- round(nrow(cpp_data) * 0.9)
set.seed(22)
for (i in 1:10) {
  # random shuffle
  cpp_data_ <- cpp_data[sample(1:nrow(cpp_data), replace = FALSE), ]
  outer_train <- cpp_data_[1:n_train, ]
  outer_test <- cpp_data_[(n_train + 1):nrow(cpp_data_), ]
  outer_splits <- list.append(outer_splits, list(train = outer_train, test = outer_test))
}
```

```
save(cpp_data, outer_splits, file = "./rdata/var_reduct_cpp_train_test_splits.RData")
```

2.6 Toxicity (classification)

2.6.1 Generating ML input

```
generate_ml_input(
  fasta_file = "./other_data/toxicity.fasta",
  label_file = "./other_data/tx_labels.csv",
  output_file = "./data/tx_ml_input.csv"
)
```

2.6.2 Variable importance

```
set.seed(12)
tx_data <- read.csv("./data/tx_ml_input.csv", check.names = FALSE, row.names = 1)
# random shuffle
tx_data <- tx_data[sample(1:nrow(tx_data), replace = FALSE), ]
# train random forest using ranger
ntree <- 100000
predictor_variables <- subset(tx_data, select = -category)
response_variable <- factor(tx_data$category, levels = c("non-toxic", "toxic"))
rf <- ranger(
  x = predictor_variables, y = response_variable, num.trees = ntree, sample.fraction = rep(0.5, 2),
  importance = "permutation", verbose = TRUE, scale.permutation.importance = TRUE, seed = 3,
  ↪  keep.inbag = TRUE
)
save(rf, file = "./rdata/var_reduct_tx_rf.RData")
```

2.6.3 Variable reduction

```
load(file = "./rdata/var_reduct_tx_rf.RData")
var_imp <- ranger::importance(rf)
var_imp <- var_imp[order(-var_imp)]
var_imp <- var_imp[var_imp >= 0]
```

```r
set.seed(12)
tx_data <- read.csv("./data/tx_ml_input.csv", check.names = FALSE, row.names = 1)
# random shuffle
tx_data <- tx_data[sample(1:nrow(tx_data), replace = FALSE), ]

ntree <- 1000
var_subset_res <- list()
for (i in 1:length(var_imp)) {
  cat(paste0(i, "\n"))
  predictor_variables <- tx_data[, colnames(tx_data) %in% names(var_imp)[1:i], drop = FALSE]
  response_variable <- factor(tx_data$category, levels = c("non-toxic", "toxic"))
  rf <- ranger(
    x = predictor_variables, y = response_variable, num.trees = ntree, importance = "none", verbose =
    ↪  TRUE, seed = 3,
    probability = TRUE
  )
  var_subset_res <- list.append(var_subset_res, list(
    acc = 1 - rf$prediction.error,
    aic = aic_clf(
      y_true = response_variable,
      y_pred = rf$predictions,
      k = i
    )
  ))
}
save(var_subset_res, file = "./rdata/var_reduct_tx_var_subset_res.RData")
```

```r
load(file = "./rdata/var_reduct_tx_var_subset_res.RData")
acc_vec <- unlist(lapply(var_subset_res, function(x) x$acc))
aic_vec <- unlist(lapply(var_subset_res, function(x) x$aic))
acc_cutoff <- which.max(acc_vec)
aic_cutoff <- which.min(aic_vec)

p3_1 <- ggplot(data = data.frame(x = 1:length(acc_vec), y = acc_vec), aes(x = x, y = y)) +
  geom_point(colour = "black", alpha = 0.3, size = 0.5) +
  geom_vline(xintercept = acc_cutoff, colour = "red", linetype = "dashed") +
  scale_x_continuous(
    breaks = c(1, acc_cutoff, 300, 600, 900, 1076),
```

```
    labels = c("1", "148", "300", "600", "900", "1,076")
  ) +
  scale_y_continuous(
    breaks = c(0.91, 0.92, 0.93, 0.94, 0.95, 0.96),
    labels = c("0.91", "0.92", "0.93", "0.94", "0.95", "0.96")
  ) +
  theme_bw() +
  theme(
    plot.margin = unit(c(10, 10, -10, 10), "pt"),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    strip.background = element_blank(),
    panel.border = element_blank(),
    axis.line = element_line(color = "black"),
    plot.title = element_text(hjust = 0.5),
    axis.title = element_text(colour = "black"),
    axis.text = element_text(colour = "black")
  ) +
  xlab("") +
  ylab("Accuracy") +
  ggtitle("Toxicity")

p3_2 <- ggplot(data = data.frame(x = 1:length(aic_vec), y = aic_vec), aes(x = x, y = y)) +
  geom_point(colour = "black", alpha = 0.3, size = 0.5) +
  geom_vline(xintercept = aic_cutoff, colour = "red", linetype = "dashed") +
  scale_x_continuous(
    breaks = c(aic_cutoff, 300, 600, 900, 1076),
    labels = c("56", "300", "600", "900", "1,076")
  ) +
  scale_y_continuous(
    breaks = c(2000, 3000, 4000, 5000, 6000, 7000),
    labels = c("2,000", "3,000", "4,000", "5,000", "6,000", "7,000")
  ) +
  theme_bw() +
  theme(
    plot.margin = unit(c(-15, 10, 10, 10), "pt"),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    strip.background = element_blank(),
    panel.border = element_blank(),
    axis.line = element_line(color = "black"),
```

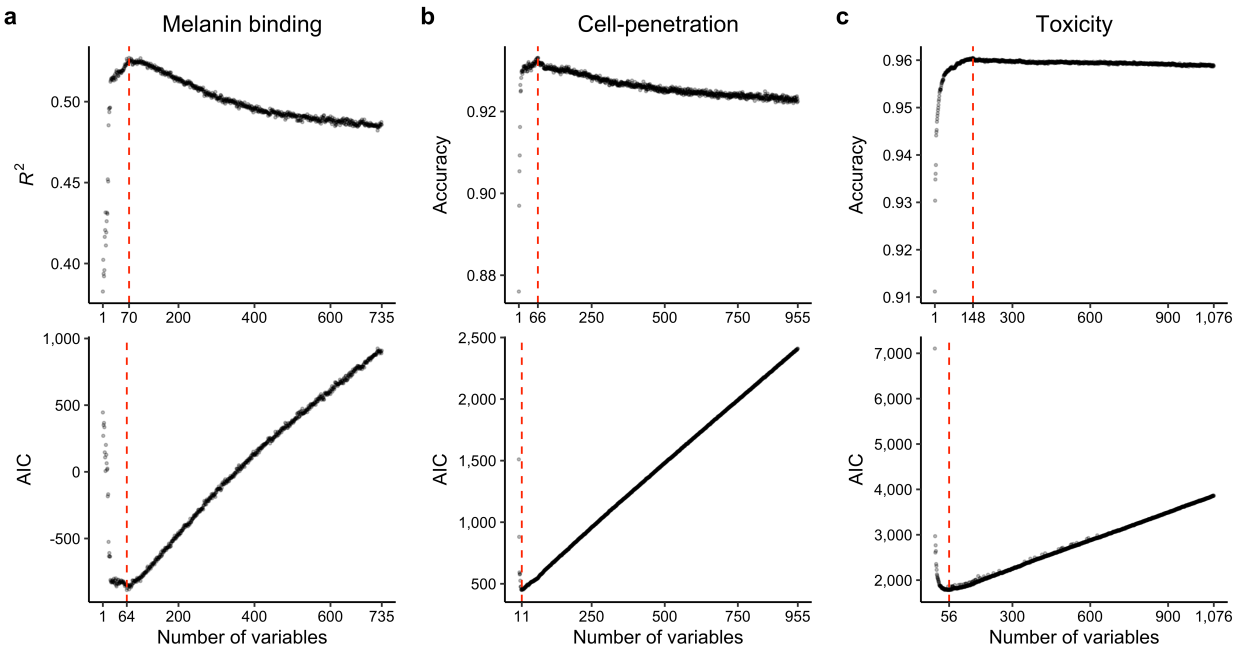
```
    plot.title = element_text(hjust = 0.5),
    axis.title = element_text(colour = "black"),
    axis.text = element_text(colour = "black")
  ) +
  xlab("Number of variables") +
  ylab("AIC") +
  ggtitle("")

cat(paste0("Maximum accuracy: ", formatC(acc_vec[acc_cutoff], format = "f", digits = 3), "\n"))
cat(paste0("Accuracy at AIC cutoff: ", formatC(acc_vec[aic_cutoff], format = "f", digits = 3), "\n"))
```

2.6.4 Generating train-test splits

```
set.seed(12)
tx_data <- read.csv("./data/tx_ml_input.csv", check.names = FALSE, row.names = 1)
# random shuffle
tx_data <- tx_data[sample(1:nrow(tx_data), replace = FALSE), ]
# convert category so that positive class represents non-toxic
# for evaluation metrics that focus on true positives more than
# true negatives
tx_data$category <- as.factor(sapply(tx_data$category, function(x) if (x == "non-toxic") 1 else 0))

cutoff <- aic_cutoff
load(file = "./rdata/var_reduct_tx_rf.RData")
var_imp <- ranger::importance(rf)
var_imp <- var_imp[order(-var_imp)][1:cutoff]
tx_data <- tx_data[, colnames(tx_data) %in% c(names(var_imp), "category")]

outer_splits <- list()
n_train <- round(nrow(tx_data) * 0.9)
set.seed(22)
for (i in 1:10) {
  # random shuffle
  tx_data_ <- tx_data[sample(1:nrow(tx_data), replace = FALSE), ]
  outer_train <- tx_data_[1:n_train, ]
  outer_test <- tx_data_[(n_train + 1):nrow(tx_data_), ]
  outer_splits <- list.append(outer_splits, list(train = outer_train, test = outer_test))
}
save(tx_data, outer_splits, file = "./rdata/var_reduct_tx_train_test_splits.RData")
```

2.7 Plotting

```
p_combined <- plot_grid(p1_1, p2_1, p3_1,
  p1_2, p2_2, p3_2,
  nrow = 2,
  labels = c("a", "b", "c", "", "", ""),
  align = "v"
)
p_combined
```




```
sessionInfo()
```

```
## R version 4.2.2 (2022-10-31)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Big Sur ... 10.16
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRlapack.dylib
##
## locale:
```

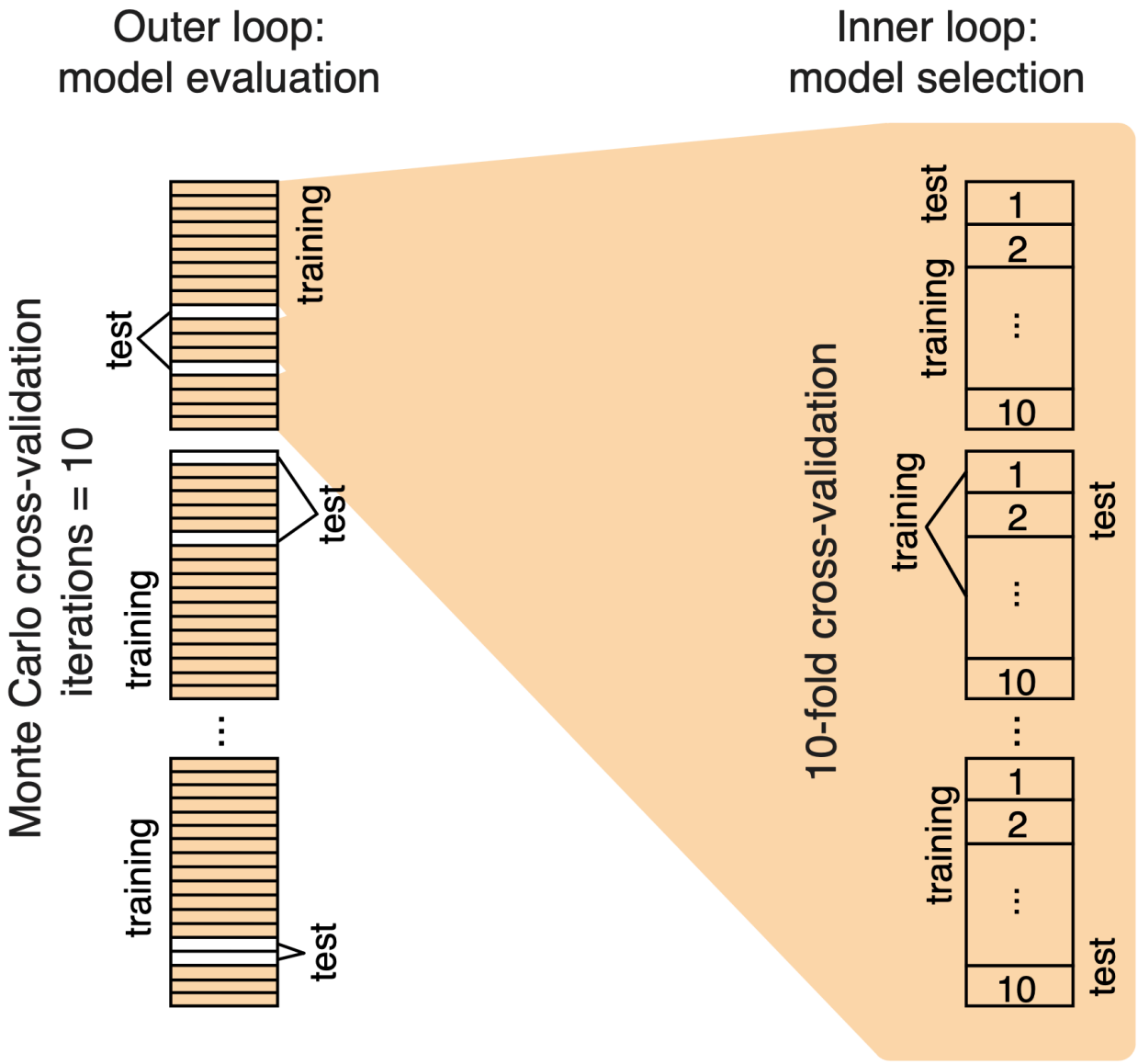
```
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] scales_1.2.1   cowplot_1.1.1  ggplot2_3.3.6  ranger_0.14.1  rlist_0.4.6.2
## [6] protr_1.6-2    Peptides_2.4.4
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.9         pillar_1.8.1       compiler_4.2.2     R.methodsS3_1.8.2
##  [5] R.utils_2.12.0     tools_4.2.2        digest_0.6.29      tibble_3.1.8
##  [9] evaluate_0.16      lifecycle_1.0.1    gtable_0.3.0       R.cache_0.16.0
## [13] lattice_0.20-45    pkgconfig_2.0.3    rlang_1.0.4        Matrix_1.5-1
## [17] DBI_1.1.3          cli_3.3.0          rstudioapi_0.14    yaml_2.3.5
## [21] xfun_0.32          fastmap_1.1.0      withr_2.5.0        dplyr_1.0.9
## [25] styler_1.8.0       stringr_1.4.1      knitr_1.40         generics_0.1.3
## [29] vctrs_0.4.1        tidyselect_1.1.2   grid_4.2.2         glue_1.6.2
## [33] data.table_1.14.2 R6_2.5.1           fansi_1.0.3        rmarkdown_2.16
## [37] bookdown_0.28      purrr_0.3.4        magrittr_2.0.3     codetools_0.2-18
## [41] htmltools_0.5.3    assertthat_0.2.1   colorspace_2.0-3   utf8_1.2.2
## [45] stringi_1.7.8      munsell_0.5.0      R.oo_1.25.0
```

Section 3

Model training

```
library(h2o)
library(MLmetrics)
library(mltools)
library(scales)
library(enrichvs)
library(rlist)
library(stringr)
library(digest)
library(DT)
library(reshape2)
```

3.1 Overview



3.2 General code/functions

```
# h2o.init(nthreads=-1, max_mem_size='100G', port=54321)
h2o.init(nthreads = -1)
h2o.removeAll()
```

```
# DeepLearning Grid 1
deeplearning_params_1 <- list(
  activation = "RectifierWithDropout",
  epochs = 10000, # early stopping
  epsilon = c(1e-6, 1e-7, 1e-8, 1e-9),
  hidden = list(c(50), c(200), c(500)),
  hidden_dropout_ratios = list(c(0.1), c(0.2), c(0.3), c(0.4), c(0.5)),
  input_dropout_ratio = c(0, 0.05, 0.1, 0.15, 0.2),
  rho = c(0.9, 0.95, 0.99)
)
# DeepLearning Grid 2
deeplearning_params_2 <- list(
  activation = "RectifierWithDropout",
  epochs = 10000, # early stopping
  epsilon = c(1e-6, 1e-7, 1e-8, 1e-9),
  hidden = list(c(50, 50), c(200, 200), c(500, 500)),
  hidden_dropout_ratios = list(
    c(0.1, 0.1), c(0.2, 0.2), c(0.3, 0.3),
    c(0.4, 0.4), c(0.5, 0.5)
  ),
  input_dropout_ratio = c(0, 0.05, 0.1, 0.15, 0.2),
  rho = c(0.9, 0.95, 0.99)
)
# DeepLearning Grid 3
deeplearning_params_3 <- list(
  activation = "RectifierWithDropout",
  epochs = 10000, # early stopping
  epsilon = c(1e-6, 1e-7, 1e-8, 1e-9),
  hidden = list(c(50, 50, 50), c(200, 200, 200), c(500, 500, 500)),
  hidden_dropout_ratios = list(
    c(0.1, 0.1, 0.1), c(0.2, 0.2, 0.2),
    c(0.3, 0.3, 0.3), c(0.4, 0.4, 0.4),
    c(0.5, 0.5, 0.5)
  ),
  input_dropout_ratio = c(0, 0.05, 0.1, 0.15, 0.2),
  rho = c(0.9, 0.95, 0.99)
)
# GBM
gbm_params <- list(
  col_sample_rate = c(0.4, 0.7, 1.0),
  col_sample_rate_per_tree = c(0.4, 0.7, 1.0),
```

```
  learn_rate = 0.1,
  max_depth = c(3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17),
  min_rows = c(1, 5, 10, 15, 30, 100),
  min_split_improvement = c(1e-4, 1e-5),
  ntrees = 10000, # early stopping
  sample_rate = c(0.5, 0.6, 0.7, 0.8, 0.9, 1.0)
)
# XGBoost
xgboost_params <- list(
  booster = c("gbtree", "dart"),
  col_sample_rate = c(0.6, 0.8, 1.0),
  col_sample_rate_per_tree = c(0.7, 0.8, 0.9, 1.0),
  max_depth = c(5, 10, 15, 20),
  min_rows = c(0.01, 0.1, 1.0, 3.0, 5.0, 10.0, 15.0, 20.0),
  ntrees = 10000, # early stopping
  reg_alpha = c(0.001, 0.01, 0.1, 1, 10, 100),
  reg_lambda = c(0.001, 0.01, 0.1, 0.5, 1),
  sample_rate = c(0.6, 0.8, 1.0)
)
```

```
balanced_accuracy <- function(y_pred, y_true) {
  return(mean(c(Sensitivity(y_pred, y_true), Specificity(y_pred, y_true)), na.rm = TRUE))
}

sem <- function(x, na.rm = TRUE) sd(x, na.rm) / sqrt(length(na.omit(x)))

statistical_testing <- function(data, metric_vec, decreasing_vec, p_value_threshold = 0.05,
↪  num_entries = 10,
                                output_path = NULL) {
  for (i in 1:length(metric_vec)) {
    metric <- metric_vec[i]
    ranked_models <- rownames(data[order(data[, paste(metric, "mean")], decreasing =
↪  decreasing_vec[i]), ])
    pval <- c()
    for (j in 2:length(ranked_models)) {
      if (sum(is.na(data[ranked_models[j], paste(metric, "CV", 1:10)]))) {
        pval <- c(pval, NA)
      } else {
        pval <- c(pval, wilcox.test(as.numeric(data[ranked_models[1], paste(metric, "CV", 1:10)]),
```

```
          as.numeric(data[ranked_models[j], paste(metric, "CV", 1:10)]),
          exact = FALSE
        )$p.value)
      }
    }
    adj_pval <- c(NA, p.adjust(pval, method = "BH", n = length(pval)))
    df <- data.frame(adj_pval)
    rownames(df) <- ranked_models
    colnames(df) <- paste(metric, "adj. <i>p</i> value")
    data <- merge(data, df, by = "row.names", all = TRUE)
    rownames(data) <- data$Row.names
    data$Row.names <- NULL
  }
  for (i in 1:length(metric_vec)) {
    if (decreasing_vec[i]) {
      data[paste(metric_vec[i], "rank")] <- rank(-data[, paste(metric_vec[i], "mean")], ties.method =
↪  "average")
    } else {
      data[paste(metric_vec[i], "rank")] <- rank(data[, paste(metric_vec[i], "mean")], ties.method =
↪  "average")
    }
  }
  data["Rank sum"] <- rowSums(data[(ncol(data) - length(metric_vec) + 1):ncol(data)])
  data <- data[order(data$`Rank sum`), ]
  competitive <- rowSums(data[paste(metric_vec, "adj. <i>p</i> value")] < p_value_threshold) == 0
  competitive[is.na(competitive)] <- TRUE
  data <- data[competitive, ]
  if (!is.null(output_path)) {
    data[c(
      paste(metric_vec, "adj. <i>p</i> value"), paste(metric_vec, "rank"), "Rank sum",
      melt(sapply(metric_vec, function(x) paste(x, c("mean", "s.e.m."))))$value
    )] %>%
      round(digits = 4) %>%
      write.csv(., file = output_path)
  }
  data[c(paste(metric_vec, "adj. <i>p</i> value"), paste(metric_vec, "rank"), "Rank sum")] %>%
    round(digits = 4) %>%
    datatable(options = list(pageLength = num_entries), escape = FALSE)
}
```

3.3 Melanin binding (regression)

3.3.1 Model training

```
train_mb_models <- function(train_set, exp_dir, prefix, nfolds = 10, grid_seed = 1) {
  tmp <- as.h2o(train_set, destination_frame = prefix)
  y <- "log_intensity"
  x <- setdiff(names(tmp), y)
  res <- as.data.frame(tmp$log_intensity)
  # -------------------
  # base model training
  # -------------------
  cat("Deep learning grid 1\n")
  deeplearning_1 <- h2o.grid(
    algorithm = "deeplearning", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
    keep_cross_validation_predictions = TRUE, hyper_params = deeplearning_params_1,
    stopping_rounds = 3, keep_cross_validation_models = FALSE,
    search_criteria = list(
      strategy = "RandomDiscrete",
      max_models = 100, seed = grid_seed
    ),
    fold_assignment = "Modulo", parallelism = 0
  )
  for (model_id in deeplearning_1@model_ids) {
    tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
  }
  cat("Deep learning grid 2\n")
  deeplearning_2 <- h2o.grid(
    algorithm = "deeplearning", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
    keep_cross_validation_predictions = TRUE, hyper_params = deeplearning_params_2,
    stopping_rounds = 3, keep_cross_validation_models = FALSE,
    search_criteria = list(
      strategy = "RandomDiscrete",
      max_models = 100, seed = grid_seed
    ),
    fold_assignment = "Modulo", parallelism = 0
  )
  for (model_id in deeplearning_2@model_ids) {
    tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
  }
  cat("Deep learning grid 3\n")
```

```
deeplearning_3 <- h2o.grid(
  algorithm = "deeplearning", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, hyper_params = deeplearning_params_3,
  stopping_rounds = 3, keep_cross_validation_models = FALSE,
  search_criteria = list(
    strategy = "RandomDiscrete",
    max_models = 100, seed = grid_seed
  ),
  fold_assignment = "Modulo", parallelism = 0
)
for (model_id in deeplearning_3@model_ids) {
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
}
cat("GBM grid\n")
gbm <- h2o.grid(
  algorithm = "gbm", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, hyper_params = gbm_params, stopping_rounds = 3,
  search_criteria = list(strategy = "RandomDiscrete", max_models = 300, seed = grid_seed),
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo", parallelism = 0
)
for (model_id in gbm@model_ids) {
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
}
cat("GBM 5 default models\n")
gbm_1 <- h2o.gbm(
  model_id = "GBM_1", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8, learn_rate = 0.1,
  max_depth = 6, min_rows = 1, min_split_improvement = 1e-5, ntrees = 10000, sample_rate = 0.8,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
gbm_2 <- h2o.gbm(
  model_id = "GBM_2", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8, learn_rate = 0.1,
  max_depth = 7, min_rows = 10, min_split_improvement = 1e-5, ntrees = 10000, sample_rate = 0.8,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
gbm_3 <- h2o.gbm(
  model_id = "GBM_3", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
```

```
  col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8, learn_rate = 0.1,
  max_depth = 8, min_rows = 10, min_split_improvement = 1e-5, ntrees = 10000, sample_rate = 0.8,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
gbm_4 <- h2o.gbm(
  model_id = "GBM_4", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8, learn_rate = 0.1,
  max_depth = 10, min_rows = 10, min_split_improvement = 1e-5, ntrees = 10000, sample_rate = 0.8,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
gbm_5 <- h2o.gbm(
  model_id = "GBM_5", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8, learn_rate = 0.1,
  max_depth = 15, min_rows = 100, min_split_improvement = 1e-5, ntrees = 10000, sample_rate = 0.8,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
for (model_id in paste0("GBM_", 1:5)) {
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
}
cat("XGBoost grid\n")
xgboost <- h2o.grid(
  algorithm = "xgboost", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, hyper_params = xgboost_params, stopping_rounds = 3,
  search_criteria = list(strategy = "RandomDiscrete", max_models = 300, seed = grid_seed),
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo", parallelism = 0
)
for (model_id in xgboost@model_ids) {
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
}
cat("XGBoost 3 default models\n")
xgboost_1 <- h2o.xgboost(
  model_id = "XGBoost_1", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  booster = "gbtree", col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8,
  max_depth = 10, min_rows = 5, ntrees = 10000, reg_alpha = 0, reg_lambda = 1,
  sample_rate = 0.6, keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
xgboost_2 <- h2o.xgboost(
  model_id = "XGBoost_2", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
```

```
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  booster = "gbtree", col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8,
  max_depth = 20, min_rows = 10, ntrees = 10000, reg_alpha = 0, reg_lambda = 1,
  sample_rate = 0.6, keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
xgboost_3 <- h2o.xgboost(
  model_id = "XGBoost_3", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  booster = "gbtree", col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8,
  max_depth = 5, min_rows = 3, ntrees = 10000, reg_alpha = 0, reg_lambda = 1,
  sample_rate = 0.8, keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
for (model_id in paste0("XGBoost_", 1:3)) {
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
}
cat("GLM\n")
glm <- h2o.glm(
  model_id = "GLM", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, alpha = c(0.0, 0.2, 0.4, 0.6, 0.8, 1.0),
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
tmp_path <- h2o.saveModel(h2o.getModel("GLM"), path = exp_dir, force = TRUE)
cat("DRF\n")
drf <- h2o.randomForest(
  model_id = "DRF", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, ntrees = 10000,
  score_tree_interval = 5, stopping_rounds = 3,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
tmp_path <- h2o.saveModel(h2o.getModel("DRF"), path = exp_dir, force = TRUE)
cat("XRT\n")
xrt <- h2o.randomForest(
  model_id = "XRT", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, ntrees = 10000, histogram_type = "Random",
  score_tree_interval = 5, stopping_rounds = 3,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
tmp_path <- h2o.saveModel(h2o.getModel("XRT"), path = exp_dir, force = TRUE)
# -----------------------
# get holdout predictions
# -----------------------
```

```
base_models <- as.list(c(
  unlist(deeplearning_1@model_ids),
  unlist(deeplearning_2@model_ids),
  unlist(deeplearning_3@model_ids),
  unlist(gbm@model_ids), paste0("GBM_", 1:5),
  unlist(xgboost@model_ids), paste0("XGBoost_", 1:3),
  "GLM",
  "DRF",
  "XRT"
))
for (model_id in base_models) {
  res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_", model_id)),
    col.names = model_id
  ))
}
# ----------------------
# super learner training
# ----------------------
sl_iter <- 0
cat(paste0("Super learner iteration ", sl_iter, " (", length(base_models), " models)\n"))
sl <- h2o.stackedEnsemble(
  x = x, y = y, model_id = paste0("superlearner_iter_", sl_iter),
  training_frame = tmp, seed = 1,
  base_models = base_models,
  metalearner_algorithm = "glm",
  metalearner_nfolds = nfolds,
  keep_levelone_frame = TRUE,
  metalearner_params = list(standardize = TRUE, keep_cross_validation_predictions = TRUE)
)
tmp_path <- h2o.saveModel(h2o.getModel(paste0("superlearner_iter_", sl_iter)), path = exp_dir, force
↪  = TRUE)
model_id <- paste0("metalearner_glm_superlearner_iter_", sl_iter)
tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_", model_id)),
  col.names = paste0(model_id, "_", length(base_models), "_models")
))
# ---------------------------------
# super learner base model reduction
# ---------------------------------
while (TRUE) {
  meta <- h2o.getModel(paste0("metalearner_glm_superlearner_iter_", sl_iter, "_cv_1"))
```

```
    names <- meta@model$coefficients_table[, "names"]
    coeffs <- (meta@model$coefficients_table[, "standardized_coefficients"] > 0)
    for (j in 2:nfolds) {
      meta <- h2o.getModel(paste0("metalearner_glm_superlearner_iter_", sl_iter, "_cv_", j))
      names <- meta@model$coefficients_table[, "names"]
      coeffs <- coeffs + (meta@model$coefficients_table[, "standardized_coefficients"] > 0)
    }
    base_models_ <- as.list(names[coeffs >= ceiling(nfolds / 2) & names != "Intercept"])
    if (length(base_models_) == 0) {
      cat("No base models passing the threshold\n\n")
      break
    }
    if (sum(base_models %in% base_models_) == length(base_models)) {
      cat("No further reduction of base models\n\n")
      break
    }
    sl_iter <- sl_iter + 1
    base_models <- base_models_
    cat(paste0("Super learner iteration ", sl_iter, " (", length(base_models), " models)\n"))
    sl <- h2o.stackedEnsemble(
      x = x, y = y, model_id = paste0("superlearner_iter_", sl_iter),
      training_frame = tmp, seed = 1,
      base_models = base_models,
      metalearner_algorithm = "glm",
      metalearner_nfolds = nfolds,
      keep_levelone_frame = TRUE,
      metalearner_params = list(standardize = TRUE, keep_cross_validation_predictions = TRUE)
    )
    tmp_path <- h2o.saveModel(h2o.getModel(paste0("superlearner_iter_", sl_iter)), path = exp_dir,
↪   force = TRUE)
    model_id <- paste0("metalearner_glm_superlearner_iter_", sl_iter)
    tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
    res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_", model_id)),
      col.names = paste0(model_id, "_", length(base_models), "_models")
    ))
  }
  # ----------------------------------------
  # super learner for homogeneous base models
  # ----------------------------------------
  # DeepLearning
  base_models <- as.list(c(
```

```
  unlist(deeplearning_1@model_ids),
  unlist(deeplearning_2@model_ids),
  unlist(deeplearning_3@model_ids)
))
cat(paste0("Super learner deep learning (", length(base_models), " models)\n"))
sl <- h2o.stackedEnsemble(
  x = x, y = y, model_id = "superlearner_deeplearning",
  training_frame = tmp, seed = 1,
  base_models = base_models,
  metalearner_algorithm = "glm",
  metalearner_nfolds = nfolds,
  keep_levelone_frame = TRUE,
  metalearner_params = list(standardize = TRUE, keep_cross_validation_predictions = TRUE)
)
tmp_path <- h2o.saveModel(h2o.getModel("superlearner_deeplearning"), path = exp_dir, force = TRUE)
model_id <- "metalearner_glm_superlearner_deeplearning"
tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_", model_id)),
  col.names = paste0(model_id, "_", length(base_models), "_models")
))
# GBM
base_models <- as.list(c(unlist(gbm@model_ids), paste0("GBM_", 1:5)))
cat(paste0("Super learner GBM (", length(base_models), " models)\n"))
sl <- h2o.stackedEnsemble(
  x = x, y = y, model_id = "superlearner_gbm",
  training_frame = tmp, seed = 1,
  base_models = base_models,
  metalearner_algorithm = "glm",
  metalearner_nfolds = nfolds,
  keep_levelone_frame = TRUE,
  metalearner_params = list(standardize = TRUE, keep_cross_validation_predictions = TRUE)
)
tmp_path <- h2o.saveModel(h2o.getModel("superlearner_gbm"), path = exp_dir, force = TRUE)
model_id <- "metalearner_glm_superlearner_gbm"
tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_", model_id)),
  col.names = paste0(model_id, "_", length(base_models), "_models")
))
# XGBoost
base_models <- as.list(c(unlist(xgboost@model_ids), paste0("XGBoost_", 1:3)))
cat(paste0("Super learner XGBoost (", length(base_models), " models)\n"))
```

```
  sl <- h2o.stackedEnsemble(
    x = x, y = y, model_id = "superlearner_xgboost",
    training_frame = tmp, seed = 1,
    base_models = base_models,
    metalearner_algorithm = "glm",
    metalearner_nfolds = nfolds,
    keep_levelone_frame = TRUE,
    metalearner_params = list(standardize = TRUE, keep_cross_validation_predictions = TRUE)
  )
  tmp_path <- h2o.saveModel(h2o.getModel("superlearner_xgboost"), path = exp_dir, force = TRUE)
  model_id <- "metalearner_glm_superlearner_xgboost"
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
  res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_", model_id)),
    col.names = paste0(model_id, "_", length(base_models), "_models")
  ))
  write.csv(res, file = paste0(exp_dir, "/cv_holdout_predictions.csv"), row.names = FALSE)
  cat("\n\n")
  h2o.removeAll()
}
```

3.3.2 Inner cross-validation

```
load(file = "./rdata/var_reduct_mb_train_test_splits.RData")
for (i in 1:10) {
  cat(paste0("Outer training set ", i, "\n"))
  prefix <- paste0("outer_", i)
  exp_dir <- paste0("/Users/renee/Downloads/melanin_binding/", prefix)
  dir.create(exp_dir)
  train_mb_models(train_set = outer_splits[[i]][["train"]], exp_dir = exp_dir, prefix = prefix)
}
```

3.3.3 Training on whole data set

```
load(file = "./rdata/var_reduct_mb_train_test_splits.RData")
prefix <- "whole_data_set"
exp_dir <- paste0("/Users/renee/Downloads/melanin_binding/", prefix)
dir.create(exp_dir)
```

```
train_mb_models(train_set = mb_data, exp_dir = exp_dir, prefix = prefix)
```

```
# Keep track of grid models
dl_grid <- list()
gbm_grid <- list()
xgboost_grid <- list()
dl_grid_params <- list()
gbm_grid_params <- list()
xgboost_grid_params <- list()
for (i in 1:11) {
  if (i == 11) {
    cat(paste0("Whole data set\n"))
    prefix <- "whole_data_set"
  } else {
    cat(paste0("Outer training set ", i, "\n"))
    prefix <- paste0("outer_", i)
  }
  dir <- paste0("/Users/renee/Downloads/melanin_binding/", prefix)
  files <- list.files(dir)
  # Deep learning
  dl <- files[str_detect(files, "DeepLearning_model")]
  for (m in dl) {
    model <- h2o.loadModel(paste0(dir, "/", m))
    hs <- sha1(paste(c(
      model@allparameters$epsilon,
      model@allparameters$hidden,
      model@allparameters$hidden_dropout_ratios,
      model@allparameters$input_dropout_ratio,
      model@allparameters$rho
    ), collapse = " "))
    if (hs %in% names(dl_grid)) {
      dl_grid[[hs]] <- c(dl_grid[[hs]], m)
    } else {
      dl_grid[[hs]] <- c(m)
      dl_grid_params <- list.append(
        dl_grid_params,
        c(
          "epsilon" = model@allparameters$epsilon,
          "hidden" = paste0("[", paste(model@allparameters$hidden, collapse = ","), "]"),
```

```
        "hidden_dropout_ratios" = paste0(
          "[",
          paste(model@allparameters$hidden_dropout_ratios,
            collapse = ","
          ), "]"
        ),
        "input_dropout_ratio" = model@allparameters$input_dropout_ratio,
        "rho" = model@allparameters$rho
      )
    )
  }
}
h2o.removeAll()
# GBM
gbm <- files[str_detect(files, "GBM_model")]
for (m in gbm) {
  model <- h2o.loadModel(paste0(dir, "/", m))
  hs <- sha1(paste(c(
    model@allparameters$col_sample_rate,
    model@allparameters$col_sample_rate_per_tree,
    model@allparameters$max_depth,
    model@allparameters$min_rows,
    model@allparameters$min_split_improvement,
    model@allparameters$sample_rate
  ), collapse = " "))
  if (hs %in% names(gbm_grid)) {
    gbm_grid[[hs]] <- c(gbm_grid[[hs]], m)
  } else {
    gbm_grid[[hs]] <- c(m)
    gbm_grid_params <- list.append(
      gbm_grid_params,
      c(
        "col_sample_rate" = model@allparameters$col_sample_rate,
        "col_sample_rate_per_tree" = model@allparameters$col_sample_rate_per_tree,
        "max_depth" = model@allparameters$max_depth,
        "min_rows" = model@allparameters$min_rows,
        "min_split_improvement" = model@allparameters$min_split_improvement,
        "sample_rate" = model@allparameters$sample_rate
      )
    )
  }
```

```
  }
  h2o.removeAll()
  # XGBoost
  xgboost <- files[str_detect(files, "XGBoost_model")]
  for (m in xgboost) {
    model <- h2o.loadModel(paste0(dir, "/", m))
    hs <- sha1(paste(c(
      model@allparameters$booster,
      model@allparameters$col_sample_rate,
      model@allparameters$col_sample_rate_per_tree,
      model@allparameters$max_depth,
      model@allparameters$min_rows,
      model@allparameters$reg_alpha,
      model@allparameters$reg_lambda,
      model@allparameters$sample_rate
    ), collapse = " "))
    if (hs %in% names(xgboost_grid)) {
      xgboost_grid[[hs]] <- c(xgboost_grid[[hs]], m)
    } else {
      xgboost_grid[[hs]] <- c(m)
      xgboost_grid_params <- list.append(
        xgboost_grid_params,
        c(
          "booster" = model@allparameters$booster,
          "col_sample_rate" = model@allparameters$col_sample_rate,
          "col_sample_rate_per_tree" = model@allparameters$col_sample_rate_per_tree,
          "max_depth" = model@allparameters$max_depth,
          "min_rows" = model@allparameters$min_rows,
          "reg_alpha" = model@allparameters$reg_alpha,
          "reg_lambda" = model@allparameters$reg_lambda,
          "sample_rate" = model@allparameters$sample_rate
        )
      )
    }
  }
  h2o.removeAll()
}

dl_grid_params <- as.data.frame(t(data.frame(dl_grid_params)))
rownames(dl_grid_params) <- paste("Neural network grid model", 1:nrow(dl_grid_params))
```

```
gbm_grid_params <- as.data.frame(t(data.frame(gbm_grid_params)))
rownames(gbm_grid_params) <- paste("GBM grid model", 1:nrow(gbm_grid_params))

xgboost_grid_params <- as.data.frame(t(data.frame(xgboost_grid_params)))
rownames(xgboost_grid_params) <- paste("XGBoost grid model", 1:nrow(xgboost_grid_params))

write.csv(dl_grid_params, "./other_data/melanin_binding/neural_network_grid_params.csv")
write.csv(gbm_grid_params, "./other_data/melanin_binding/gbm_grid_params.csv")
write.csv(xgboost_grid_params, "./other_data/melanin_binding/xgboost_grid_params.csv")

save(dl_grid, gbm_grid, xgboost_grid, file = "./rdata/model_training_grid_models_mb.RData")
```

3.3.4 Model evaluation

```
model_evaluation <- function(holdout_pred, fold_asign, grid_name, grid_meta = NULL) {
  load(file = "./rdata/var_reduct_mb_train_test_splits.RData")
  res_ <- list()
  model_num <- c()
  if (startsWith(grid_name, "Super learner")) {
    if (grid_name == "Super learner all models") {
      m <- colnames(holdout_pred)[startsWith(colnames(holdout_pred),
↪  "metalearner_glm_superlearner_iter_0")]
    } else if (grid_name == "Super learner final") {
      sl_models <- colnames(holdout_pred)[startsWith(colnames(holdout_pred),
↪  "metalearner_glm_superlearner_iter")]
      m <- sl_models[length(sl_models)]
    } else if (grid_name == "Super learner neural network") {
      m <- colnames(holdout_pred)[startsWith(colnames(holdout_pred),
↪  "metalearner_glm_superlearner_deeplearning")]
    } else if (grid_name == "Super learner GBM") {
      m <- colnames(holdout_pred)[startsWith(colnames(holdout_pred),
↪  "metalearner_glm_superlearner_gbm")]
    } else if (grid_name == "Super learner XGBoost") {
      m <- colnames(holdout_pred)[startsWith(colnames(holdout_pred),
↪  "metalearner_glm_superlearner_xgboost")]
    }
    mae <- c()
    rmse <- c()
    R2 <- c()
```

```
    for (k in 1:10) {
      y_true <- holdout_pred[fold_assign == k, "log_intensity"]
      y_pred <- holdout_pred[fold_assign == k, m]
      mae <- c(mae, MAE(y_pred = y_pred, y_true = y_true))
      rmse <- c(rmse, RMSE(y_pred = y_pred, y_true = y_true))
      R2 <- c(R2, R2_Score(y_pred = y_pred, y_true = y_true))
    }
    # Re-scale to 0-100
    mae <- rescale(mae, to = c(0, 100), from = range(mb_data$log_intensity))
    rmse <- rescale(rmse, to = c(0, 100), from = range(mb_data$log_intensity))
    res_ <- list.append(res_, c(
      mean(mae, na.rm = TRUE), sem(mae),
      mean(rmse, na.rm = TRUE), sem(rmse),
      mean(R2, na.rm = TRUE), sem(R2),
      mae, rmse, R2
    ))
  } else {
    for (j in 1:length(grid_meta)) {
      g <- grid_meta[[j]]
      m <- intersect(colnames(holdout_pred), g)
      if (length(m) == 1) {
        model_num <- c(model_num, j)
        mae <- c()
        rmse <- c()
        R2 <- c()
        for (k in 1:10) {
          y_true <- holdout_pred[fold_assign == k, "log_intensity"]
          y_pred <- holdout_pred[fold_assign == k, m]
          mae <- c(mae, MAE(y_pred = y_pred, y_true = y_true))
          rmse <- c(rmse, RMSE(y_pred = y_pred, y_true = y_true))
          R2 <- c(R2, R2_Score(y_pred = y_pred, y_true = y_true))
        }
        # Re-scale to 0-100
        mae <- rescale(mae, to = c(0, 100), from = range(mb_data$log_intensity))
        rmse <- rescale(rmse, to = c(0, 100), from = range(mb_data$log_intensity))
        res_ <- list.append(res_, c(
          mean(mae, na.rm = TRUE), sem(mae),
          mean(rmse, na.rm = TRUE), sem(rmse),
          mean(R2, na.rm = TRUE), sem(R2),
          mae, rmse, R2
        ))
```

```
      }
    }
  }
  res <- as.data.frame(t(data.frame(res_)))
  colnames(res) <- c(
    "Norm. MAE mean", "Norm. MAE s.e.m.", "Norm. RMSE mean", "Norm. RMSE s.e.m.",
    "R^2 mean", "R^2 s.e.m.",
    paste("Norm. MAE CV", 1:10), paste("Norm. RMSE CV", 1:10), paste("R^2 CV", 1:10)
  )
  if (nrow(res) == 1) {
    rownames(res) <- grid_name
  } else {
    rownames(res) <- paste(grid_name, model_num)
  }
  return(res)
}
```

3.3.5 Inner loop model selection

```
load(file = "./rdata/model_training_grid_models_mb.RData")

for (i in 1:10) {
  holdout_pred <- read.csv(paste0("./other_data/melanin_binding/outer_", i,
↪  "/cv_holdout_predictions.csv"))
  fold_assign <- rep(1:10, ceiling(nrow(holdout_pred) / 10))[1:nrow(holdout_pred)]
  data_ <- list()
  # Deep learning grid models
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
    grid_name = "Neural network grid model",
    grid_meta = dl_grid
  ))
  # GBM grid models
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
    grid_name = "GBM grid model",
    grid_meta = gbm_grid
  ))
  # GBM default models
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
    grid_name = "GBM model",
```

```
    grid_meta = as.list(paste0("GBM_", 1:5))
  ))
  # XGBoost grid models
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
    grid_name = "XGBoost grid model",
    grid_meta = xgboost_grid
  ))
  # XGBoost default models
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
    grid_name = "XGBoost model",
    grid_meta = as.list(paste0("XGBoost_", 1:3))
  ))
  # GLM
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "GLM model",
↪   grid_meta = list("GLM")))
  # DRF
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "DRF model",
↪   grid_meta = list("DRF")))
  # XRT
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "XRT model",
↪   grid_meta = list("XRT")))
  # Super learner all
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪   all models"))
  # Super learner final
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪   final"))
  # Super learner deep learning
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪   neural network"))
  # Super learner GBM
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪   GBM"))
  # Super learner XGBoost
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪   XGBoost"))

  data <- do.call(rbind, data_)
  write.csv(data, paste0("./other_data/melanin_binding/outer_", i, "/cv_res.csv"))
}
```

3.3.5.1 CV 1

Show 1 entries Search:

	Norm. MAE adj. *p* value	Norm. RMSE adj. *p* value	R^2 adj. *p* value	Norm. MAE rank	Norm. RMSE rank	R^2 rank	Rank sum
Super learner final	0.1871			6	1	1	8

Showing 1 to 1 of 2 entries Previous 1 2 Next

3.3.5.2 CV 2

Show 1 entries Search:

	Norm. MAE adj. *p* value	Norm. RMSE adj. *p* value	R^2 adj. *p* value	Norm. MAE rank	Norm. RMSE rank	R^2 rank	Rank sum
Super learner final	0.5227			6	1	1	8

Showing 1 to 1 of 50 entries Previous 1 2 3 4 5 … 50 Next

3.3.5.3 CV 3

Show 1 entries Search:

	Norm. MAE adj. *p* value	Norm. RMSE adj. *p* value	R^2 adj. *p* value	Norm. MAE rank	Norm. RMSE rank	R^2 rank	Rank sum
Super learner final	0.1423			4	1	1	6

Showing 1 to 1 of 15 entries Previous 1 2 3 4 5 … 15 Next

3.3.5.4 CV 4

Show 1 entries Search:

	Norm. MAE adj. *p* value	Norm. RMSE adj. *p* value	R^2 adj. *p* value	Norm. MAE rank	Norm. RMSE rank	R^2 rank	Rank sum
Super learner final	0.2447			6	1	1	8

Showing 1 to 1 of 43 entries Previous 1 2 3 4 5 … 43 Next

3.3.5.5 CV 5

Show 1 entries Search:

	Norm. MAE adj. *p* value	Norm. RMSE adj. *p* value	R^2 adj. *p* value	Norm. MAE rank	Norm. RMSE rank	R^2 rank	Rank sum
Super learner final				1	1	1	3

Showing 1 to 1 of 41 entries Previous 1 2 3 4 5 … 41 Next

3.3.5.6 CV 6

Show 1 entries Search:

	Norm. MAE adj. *p* value	Norm. RMSE adj. *p* value	R^2 adj. *p* value	Norm. MAE rank	Norm. RMSE rank	R^2 rank	Rank sum
Super learner final	0.6795			4	1	1	6

Showing 1 to 1 of 49 entries Previous 1 2 3 4 5 … 49 Next

3.3.5.7 CV 7

Show 1 entries Search:

	Norm. MAE adj. *p* value	Norm. RMSE adj. *p* value	R^2 adj. *p* value	Norm. MAE rank	Norm. RMSE rank	R^2 rank	Rank sum
Super learner final	0.3451			4	1	1	6

Showing 1 to 1 of 19 entries Previous 1 2 3 4 5 … 19 Next

3.3.5.8 CV 8

Show 1 entries Search:

	Norm. MAE adj. *p* value	Norm. RMSE adj. *p* value	R^2 adj. *p* value	Norm. MAE rank	Norm. RMSE rank	R^2 rank	Rank sum
Super learner final	0.2142			6	1	1	8

Showing 1 to 1 of 33 entries Previous 1 2 3 4 5 … 33 Next

3.3.5.9 CV 9

Show 1 entries Search:

	Norm. MAE adj. *p* value	Norm. RMSE adj. *p* value	R^2 adj. *p* value	Norm. MAE rank	Norm. RMSE rank	R^2 rank	Rank sum
Super learner final	0.1057			9	1	1	11

Showing 1 to 1 of 32 entries Previous 1 2 3 4 5 … 32 Next

3.3.5.10 CV 10

Show 1 entries Search:

	Norm. MAE adj. *p* value	Norm. RMSE adj. *p* value	R^2 adj. *p* value	Norm. MAE rank	Norm. RMSE rank	R^2 rank	Rank sum
Super learner final	0.0323			9	1	1	11

Showing 1 to 1 of 26 entries Previous 1 2 3 4 5 … 26 Next

3.3.6 Final evaluation

```
load(file = "./rdata/var_reduct_mb_train_test_splits.RData")
dir <- "/Users/renee/Downloads/melanin_binding"
selected_models <- c(
  "Super learner final", "Super learner final", "Super learner final",
  "Super learner final", "Super learner final", "Super learner final",
  "Super learner final", "Super learner final", "Super learner final",
  "Super learner final"
)
mae <- c()
rmse <- c()
R2 <- c()
for (i in 1:10) {
  holdout_pred <- read.csv(paste0("./other_data/melanin_binding/outer_", i,
↪  "/cv_holdout_predictions.csv"))
  if (startsWith(selected_models[i], "Neural network grid model")) {
    grid_num <- as.integer(str_extract(selected_models[i], "[0-9]+"))
    m <- intersect(colnames(holdout_pred), dl_grid[[grid_num]])
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/", m))
  } else if (startsWith(selected_models[i], "GBM grid model")) {
    grid_num <- as.integer(str_extract(selected_models[i], "[0-9]+"))
```

```
    m <- intersect(colnames(holdout_pred), gbm_grid[[grid_num]])
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/", m))
  } else if (startsWith(selected_models[i], "GBM model")) {
    grid_num <- as.integer(str_extract(selected_models[i], "[0-9]+"))
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/GBM_", grid_num))
  } else if (startsWith(selected_models[i], "XGBoost grid model")) {
    grid_num <- as.integer(str_extract(selected_models[i], "[0-9]+"))
    m <- intersect(colnames(holdout_pred), xgboost_grid[[grid_num]])
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/", m))
  } else if (startsWith(selected_models[i], "XGBoost model")) {
    grid_num <- as.integer(str_extract(selected_models[i], "[0-9]+"))
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/XGBoost_", grid_num))
  } else if (selected_models[i] == "Super learner all models") {
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/superlearner_iter_0"))
  } else if (selected_models[i] == "Super learner final") {
    files <- list.files(paste0(dir, "/outer_", i))
    files <- files[startsWith(files, "superlearner_iter")]
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/", files[length(files)]))
  } else if (selected_models[i] == "Super learner neural network") {
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/superlearner_deeplearning"))
  } else if (selected_models[i] == "Super learner GBM") {
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/superlearner_gbm"))
  } else if (selected_models[i] == "Super learner XGBoost") {
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/superlearner_xgboost"))
  } else {
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/", unlist(strsplit(selected_models[i], "
↪  "))[1]))
  }
  tmp <- as.h2o(subset(outer_splits[[i]][["test"]], select = -log_intensity))
  y_true <- outer_splits[[i]][["test"]]$log_intensity
  y_pred <- as.data.frame(as.data.frame(h2o.predict(model, tmp)))[, 1]
  mae <- c(mae, MAE(y_pred = y_pred, y_true = y_true))
  rmse <- c(rmse, RMSE(y_pred = y_pred, y_true = y_true))
  R2 <- c(R2, R2_Score(y_pred = y_pred, y_true = y_true))
}

# Re-scale to 0-100
mae <- rescale(mae, to = c(0, 100), from = range(mb_data$log_intensity))
rmse <- rescale(rmse, to = c(0, 100), from = range(mb_data$log_intensity))

h2o.removeAll()
```

```
save(mae, rmse, R2, file = "./rdata/final_evaluation_mb.RData")
```

```
load(file = "./rdata/final_evaluation_mb.RData")
data.frame(
  Metric = c("Norm. MAE", "Norm. RMSE", "R<sup>2</sup>"),
  `Mean ± s.e.m.` = c(
    paste0(round(mean(mae), 3), " ± ", round(sem(mae), 3)),
    paste0(round(mean(rmse), 3), " ± ", round(sem(rmse), 3)),
    paste0(round(mean(R2), 3), " ± ", round(sem(R2), 3))
  ),
  check.names = FALSE
) %>%
  datatable(escape = FALSE, rownames = FALSE)
```

Show 10 entries Search:

Metric	Mean ± s.e.m.
Norm. MAE	16.812 ± 0.166
Norm. RMSE	22.782 ± 0.222
R^2	0.543 ± 0.007

Showing 1 to 3 of 3 entries Previous 1 Next

3.3.7 Final model selection

```
load(file = "./rdata/model_training_grid_models_mb.RData")

holdout_pred <- read.csv("./other_data/melanin_binding/whole_data_set/cv_holdout_predictions.csv")
fold_assign <- rep(1:10, ceiling(nrow(holdout_pred) / 10))[1:nrow(holdout_pred)]
data_ <- list()
# Deep learning grid models
data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
  grid_name = "Neural network grid model",
  grid_meta = dl_grid
))
# GBM grid models
data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
  grid_name = "GBM grid model",
  grid_meta = gbm_grid
```

```
))
# GBM default models
data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
  grid_name = "GBM model",
  grid_meta = as.list(paste0("GBM_", 1:5))
))
# XGBoost grid models
data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
  grid_name = "XGBoost grid model",
  grid_meta = xgboost_grid
))
# XGBoost default models
data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
  grid_name = "XGBoost model",
  grid_meta = as.list(paste0("XGBoost_", 1:3))
))
# GLM
data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "GLM model",
↪  grid_meta = list("GLM")))
# DRF
data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "DRF model",
↪  grid_meta = list("DRF")))
# XRT
data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "XRT model",
↪  grid_meta = list("XRT")))
# Super learner all
data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner all
↪  models"))
# Super learner final
data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪  final"))
# Super learner deep learning
data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪  neural network"))
# Super learner GBM
data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪  GBM"))
# Super learner XGBoost
data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪  XGBoost"))
```

```
data <- do.call(rbind, data_)
write.csv(data, "./other_data/melanin_binding/whole_data_set/cv_res.csv")
```

```
data <- read.csv("./other_data/melanin_binding/whole_data_set/cv_res.csv", row.names = 1, check.names
↪  = FALSE)
statistical_testing(data, metric_vec = c("Norm. MAE", "Norm. RMSE", "R^2"), decreasing_vec = c(FALSE,
↪  FALSE, TRUE))
```

Show 10 entries Search:

	Norm. MAE adj. *p* value	Norm. RMSE adj. *p* value	R^2 adj. *p* value	Norm. MAE rank	Norm. RMSE rank	R^2 rank	Rank sum
Super learner final	0.4742			5	1	1	7
Super learner neural network	0.1875	0.3455	0.3855	13	3	3	19
Neural network grid model 154	0.1235	0.3455	0.3466	16	5	5	26
Neural network grid model 45	0.0775	0.3092	0.2801	18	10	10	38
Neural network grid model 210	0.0775	0.3092	0.4278	32	4	4	40
Neural network grid model 201	0.1428	0.3455	0.3466	25	8	8	41
Neural network grid model 203	0.8501	0.3092	0.3119	2	25	28	55
Neural network grid model 146	0.5714	0.249	0.2801	4	28	32	64
Neural network grid model 145	0.1235	0.2236	0.2801	26	20	20	66
Neural network grid model 182	0.3462	0.3092	0.2801	6	35	37	78

Showing 1 to 10 of 31 entries Previous 1 2 3 4 Next

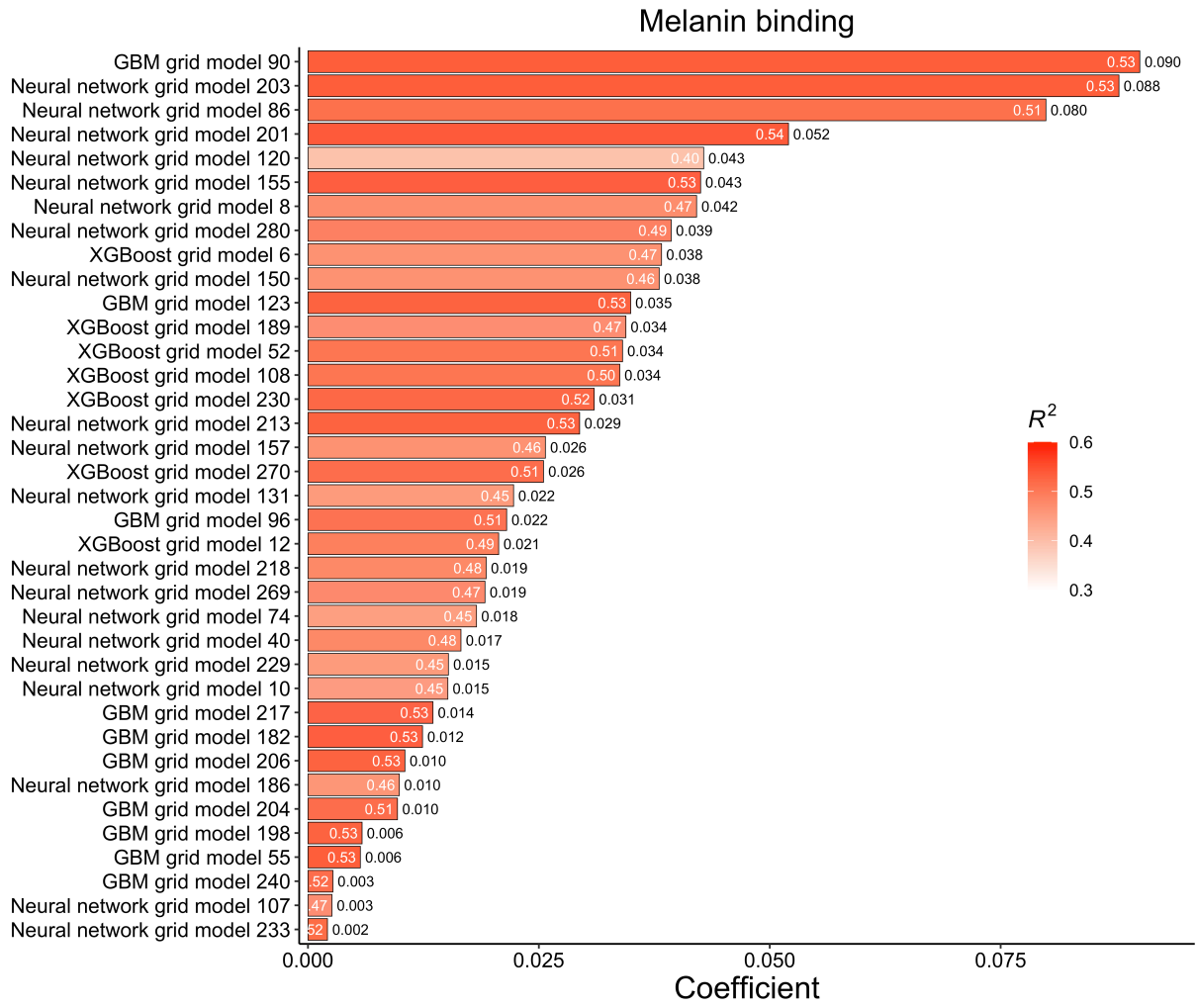
```
# statistical_testing(data, metric_vec=c('Norm. MAE', 'Norm. RMSE', 'R^2'), decreasing_vec=c(FALSE,
↪  FALSE, TRUE),
#
↪  output_path='./other_data/melanin_binding/whole_data_set/cv_res_statistical_testing.csv')
```

3.3.8 Final reduced SL

```
h2o.init(nthreads = -1)
model_path <- "/Users/renee/Downloads/melanin_binding/whole_data_set/superlearner_iter_3"
model <- h2o.loadModel(model_path)
load(file = "./rdata/model_training_grid_models_mb.RData")
data <- read.csv("./other_data/melanin_binding/whole_data_set/cv_res.csv", row.names = 1, check.names
↪  = FALSE)
df <- as.data.frame(model@model$metalearner_model@model$coefficients_table)[c("names",
↪  "standardized_coefficients")][-1, ]

for (i in 1:nrow(df)) {
  name <- df$names[i]
  if (startsWith(name, "DeepLearning_model")) {
    for (j in 1:length(dl_grid)) {
      if (name %in% dl_grid[[j]]) break
    }
    df$names[i] <- paste("Neural network grid model", j)
  } else if (startsWith(name, "GBM_model")) {
    for (j in 1:length(gbm_grid)) {
      if (name %in% gbm_grid[[j]]) break
    }
    df$names[i] <- paste("GBM grid model", j)
  } else if (startsWith(name, "XGBoost_model")) {
    for (j in 1:length(xgboost_grid)) {
      if (name %in% xgboost_grid[[j]]) break
    }
    df$names[i] <- paste("XGBoost grid model", j)
  } else {
    df$names[i] <- paste(strsplit(name, "_")[[1]], collapse = " model ")
  }
}

rownames(df) <- df$names
df <- merge(df, data["R^2 mean"], by = "row.names")[, -1]
df <- df[df$standardized_coefficients > 0, ]
```

```
p1 <- ggplot(df, aes(x = reorder(names, standardized_coefficients), y = standardized_coefficients,
↪  fill = `R^2 mean`)) +
  geom_col(color = "black", size = 0.2) +
```

```
scale_fill_gradient(low = "white", high = "red", n.breaks = 4, limits = c(0.30, 0.60), breaks =
↪  c(0.3, 0.4, 0.5, 0.6)) +
geom_text(aes(label = sprintf("%.2f", `R^2 mean`)), nudge_y = -0.002, size = 3, color = "white") +
geom_text(aes(label = sprintf("%.3f", standardized_coefficients)), nudge_y = 0.0025, size = 3) +
coord_flip() +
scale_y_continuous(limits = c(0, max(df$standardized_coefficients) + 0.005), expand = c(0.01, 0)) +
theme_bw() +
theme(
  panel.grid.major = element_blank(),
  panel.grid.minor = element_blank(),
  strip.background = element_blank(),
  panel.border = element_blank(),
  axis.line = element_line(color = "black"),
  legend.text = element_text(colour = "black", size = 10),
  plot.title = element_text(hjust = 0.5, size = 18),
  plot.margin = ggplot2::margin(10, 10, 10, 0, "pt"),
  axis.title.x = element_text(colour = "black", size = 18),
  axis.title.y = element_text(colour = "black", size = 18),
  axis.text.x = element_text(colour = "black", size = 12),
  axis.text.y = element_text(colour = "black", size = 12),
  legend.title = element_text(size = 14),
  legend.position = c(0.85, 0.5)
) +
guides(fill = guide_colorbar(title = expression(italic("R"^2)))) +
xlab("") +
ylab("Coefficient") +
ggtitle("Melanin binding")
```



3.4 Cell-penetration (classification)

3.4.1 Model training

```
train_cpp_models <- function(train_set, exp_dir, prefix, nfolds = 10, grid_seed = 1) {
  tmp <- as.h2o(train_set, destination_frame = prefix)
  tmp["category"] <- as.factor(tmp["category"])
  y <- "category"
  x <- setdiff(names(tmp), y)
  res <- as.data.frame(tmp$category)
```

```
samp_factors <- as.vector(mean(table(train_set$category)) / table(train_set$category))
# -------------------
# base model training
# -------------------
cat("Deep learning grid 1\n")
deeplearning_1 <- h2o.grid(
  algorithm = "deeplearning", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, hyper_params = deeplearning_params_1,
  stopping_rounds = 3, balance_classes = TRUE, class_sampling_factors = samp_factors,
  search_criteria = list(
    strategy = "RandomDiscrete",
    max_models = 100, seed = grid_seed
  ),
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo", parallelism = 0
)
for (model_id in deeplearning_1@model_ids) {
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
}
cat("GBM grid\n")
gbm <- h2o.grid(
  algorithm = "gbm", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, hyper_params = gbm_params, stopping_rounds = 3,
  balance_classes = TRUE, class_sampling_factors = samp_factors,
  search_criteria = list(strategy = "RandomDiscrete", max_models = 100, seed = grid_seed),
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo", parallelism = 0
)
for (model_id in gbm@model_ids) {
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
}
cat("GBM 5 default models\n")
gbm_1 <- h2o.gbm(
  model_id = "GBM_1", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  balance_classes = TRUE, class_sampling_factors = samp_factors,
  col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8, learn_rate = 0.1,
  max_depth = 6, min_rows = 1, min_split_improvement = 1e-5, ntrees = 10000, sample_rate = 0.8,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
gbm_2 <- h2o.gbm(
  model_id = "GBM_2", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
```

```
  balance_classes = TRUE, class_sampling_factors = samp_factors,
  col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8, learn_rate = 0.1,
  max_depth = 7, min_rows = 10, min_split_improvement = 1e-5, ntrees = 10000, sample_rate = 0.8,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
gbm_3 <- h2o.gbm(
  model_id = "GBM_3", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  balance_classes = TRUE, class_sampling_factors = samp_factors,
  col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8, learn_rate = 0.1,
  max_depth = 8, min_rows = 10, min_split_improvement = 1e-5, ntrees = 10000, sample_rate = 0.8,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
gbm_4 <- h2o.gbm(
  model_id = "GBM_4", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  balance_classes = TRUE, class_sampling_factors = samp_factors,
  col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8, learn_rate = 0.1,
  max_depth = 10, min_rows = 10, min_split_improvement = 1e-5, ntrees = 10000, sample_rate = 0.8,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
gbm_5 <- h2o.gbm(
  model_id = "GBM_5", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  balance_classes = TRUE, class_sampling_factors = samp_factors,
  col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8, learn_rate = 0.1,
  max_depth = 15, min_rows = 100, min_split_improvement = 1e-5, ntrees = 10000, sample_rate = 0.8,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
for (model_id in paste0("GBM_", 1:5)) {
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
}
cat("XGBoost grid\n")
xgboost <- h2o.grid(
  algorithm = "xgboost", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, hyper_params = xgboost_params, stopping_rounds = 3,
  search_criteria = list(strategy = "RandomDiscrete", max_models = 100, seed = grid_seed),
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo", parallelism = 0
)
for (model_id in xgboost@model_ids) {
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
```

```
}
cat("XGBoost 3 default models\n")
xgboost_1 <- h2o.xgboost(
  model_id = "XGBoost_1", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  booster = "gbtree", col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8,
  max_depth = 10, min_rows = 5, ntrees = 10000, reg_alpha = 0, reg_lambda = 1,
  sample_rate = 0.6, keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
xgboost_2 <- h2o.xgboost(
  model_id = "XGBoost_2", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  booster = "gbtree", col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8,
  max_depth = 20, min_rows = 10, ntrees = 10000, reg_alpha = 0, reg_lambda = 1,
  sample_rate = 0.6, keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
xgboost_3 <- h2o.xgboost(
  model_id = "XGBoost_3", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  booster = "gbtree", col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8,
  max_depth = 5, min_rows = 3, ntrees = 10000, reg_alpha = 0, reg_lambda = 1,
  sample_rate = 0.8, keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
for (model_id in paste0("XGBoost_", 1:3)) {
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
}
cat("DRF\n")
drf <- h2o.randomForest(
  model_id = "DRF", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, ntrees = 10000,
  score_tree_interval = 5, stopping_rounds = 3,
  balance_classes = TRUE, class_sampling_factors = samp_factors,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
tmp_path <- h2o.saveModel(h2o.getModel("DRF"), path = exp_dir, force = TRUE)
cat("XRT\n")
xrt <- h2o.randomForest(
  model_id = "XRT", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, ntrees = 10000, histogram_type = "Random",
  score_tree_interval = 5, stopping_rounds = 3,
  balance_classes = TRUE, class_sampling_factors = samp_factors,
```

```
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
tmp_path <- h2o.saveModel(h2o.getModel("XRT"), path = exp_dir, force = TRUE)
# -----------------------
# get holdout predictions
# -----------------------
base_models <- as.list(c(
  unlist(deeplearning_1@model_ids),
  unlist(gbm@model_ids), paste0("GBM_", 1:5),
  unlist(xgboost@model_ids), paste0("XGBoost_", 1:3),
  "DRF",
  "XRT"
))
for (model_id in base_models) {
  res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_",
↪  model_id))["penetrating"],
    col.names = model_id
  ))
}
# ----------------------
# super learner training
# ----------------------
sl_iter <- 0
cat(paste0("Super learner iteration ", sl_iter, " (", length(base_models), " models)\n"))
sl <- h2o.stackedEnsemble(
  x = x, y = y, model_id = paste0("superlearner_iter_", sl_iter),
  training_frame = tmp, seed = 1,
  base_models = base_models,
  metalearner_algorithm = "glm",
  metalearner_nfolds = nfolds,
  keep_levelone_frame = TRUE,
  metalearner_params = list(
    standardize = TRUE, keep_cross_validation_predictions = TRUE,
    balance_classes = TRUE, class_sampling_factors = samp_factors
  )
)
tmp_path <- h2o.saveModel(h2o.getModel(paste0("superlearner_iter_", sl_iter)), path = exp_dir, force
↪  = TRUE)
model_id <- paste0("metalearner_glm_superlearner_iter_", sl_iter)
tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_",
↪  model_id))["penetrating"],
```

```
    col.names = paste0(model_id, "_", length(base_models), "_models")
  ))
  # ----------------------------------
  # super learner base model reduction
  # ----------------------------------
  while (TRUE) {
    meta <- h2o.getModel(paste0("metalearner_glm_superlearner_iter_", sl_iter, "_cv_1"))
    names <- meta@model$coefficients_table[, "names"]
    coeffs <- (meta@model$coefficients_table[, "standardized_coefficients"] > 0)
    for (j in 2:nfolds) {
      meta <- h2o.getModel(paste0("metalearner_glm_superlearner_iter_", sl_iter, "_cv_", j))
      names <- meta@model$coefficients_table[, "names"]
      coeffs <- coeffs + (meta@model$coefficients_table[, "standardized_coefficients"] > 0)
    }
    base_models_ <- as.list(names[coeffs >= ceiling(nfolds / 2) & names != "Intercept"])
    if (length(base_models_) == 0) {
      cat("No base models passing the threshold\n\n")
      break
    }
    if (sum(base_models %in% base_models_) == length(base_models)) {
      cat("No further reduction of base models\n\n")
      break
    }
    sl_iter <- sl_iter + 1
    base_models <- base_models_
    cat(paste0("Super learner iteration ", sl_iter, " (", length(base_models), " models)\n"))
    sl <- h2o.stackedEnsemble(
      x = x, y = y, model_id = paste0("superlearner_iter_", sl_iter),
      training_frame = tmp, seed = 1,
      base_models = base_models,
      metalearner_algorithm = "glm",
      metalearner_nfolds = nfolds,
      keep_levelone_frame = TRUE,
      metalearner_params = list(
        standardize = TRUE, keep_cross_validation_predictions = TRUE,
        balance_classes = TRUE, class_sampling_factors = samp_factors
      )
    )
    tmp_path <- h2o.saveModel(h2o.getModel(paste0("superlearner_iter_", sl_iter)), path = exp_dir,
↪   force = TRUE)
    model_id <- paste0("metalearner_glm_superlearner_iter_", sl_iter)
```

```
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
  res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_",
↪ model_id))["penetrating"],
    col.names = paste0(model_id, "_", length(base_models), "_models")
  ))
}
# ----------------------------------------
# super learner for homogeneous base models
# ----------------------------------------
# DeepLearning
base_models <- deeplearning_1@model_ids
cat(paste0("Super learner deep learning (", length(base_models), " models)\n"))
sl <- h2o.stackedEnsemble(
  x = x, y = y, model_id = "superlearner_deeplearning",
  training_frame = tmp, seed = 1,
  base_models = base_models,
  metalearner_algorithm = "glm",
  metalearner_nfolds = nfolds,
  keep_levelone_frame = TRUE,
  metalearner_params = list(
    standardize = TRUE, keep_cross_validation_predictions = TRUE,
    balance_classes = TRUE, class_sampling_factors = samp_factors
  )
)
tmp_path <- h2o.saveModel(h2o.getModel("superlearner_deeplearning"), path = exp_dir, force = TRUE)
model_id <- "metalearner_glm_superlearner_deeplearning"
tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_",
↪ model_id))["penetrating"],
  col.names = paste0(model_id, "_", length(base_models), "_models")
))
# GBM
base_models <- as.list(c(unlist(gbm@model_ids), paste0("GBM_", 1:5)))
cat(paste0("Super learner GBM (", length(base_models), " models)\n"))
sl <- h2o.stackedEnsemble(
  x = x, y = y, model_id = "superlearner_gbm",
  training_frame = tmp, seed = 1,
  base_models = base_models,
  metalearner_algorithm = "glm",
  metalearner_nfolds = nfolds,
  keep_levelone_frame = TRUE,
```

```
    metalearner_params = list(
      standardize = TRUE, keep_cross_validation_predictions = TRUE,
      balance_classes = TRUE, class_sampling_factors = samp_factors
    )
  )
  tmp_path <- h2o.saveModel(h2o.getModel("superlearner_gbm"), path = exp_dir, force = TRUE)
  model_id <- "metalearner_glm_superlearner_gbm"
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
  res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_",
↪  model_id))["penetrating"],
    col.names = paste0(model_id, "_", length(base_models), "_models")
  ))
  # XGBoost
  base_models <- as.list(c(unlist(xgboost@model_ids), paste0("XGBoost_", 1:3)))
  cat(paste0("Super learner XGBoost (", length(base_models), " models)\n"))
  sl <- h2o.stackedEnsemble(
    x = x, y = y, model_id = "superlearner_xgboost",
    training_frame = tmp, seed = 1,
    base_models = base_models,
    metalearner_algorithm = "glm",
    metalearner_nfolds = nfolds,
    keep_levelone_frame = TRUE,
    metalearner_params = list(
      standardize = TRUE, keep_cross_validation_predictions = TRUE,
      balance_classes = TRUE, class_sampling_factors = samp_factors
    )
  )
  tmp_path <- h2o.saveModel(h2o.getModel("superlearner_xgboost"), path = exp_dir, force = TRUE)
  model_id <- "metalearner_glm_superlearner_xgboost"
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
  res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_",
↪  model_id))["penetrating"],
    col.names = paste0(model_id, "_", length(base_models), "_models")
  ))
  write.csv(res, file = paste0(exp_dir, "/cv_holdout_predictions.csv"), row.names = FALSE)
  cat("\n\n")
  h2o.removeAll()
}
```

3.4.2 Inner cross-validation

```
load(file = "./rdata/var_reduct_cpp_train_test_splits.RData")
for (i in 1:10) {
  cat(paste0("Outer training set ", i, "\n"))
  prefix <- paste0("outer_", i)
  exp_dir <- paste0("/Users/renee/Downloads/cell_penetration/", prefix)
  dir.create(exp_dir)
  train_cpp_models(train_set = outer_splits[[i]][["train"]], exp_dir = exp_dir, prefix = prefix)
}
```

3.4.3 Training on whole data set

```
load(file = "./rdata/var_reduct_cpp_train_test_splits.RData")
prefix <- "whole_data_set"
exp_dir <- paste0("/Users/renee/Downloads/cell_penetration/", prefix)
dir.create(exp_dir)
train_cpp_models(train_set = cpp_data, exp_dir = exp_dir, prefix = prefix)
```

```
# Keep track of grid models
dl_grid <- list()
gbm_grid <- list()
xgboost_grid <- list()
dl_grid_params <- list()
gbm_grid_params <- list()
xgboost_grid_params <- list()
for (i in 1:11) {
  if (i == 11) {
    cat(paste0("Whole data set\n"))
    prefix <- "whole_data_set"
  } else {
    cat(paste0("Outer training set ", i, "\n"))
    prefix <- paste0("outer_", i)
  }
  dir <- paste0("/Users/renee/Downloads/cell_penetration/", prefix)
  files <- list.files(dir)
  # Deep learning
  dl <- files[str_detect(files, "DeepLearning_model")]
```

```
for (m in dl) {
  model <- h2o.loadModel(paste0(dir, "/", m))
  hs <- sha1(paste(c(
    model@allparameters$epsilon,
    model@allparameters$hidden,
    model@allparameters$hidden_dropout_ratios,
    model@allparameters$input_dropout_ratio,
    model@allparameters$rho
  ), collapse = " "))
  if (hs %in% names(dl_grid)) {
    dl_grid[[hs]] <- c(dl_grid[[hs]], m)
  } else {
    dl_grid[[hs]] <- c(m)
    dl_grid_params <- list.append(
      dl_grid_params,
      c(
        "epsilon" = model@allparameters$epsilon,
        "hidden" = paste0("[", paste(model@allparameters$hidden, collapse = ","), "]"),
        "hidden_dropout_ratios" = paste0(
          "[",
          paste(model@allparameters$hidden_dropout_ratios,
            collapse = ","
          ), "]"
        ),
        "input_dropout_ratio" = model@allparameters$input_dropout_ratio,
        "rho" = model@allparameters$rho
      )
    )
  }
}
h2o.removeAll()
# GBM
gbm <- files[str_detect(files, "GBM_model")]
for (m in gbm) {
  model <- h2o.loadModel(paste0(dir, "/", m))
  hs <- sha1(paste(c(
    model@allparameters$col_sample_rate,
    model@allparameters$col_sample_rate_per_tree,
    model@allparameters$max_depth,
    model@allparameters$min_rows,
    model@allparameters$min_split_improvement,
```

```
    model@allparameters$sample_rate
  ), collapse = " "))
  if (hs %in% names(gbm_grid)) {
    gbm_grid[[hs]] <- c(gbm_grid[[hs]], m)
  } else {
    gbm_grid[[hs]] <- c(m)
    gbm_grid_params <- list.append(
      gbm_grid_params,
      c(
        "col_sample_rate" = model@allparameters$col_sample_rate,
        "col_sample_rate_per_tree" = model@allparameters$col_sample_rate_per_tree,
        "max_depth" = model@allparameters$max_depth,
        "min_rows" = model@allparameters$min_rows,
        "min_split_improvement" = model@allparameters$min_split_improvement,
        "sample_rate" = model@allparameters$sample_rate
      )
    )
  }
}
h2o.removeAll()
# XGBoost
xgboost <- files[str_detect(files, "XGBoost_model")]
for (m in xgboost) {
  model <- h2o.loadModel(paste0(dir, "/", m))
  hs <- sha1(paste(c(
    model@allparameters$booster,
    model@allparameters$col_sample_rate,
    model@allparameters$col_sample_rate_per_tree,
    model@allparameters$max_depth,
    model@allparameters$min_rows,
    model@allparameters$reg_alpha,
    model@allparameters$reg_lambda,
    model@allparameters$sample_rate
  ), collapse = " "))
  if (hs %in% names(xgboost_grid)) {
    xgboost_grid[[hs]] <- c(xgboost_grid[[hs]], m)
  } else {
    xgboost_grid[[hs]] <- c(m)
    xgboost_grid_params <- list.append(
      xgboost_grid_params,
      c(
```

```
          "booster" = model@allparameters$booster,
          "col_sample_rate" = model@allparameters$col_sample_rate,
          "col_sample_rate_per_tree" = model@allparameters$col_sample_rate_per_tree,
          "max_depth" = model@allparameters$max_depth,
          "min_rows" = model@allparameters$min_rows,
          "reg_alpha" = model@allparameters$reg_alpha,
          "reg_lambda" = model@allparameters$reg_lambda,
          "sample_rate" = model@allparameters$sample_rate
        )
      )
    }
  }
  h2o.removeAll()
}

dl_grid_params <- as.data.frame(t(data.frame(dl_grid_params)))
rownames(dl_grid_params) <- paste("Neural network grid model", 1:nrow(dl_grid_params))

gbm_grid_params <- as.data.frame(t(data.frame(gbm_grid_params)))
rownames(gbm_grid_params) <- paste("GBM grid model", 1:nrow(gbm_grid_params))

xgboost_grid_params <- as.data.frame(t(data.frame(xgboost_grid_params)))
rownames(xgboost_grid_params) <- paste("XGBoost grid model", 1:nrow(xgboost_grid_params))

write.csv(dl_grid_params, "./other_data/cell_penetration/neural_network_grid_params.csv")
write.csv(gbm_grid_params, "./other_data/cell_penetration/gbm_grid_params.csv")
write.csv(xgboost_grid_params, "./other_data/cell_penetration/xgboost_grid_params.csv")

save(dl_grid, gbm_grid, xgboost_grid, file = "./rdata/model_training_grid_models_cpp.RData")
```

3.4.4 Model evaluation

```
model_evaluation <- function(holdout_pred, fold_asign, grid_name, grid_meta = NULL) {
  res_ <- list()
  model_num <- c()
  if (startsWith(grid_name, "Super learner")) {
    if (grid_name == "Super learner all models") {
      m <- colnames(holdout_pred)[startsWith(colnames(holdout_pred),
↪  "metalearner_glm_superlearner_iter_0")]
```

```
  } else if (grid_name == "Super learner final") {
    sl_models <- colnames(holdout_pred)[startsWith(colnames(holdout_pred),
↪ "metalearner_glm_superlearner_iter")]
    m <- sl_models[length(sl_models)]
  } else if (grid_name == "Super learner neural network") {
    m <- colnames(holdout_pred)[startsWith(colnames(holdout_pred),
↪ "metalearner_glm_superlearner_deeplearning")]
  } else if (grid_name == "Super learner GBM") {
    m <- colnames(holdout_pred)[startsWith(colnames(holdout_pred),
↪ "metalearner_glm_superlearner_gbm")]
  } else if (grid_name == "Super learner XGBoost") {
    m <- colnames(holdout_pred)[startsWith(colnames(holdout_pred),
↪ "metalearner_glm_superlearner_xgboost")]
  }
  logloss <- c()
  MCC <- c()
  F1 <- c()
  acc <- c()
  EF <- c()
  BEDROC <- c()
  threshold <- 0.5
  for (k in 1:10) {
    y_true <- sapply(holdout_pred[fold_assign == k, "category"], function(x) if (x == "penetrating")
↪ 1 else 0)
    y_pred <- holdout_pred[fold_assign == k, m]
    y_pred_cls <- sapply(y_pred, function(x) if (x >= threshold) 1 else 0)
    logloss <- c(logloss, LogLoss(y_pred = y_pred, y_true = y_true))
    MCC <- c(MCC, mcc(preds = y_pred_cls, actuals = y_true))
    F1 <- c(F1, F1_Score(y_pred = y_pred_cls, y_true = y_true))
    acc <- c(acc, balanced_accuracy(y_pred = y_pred_cls, y_true = y_true))
    EF <- c(EF, enrichment_factor(x = y_pred, y = y_true, top = 0.05))
    BEDROC <- c(BEDROC, bedroc(x = y_pred, y = y_true, alpha = 20))
  }
  res_ <- list.append(res_, c(
    mean(logloss, na.rm = TRUE), sem(logloss),
    mean(MCC, na.rm = TRUE), sem(MCC),
    mean(F1, na.rm = TRUE), sem(F1),
    mean(acc, na.rm = TRUE), sem(acc),
    mean(EF, na.rm = TRUE), sem(EF),
    mean(BEDROC, na.rm = TRUE), sem(BEDROC),
    logloss, MCC, F1, acc, EF, BEDROC
```

```
    ))
  } else {
    for (j in 1:length(grid_meta)) {
      g <- grid_meta[[j]]
      m <- intersect(colnames(holdout_pred), g)
      if (length(m) == 1) {
        model_num <- c(model_num, j)
        logloss <- c()
        MCC <- c()
        F1 <- c()
        acc <- c()
        EF <- c()
        BEDROC <- c()
        threshold <- 0.5
        for (k in 1:10) {
          y_true <- sapply(holdout_pred[fold_assign == k, "category"], function(x) if (x ==
↪  "penetrating") 1 else 0)
          y_pred <- holdout_pred[fold_assign == k, m]
          y_pred_cls <- sapply(y_pred, function(x) if (x >= threshold) 1 else 0)
          logloss <- c(logloss, LogLoss(y_pred = y_pred, y_true = y_true))
          MCC <- c(MCC, mcc(preds = y_pred_cls, actuals = y_true))
          F1 <- c(F1, F1_Score(y_pred = y_pred_cls, y_true = y_true))
          acc <- c(acc, balanced_accuracy(y_pred = y_pred_cls, y_true = y_true))
          EF <- c(EF, enrichment_factor(x = y_pred, y = y_true, top = 0.05))
          BEDROC <- c(BEDROC, bedroc(x = y_pred, y = y_true, alpha = 20))
        }
        res_ <- list.append(res_, c(
          mean(logloss, na.rm = TRUE), sem(logloss),
          mean(MCC, na.rm = TRUE), sem(MCC),
          mean(F1, na.rm = TRUE), sem(F1),
          mean(acc, na.rm = TRUE), sem(acc),
          mean(EF, na.rm = TRUE), sem(EF),
          mean(BEDROC, na.rm = TRUE), sem(BEDROC),
          logloss, MCC, F1, acc, EF, BEDROC
        ))
      }
    }
  }
  res <- as.data.frame(t(data.frame(res_)))
  colnames(res) <- c(
    "Log loss mean", "Log loss s.e.m.", "MCC mean", "MCC s.e.m.", "F_1 mean", "F_1 s.e.m.",
```

```
    "Balanced accuracy mean", "Balanced accuracy s.e.m.", "EF mean", "EF s.e.m.", "BEDROC mean",
    ↪  "BEDROC s.e.m.",
    paste("Log loss CV", 1:10), paste("MCC CV", 1:10), paste("F_1 CV", 1:10),
    paste("Balanced accuracy CV", 1:10), paste("EF CV", 1:10), paste("BEDROC CV", 1:10)
  )
  if (nrow(res) == 1) {
    rownames(res) <- grid_name
  } else {
    rownames(res) <- paste(grid_name, model_num)
  }
  return(res)
}
```

3.4.5 Inner loop model selection

```
load(file = "./rdata/model_training_grid_models_cpp.RData")

for (i in 1:10) {
  holdout_pred <- read.csv(paste0("./other_data/cell_penetration/outer_", i,
↪  "/cv_holdout_predictions.csv"))
  fold_assign <- rep(1:10, ceiling(nrow(holdout_pred) / 10))[1:nrow(holdout_pred)]
  data_ <- list()
  # Deep learning grid models
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
    grid_name = "Neural network grid model",
    grid_meta = dl_grid
  ))
  # GBM grid models
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
    grid_name = "GBM grid model",
    grid_meta = gbm_grid
  ))
  # GBM default models
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
    grid_name = "GBM model",
    grid_meta = as.list(paste0("GBM_", 1:5))
  ))
  # XGBoost grid models
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
```

```
    grid_name = "XGBoost grid model",
    grid_meta = xgboost_grid
  ))
  # XGBoost default models
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
    grid_name = "XGBoost model",
    grid_meta = as.list(paste0("XGBoost_", 1:3))
  ))
  # DRF
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "DRF model",
↪  grid_meta = list("DRF")))
  # XRT
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "XRT model",
↪  grid_meta = list("XRT")))
  # Super learner all
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪  all models"))
  # Super learner final
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪  final"))
  # Super learner deep learning
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪  neural network"))
  # Super learner GBM
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪  GBM"))
  # Super learner XGBoost
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪  XGBoost"))

  data <- do.call(rbind, data_)
  write.csv(data, paste0("./other_data/cell_penetration/outer_", i, "/cv_res.csv"))
}
```

3.4.5.1 CV 1

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
GBM grid model 30	0.9156	0.7578	0.992	0.7479	14	9	9	10	42

Showing 1 to 1 of 272 entries Previous 1 2 3 4 5 … 272 Next

3.4.5.2 CV 2

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
Neural network grid model 98		0.6819	0.777	0.5781	1	8	9	8	26

Showing 1 to 1 of 227 entries Previous 1 2 3 4 5 … 227 Next

3.4.5.3 CV 3

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
Neural network grid model 72			0.9919		1	1	3	1	6

Showing 1 to 1 of 277 entries Previous 1 2 3 4 5 … 277 Next

3.4.5.4 CV 4

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
GBM grid model 1		0.8723		0.8723	1	2	1	2	6

Showing 1 to 1 of 303 entries Previous 1 2 3 4 5 … 303 Next

3.4.5.5 CV 5

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
GBM grid model 45		1		0.9887	1	2	1	2	6

Showing 1 to 1 of 300 entries Previous 1 2 3 4 5 … 300 Next

3.4.5.6 CV 6

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
GBM grid model 82	0.9156	0.9239	0.7783	0.8475	2	3	3	2	10

Showing 1 to 1 of 303 entries Previous 1 2 3 4 5 … 303 Next

3.4.5.7 CV 7

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
GBM grid model 72	0.9698	0.9824	1	0.9761	4	3	2	2	11

Showing 1 to 1 of 304 entries Previous 1 2 3 4 5 … 304 Next

3.4.5.8 CV 8

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
GBM grid model 15	0.6914	1	0.9643	0.8284	3	2	2	2	9

Showing 1 to 1 of 303 entries Previous 1 2 3 4 5 … 303 Next

3.4.5.9 CV 9

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
GBM grid model 74	0.9729	0.7721	0.5349	0.8257	4	2	2	2	10

Showing 1 to 1 of 122 entries Previous 1 2 3 4 5 … 122 Next

3.4.5.10 CV 10

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
GBM grid model 24	1	0.8119		0.799	1	2	2	3	8

Showing 1 to 1 of 303 entries Previous 1 2 3 4 5 … 303 Next

3.4.6 Final evaluation

```
load(file = "./rdata/var_reduct_cpp_train_test_splits.RData")
load(file = "./rdata/model_training_grid_models_cpp.RData")
dir <- "/Users/renee/Downloads/cell_penetration"
selected_models <- c(
  "GBM grid model 30", "Neural network grid model 98", "Neural network grid model 72",
  "GBM grid model 1", "GBM grid model 45", "GBM grid model 82", "GBM grid model 72",
  "GBM grid model 15", "GBM grid model 74", "GBM grid model 24"
)
logloss <- c()
MCC <- c()
F1 <- c()
acc <- c()
EF <- c()
BEDROC <- c()
threshold <- 0.5
for (i in 1:10) {
  holdout_pred <- read.csv(paste0("./other_data/cell_penetration/outer_", i,
↪  "/cv_holdout_predictions.csv"))
  if (startsWith(selected_models[i], "Neural network grid model")) {
```

```
    grid_num <- as.integer(str_extract(selected_models[i], "[0-9]+"))
    m <- intersect(colnames(holdout_pred), dl_grid[[grid_num]])
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/", m))
  } else if (startsWith(selected_models[i], "GBM grid model")) {
    grid_num <- as.integer(str_extract(selected_models[i], "[0-9]+"))
    m <- intersect(colnames(holdout_pred), gbm_grid[[grid_num]])
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/", m))
  } else if (startsWith(selected_models[i], "GBM model")) {
    grid_num <- as.integer(str_extract(selected_models[i], "[0-9]+"))
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/GBM_", grid_num))
  } else if (startsWith(selected_models[i], "XGBoost grid model")) {
    grid_num <- as.integer(str_extract(selected_models[i], "[0-9]+"))
    m <- intersect(colnames(holdout_pred), xgboost_grid[[grid_num]])
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/", m))
  } else if (startsWith(selected_models[i], "XGBoost model")) {
    grid_num <- as.integer(str_extract(selected_models[i], "[0-9]+"))
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/XGBoost_", grid_num))
  } else if (selected_models[i] == "Super learner all models") {
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/superlearner_iter_0"))
  } else if (selected_models[i] == "Super learner final") {
    files <- list.files(paste0(dir, "/outer_", i))
    files <- files[startsWith(files, "superlearner_iter")]
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/", files[length(files)]))
  } else if (selected_models[i] == "Super learner neural network") {
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/superlearner_deeplearning"))
  } else if (selected_models[i] == "Super learner GBM") {
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/superlearner_gbm"))
  } else if (selected_models[i] == "Super learner XGBoost") {
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/superlearner_xgboost"))
  } else {
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/", unlist(strsplit(selected_models[i], "
↪  "))[1]))
  }
  tmp <- as.h2o(subset(outer_splits[[i]][["test"]], select = -category))
  y_true <- sapply(outer_splits[[i]][["test"]]$category, function(x) if (x == "penetrating") 1 else 0)
  y_pred <- as.data.frame(as.data.frame(h2o.predict(model, tmp)))[, "penetrating"]
  y_pred_cls <- sapply(y_pred, function(x) if (x >= threshold) 1 else 0)
  logloss <- c(logloss, LogLoss(y_pred = y_pred, y_true = y_true))
  MCC <- c(MCC, mcc(preds = y_pred_cls, actuals = y_true))
  F1 <- c(F1, F1_Score(y_pred = y_pred_cls, y_true = y_true))
  acc <- c(acc, balanced_accuracy(y_pred = y_pred_cls, y_true = y_true))
```

```
  EF <- c(EF, enrichment_factor(x = y_pred, y = y_true, top = 0.05))
  BEDROC <- c(BEDROC, bedroc(x = y_pred, y = y_true, alpha = 20))
}
save(logloss, MCC, F1, acc, EF, BEDROC, file = "./rdata/final_evaluation_cpp.RData")
```

```
load(file = "./rdata/final_evaluation_cpp.RData")
data.frame(
  Metric = c("Log loss", "MCC", "F<sub>1</sub>", "Balanced accuracy", "EF", "BEDROC"),
  `Mean ± s.e.m.` = c(
    paste0(round(mean(logloss), 3), " ± ", round(sem(logloss), 3)),
    paste0(round(mean(MCC), 3), " ± ", round(sem(MCC), 3)),
    paste0(round(mean(F1), 3), " ± ", round(sem(F1), 3)),
    paste0(round(mean(acc), 3), " ± ", round(sem(acc), 3)),
    paste0(round(mean(EF), 3), " ± ", round(sem(EF), 3)),
    paste0(round(mean(BEDROC), 3), " ± ", round(sem(BEDROC), 3))
  ),
  check.names = FALSE
) %>%
  datatable(escape = FALSE, rownames = FALSE)
```

Show 10 entries Search:

Metric	Mean ± s.e.m.
Log loss	0.259 ± 0.017
MCC	0.794 ± 0.014
F_1	0.901 ± 0.005
Balanced accuracy	0.897 ± 0.007
EF	2.081 ± 0.065
BEDROC	0.999 ± 0

Showing 1 to 6 of 6 entries Previous 1 Next

3.4.7 Final model selection

```
load(file = "./rdata/model_training_grid_models_cpp.RData")

holdout_pred <- read.csv("./other_data/cell_penetration/whole_data_set/cv_holdout_predictions.csv")
fold_assign <- rep(1:10, ceiling(nrow(holdout_pred) / 10))[1:nrow(holdout_pred)]
```

```
data_ <- list()
# Deep learning grid models
data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
  grid_name = "Neural network grid model",
  grid_meta = dl_grid
))
# GBM grid models
data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
  grid_name = "GBM grid model",
  grid_meta = gbm_grid
))
# GBM default models
data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
  grid_name = "GBM model",
  grid_meta = as.list(paste0("GBM_", 1:5))
))
# XGBoost grid models
data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
  grid_name = "XGBoost grid model",
  grid_meta = xgboost_grid
))
# XGBoost default models
data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
  grid_name = "XGBoost model",
  grid_meta = as.list(paste0("XGBoost_", 1:3))
))
# DRF
data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "DRF model",
↪  grid_meta = list("DRF")))
# XRT
data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "XRT model",
↪  grid_meta = list("XRT")))
# Super learner all
data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner all
↪  models"))
# Super learner final
data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪  final"))
# Super learner deep learning
data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪  neural network"))
```

```
# Super learner GBM
data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪  GBM"))
# Super learner XGBoost
data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪  XGBoost"))

data <- do.call(rbind, data_)
write.csv(data, "./other_data/cell_penetration/whole_data_set/cv_res.csv")
```

```
data <- read.csv("./other_data/cell_penetration/whole_data_set/cv_res.csv", row.names = 1, check.names
↪  = FALSE)
statistical_testing(data,
  metric_vec = c("Log loss", "MCC", "F_1", "Balanced accuracy"),
  decreasing_vec = c(FALSE, TRUE, TRUE, TRUE)
)
```

Show 10 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
GBM grid model 59		0.7499	1	0.7578	1	2	5	3	11
GBM grid model 15	0.9698	0.5261	0.7942	0.5495	2	8	16	10	36
GBM grid model 53	0.6678	0.8825	0.873	0.8854	36	6	4	7	53
XGBoost grid model 14	0.7092	0.6012	0.8867	0.7578	22	14	10	14	60
GBM grid model 46	0.5708	0.8015	0.9919	0.8041	64	3	9	2	78
GBM grid model 66	0.8041	0.6435	0.8479	0.6157	10	20	27	23	80
GBM grid model 60	0.5433	0.7745	0.9365	0.777	65	13	14	13	105
GBM grid model 67	0.4429	0.6012	0.8867	0.5495	61	15	15	17	108
GBM grid model 35	0.6678	0.4054	0.7942	0.5179	30	26	25	28	109
XGBoost grid model 88	0.7092	0.5261	0.6482	0.6157	9	32	34	34	109

Showing 1 to 10 of 300 entries Previous 1 2 3 4 5 … 30 Next

```
# statistical_testing(data, metric_vec=c('Log loss', 'MCC', 'F_1', 'Balanced accuracy'),
#                     decreasing_vec=c(FALSE, TRUE, TRUE, TRUE),
#
↪  output_path='./other_data/cell_penetration/whole_data_set/cv_res_statistical_testing.csv')
```

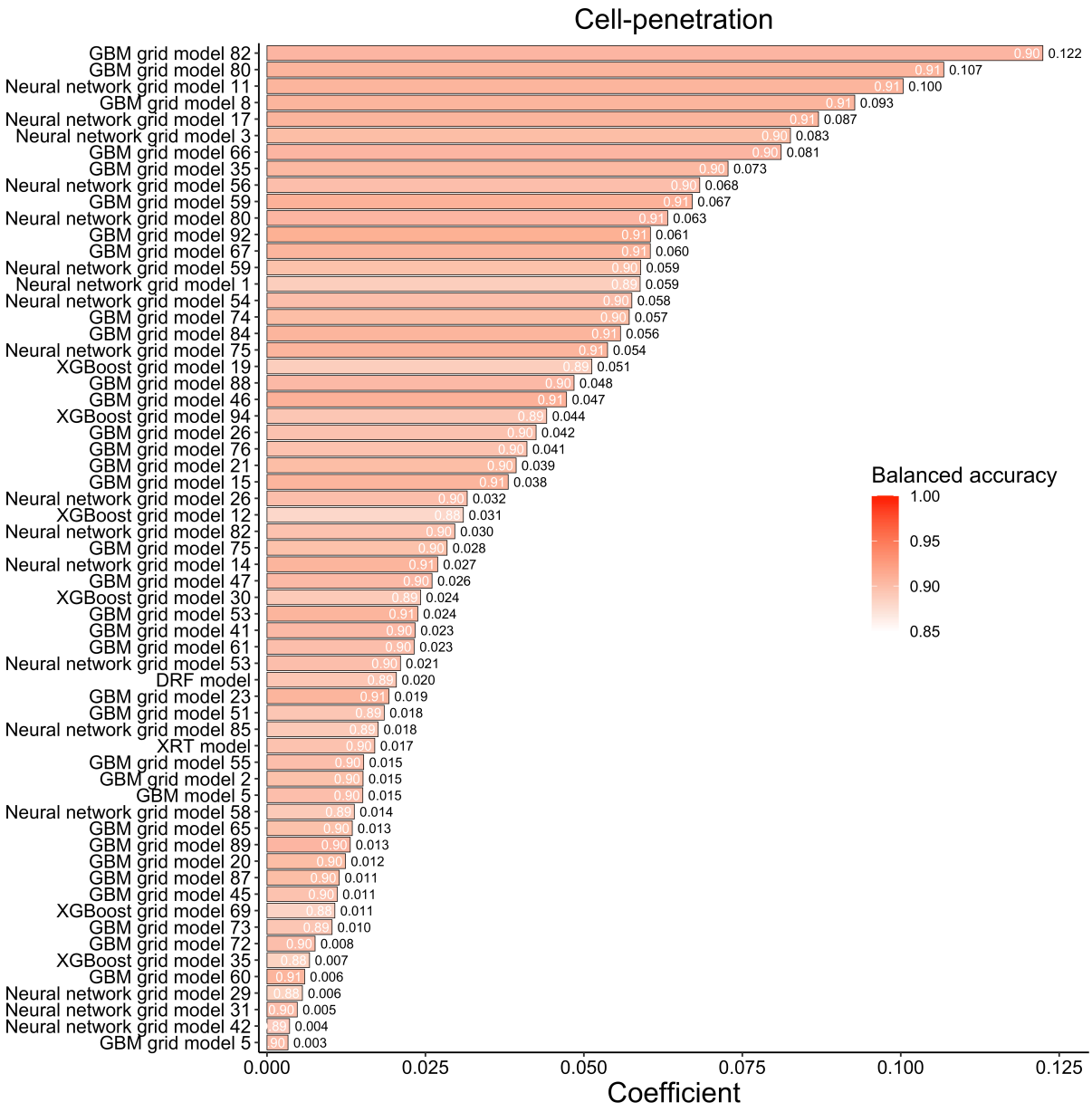
3.4.8 Final reduced SL

```
h2o.init(nthreads = -1)
model_path <- "/Users/renee/Downloads/cell_penetration/whole_data_set/superlearner_iter_1"
model <- h2o.loadModel(model_path)
load(file = "./rdata/model_training_grid_models_cpp.RData")
data <- read.csv("./other_data/cell_penetration/whole_data_set/cv_res.csv", row.names = 1, check.names
↪  = FALSE)
df <- as.data.frame(model@model$metalearner_model@model$coefficients_table)[c("names",
↪  "standardized_coefficients")][-1, ]
```

```
for (i in 1:nrow(df)) {
  name <- df$names[i]
  if (startsWith(name, "DeepLearning_model")) {
    for (j in 1:length(dl_grid)) {
      if (name %in% dl_grid[[j]]) break
    }
    df$names[i] <- paste("Neural network grid model", j)
  } else if (startsWith(name, "GBM_model")) {
    for (j in 1:length(gbm_grid)) {
      if (name %in% gbm_grid[[j]]) break
    }
    df$names[i] <- paste("GBM grid model", j)
  } else if (startsWith(name, "XGBoost_model")) {
    for (j in 1:length(xgboost_grid)) {
      if (name %in% xgboost_grid[[j]]) break
    }
    df$names[i] <- paste("XGBoost grid model", j)
  } else if (startsWith(name, "DRF")) {
    df$names[i] <- "DRF model"
  } else if (startsWith(name, "XRT")) {
    df$names[i] <- "XRT model"
  } else {
    df$names[i] <- paste(strsplit(name, "_")[[1]], collapse = " model ")
  }
}

rownames(df) <- df$names
df <- merge(df, data["Balanced accuracy mean"], by = "row.names")[, -1]
df <- df[df$standardized_coefficients > 0, ]
```

```
p2 <- ggplot(df, aes(x = reorder(names, standardized_coefficients), y = standardized_coefficients,
↪  fill = `Balanced accuracy mean`)) +
  geom_col(color = "black", size = 0.2) +
  scale_fill_gradient(low = "white", high = "red", n.breaks = 4, limits = c(0.85, 1.00)) +
  geom_text(aes(label = sprintf("%.2f", `Balanced accuracy mean`)), nudge_y = -0.0025, size = 3, color
  ↪  = "white") +
  geom_text(aes(label = sprintf("%.3f", standardized_coefficients)), nudge_y = 0.0035, size = 3) +
  coord_flip() +
```

```
scale_y_continuous(limits = c(0, max(df$standardized_coefficients) + 0.006), expand = c(0.01, 0)) +
theme_bw() +
theme(
  panel.grid.major = element_blank(),
  panel.grid.minor = element_blank(),
  strip.background = element_blank(),
  panel.border = element_blank(),
  axis.line = element_line(color = "black"),
  legend.text = element_text(colour = "black", size = 10),
  plot.title = element_text(hjust = 0.5, size = 18),
  plot.margin = ggplot2::margin(10, 10, 10, 0, "pt"),
  axis.title.x = element_text(colour = "black", size = 18),
  axis.title.y = element_text(colour = "black", size = 18),
  axis.text.x = element_text(colour = "black", size = 12),
  axis.text.y = element_text(colour = "black", size = 12),
  legend.title = element_text(size = 14),
  legend.position = c(0.85, 0.5)
) +
guides(fill = guide_colorbar(title = "Balanced accuracy")) +
xlab("") +
ylab("Coefficient") +
ggtitle("Cell-penetration")
```


Cell-penetration
GBM grid model 82 0.90 0.122
GBM grid model 80 0.91 0.107
Neural network grid model 11 0.91 0.100
GBM grid model 8 0.91 0.093
Neural network grid model 17 0.91 0.087
Neural network grid model 3 0.90 0.083
GBM grid model 66 0.90 0.081
GBM grid model 35 0.90 0.073
Neural network grid model 56 0.90 0.068
GBM grid model 59 0.91 0.067
Neural network grid model 80 0.91 0.063
GBM grid model 92 0.91 0.061
GBM grid model 67 0.91 0.060
Neural network grid model 59 0.90 0.059
Neural network grid model 1 0.89 0.059
Neural network grid model 54 0.90 0.058
GBM grid model 74 0.90 0.057
GBM grid model 84 0.91 0.056
Neural network grid model 75 0.91 0.054
XGBoost grid model 19 0.89 0.051
GBM grid model 88 0.90 0.048
GBM grid model 46 0.91 0.047
XGBoost grid model 94 0.89 0.044
GBM grid model 26 0.90 0.042
GBM grid model 76 0.90 0.041
GBM grid model 21 0.90 0.039
GBM grid model 15 0.91 0.038
Neural network grid model 26 0.90 0.032
XGBoost grid model 12 0.88 0.031
Neural network grid model 82 0.90 0.030
GBM grid model 75 0.90 0.028
Neural network grid model 14 0.91 0.027
GBM grid model 47 0.90 0.026
XGBoost grid model 30 0.89 0.024
GBM grid model 53 0.91 0.024
GBM grid model 41 0.90 0.023
GBM grid model 61 0.90 0.023
Neural network grid model 53 0.90 0.021
DRF model 0.88 0.020
GBM grid model 23 0.91 0.019
GBM grid model 51 0.89 0.018
Neural network grid model 85 0.89 0.018
XRT model 0.90 0.017
GBM grid model 55 0.90 0.015
GBM grid model 2 0.90 0.015
GBM model 5 0.90 0.015
Neural network grid model 58 0.89 0.014
GBM grid model 65 0.90 0.013
GBM grid model 89 0.90 0.013
GBM grid model 20 0.90 0.012
GBM grid model 87 0.90 0.011
GBM grid model 45 0.90 0.011
XGBoost grid model 69 0.89 0.011
GBM grid model 73 0.89 0.010
GBM grid model 72 0.90 0.008
XGBoost grid model 35 0.88 0.007
GBM grid model 60 0.91 0.006
Neural network grid model 29 0.88 0.006
Neural network grid model 31 0.90 0.005
Neural network grid model 42 0.88 0.004
GBM grid model 5 0.90 0.003
Balanced accuracy
1.00
0.95
0.90
0.85
0.000
0.025
0.050
0.075
0.100
0.125
Coefficient

3.5 Toxicity (classification)

3.5.1 Model training

```
train_tx_models <- function(train_set, exp_dir, prefix, nfolds = 10, grid_seed = 1) {
  tmp <- as.h2o(train_set, destination_frame = prefix)
  tmp["category"] <- as.factor(tmp["category"])
  y <- "category"
  x <- setdiff(names(tmp), y)
  res <- as.data.frame(tmp$category)
  samp_factors <- as.vector(mean(table(train_set$category)) / table(train_set$category))
  # -------------------
  # base model training
  # -------------------
  cat("Deep learning grid 1\n")
  deeplearning_1 <- h2o.grid(
    algorithm = "deeplearning", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
    keep_cross_validation_predictions = TRUE, hyper_params = deeplearning_params_1,
    stopping_rounds = 3, balance_classes = TRUE, class_sampling_factors = samp_factors,
    max_after_balance_size = 0.5,
    search_criteria = list(
      strategy = "RandomDiscrete",
      max_models = 100, seed = grid_seed
    ),
    keep_cross_validation_models = FALSE, fold_assignment = "Modulo", parallelism = 0
  )
  for (model_id in deeplearning_1@model_ids) {
    tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
  }
  cat("GBM grid\n")
  gbm <- h2o.grid(
    algorithm = "gbm", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
    keep_cross_validation_predictions = TRUE, hyper_params = gbm_params, stopping_rounds = 3,
    balance_classes = TRUE, class_sampling_factors = samp_factors, max_after_balance_size = 0.5,
    search_criteria = list(strategy = "RandomDiscrete", max_models = 100, seed = grid_seed),
    keep_cross_validation_models = FALSE, fold_assignment = "Modulo", parallelism = 0
  )
  for (model_id in gbm@model_ids) {
    tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
  }
  cat("GBM 5 default models\n")
```

```
gbm_1 <- h2o.gbm(
  model_id = "GBM_1", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  balance_classes = TRUE, class_sampling_factors = samp_factors, max_after_balance_size = 0.5,
  col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8, learn_rate = 0.1,
  max_depth = 6, min_rows = 1, min_split_improvement = 1e-5, ntrees = 10000, sample_rate = 0.8,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
gbm_2 <- h2o.gbm(
  model_id = "GBM_2", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  balance_classes = TRUE, class_sampling_factors = samp_factors, max_after_balance_size = 0.5,
  col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8, learn_rate = 0.1,
  max_depth = 7, min_rows = 10, min_split_improvement = 1e-5, ntrees = 10000, sample_rate = 0.8,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
gbm_3 <- h2o.gbm(
  model_id = "GBM_3", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  balance_classes = TRUE, class_sampling_factors = samp_factors, max_after_balance_size = 0.5,
  col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8, learn_rate = 0.1,
  max_depth = 8, min_rows = 10, min_split_improvement = 1e-5, ntrees = 10000, sample_rate = 0.8,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
gbm_4 <- h2o.gbm(
  model_id = "GBM_4", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  balance_classes = TRUE, class_sampling_factors = samp_factors, max_after_balance_size = 0.5,
  col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8, learn_rate = 0.1,
  max_depth = 10, min_rows = 10, min_split_improvement = 1e-5, ntrees = 10000, sample_rate = 0.8,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
gbm_5 <- h2o.gbm(
  model_id = "GBM_5", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  balance_classes = TRUE, class_sampling_factors = samp_factors, max_after_balance_size = 0.5,
  col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8, learn_rate = 0.1,
  max_depth = 15, min_rows = 100, min_split_improvement = 1e-5, ntrees = 10000, sample_rate = 0.8,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
for (model_id in paste0("GBM_", 1:5)) {
```

```
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
}
cat("XGBoost grid\n")
xgboost <- h2o.grid(
  algorithm = "xgboost", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, hyper_params = xgboost_params, stopping_rounds = 3,
  search_criteria = list(strategy = "RandomDiscrete", max_models = 100, seed = grid_seed),
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo", parallelism = 0
)
for (model_id in xgboost@model_ids) {
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
}
cat("XGBoost 3 default models\n")
xgboost_1 <- h2o.xgboost(
  model_id = "XGBoost_1", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  booster = "gbtree", col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8,
  max_depth = 10, min_rows = 5, ntrees = 10000, reg_alpha = 0, reg_lambda = 1,
  sample_rate = 0.6, keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
xgboost_2 <- h2o.xgboost(
  model_id = "XGBoost_2", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  booster = "gbtree", col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8,
  max_depth = 20, min_rows = 10, ntrees = 10000, reg_alpha = 0, reg_lambda = 1,
  sample_rate = 0.6, keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
xgboost_3 <- h2o.xgboost(
  model_id = "XGBoost_3", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  booster = "gbtree", col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8,
  max_depth = 5, min_rows = 3, ntrees = 10000, reg_alpha = 0, reg_lambda = 1,
  sample_rate = 0.8, keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
for (model_id in paste0("XGBoost_", 1:3)) {
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
}
cat("GLM\n")
glm <- h2o.glm(
  model_id = "GLM", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, alpha = c(0.0, 0.2, 0.4, 0.6, 0.8, 1.0),
```

```
  balance_classes = TRUE, class_sampling_factors = samp_factors, max_after_balance_size = 0.5,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
tmp_path <- h2o.saveModel(h2o.getModel("GLM"), path = exp_dir, force = TRUE)
cat("DRF\n")
drf <- h2o.randomForest(
  model_id = "DRF", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, ntrees = 10000,
  score_tree_interval = 5, stopping_rounds = 3,
  balance_classes = TRUE, class_sampling_factors = samp_factors, max_after_balance_size = 0.5,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
tmp_path <- h2o.saveModel(h2o.getModel("DRF"), path = exp_dir, force = TRUE)
cat("XRT\n")
xrt <- h2o.randomForest(
  model_id = "XRT", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, ntrees = 10000, histogram_type = "Random",
  score_tree_interval = 5, stopping_rounds = 3,
  balance_classes = TRUE, class_sampling_factors = samp_factors, max_after_balance_size = 0.5,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
tmp_path <- h2o.saveModel(h2o.getModel("XRT"), path = exp_dir, force = TRUE)
# -----------------------
# get holdout predictions
# -----------------------
base_models <- as.list(c(
  unlist(deeplearning_1@model_ids),
  unlist(gbm@model_ids), paste0("GBM_", 1:5),
  unlist(xgboost@model_ids), paste0("XGBoost_", 1:3),
  "GLM",
  "DRF",
  "XRT"
))
for (model_id in base_models) {
  res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_", model_id))[3],
    col.names = model_id
  ))
}
# -----------------------
# super learner training
# -----------------------
```

```
  sl_iter <- 0
  cat(paste0("Super learner iteration ", sl_iter, " (", length(base_models), " models)\n"))
  sl <- h2o.stackedEnsemble(
    x = x, y = y, model_id = paste0("superlearner_iter_", sl_iter),
    training_frame = tmp, seed = 1,
    base_models = base_models,
    metalearner_algorithm = "glm",
    metalearner_nfolds = nfolds,
    keep_levelone_frame = TRUE,
    metalearner_params = list(
      standardize = TRUE, keep_cross_validation_predictions = TRUE,
      balance_classes = TRUE, class_sampling_factors = samp_factors,
      max_after_balance_size = 0.5
    )
  )
  tmp_path <- h2o.saveModel(h2o.getModel(paste0("superlearner_iter_", sl_iter)), path = exp_dir, force
↪  = TRUE)
  model_id <- paste0("metalearner_glm_superlearner_iter_", sl_iter)
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
  res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_", model_id))[3],
    col.names = paste0(model_id, "_", length(base_models), "_models")
  ))
  # ---------------------------------
  # super learner base model reduction
  # ---------------------------------
  while (TRUE) {
    meta <- h2o.getModel(paste0("metalearner_glm_superlearner_iter_", sl_iter, "_cv_1"))
    names <- meta@model$coefficients_table[, "names"]
    coeffs <- (meta@model$coefficients_table[, "standardized_coefficients"] > 0)
    for (j in 2:nfolds) {
      meta <- h2o.getModel(paste0("metalearner_glm_superlearner_iter_", sl_iter, "_cv_", j))
      names <- meta@model$coefficients_table[, "names"]
      coeffs <- coeffs + (meta@model$coefficients_table[, "standardized_coefficients"] > 0)
    }
    base_models_ <- as.list(names[coeffs >= ceiling(nfolds / 2) & names != "Intercept"])
    if (length(base_models_) == 0) {
      cat("No base models passing the threshold\n\n")
      break
    }
    if (sum(base_models %in% base_models_) == length(base_models)) {
      cat("No further reduction of base models\n\n")
```

```
    break
  }
  sl_iter <- sl_iter + 1
  base_models <- base_models_
  cat(paste0("Super learner iteration ", sl_iter, " (", length(base_models), " models)\n"))
  sl <- h2o.stackedEnsemble(
    x = x, y = y, model_id = paste0("superlearner_iter_", sl_iter),
    training_frame = tmp, seed = 1,
    base_models = base_models,
    metalearner_algorithm = "glm",
    metalearner_nfolds = nfolds,
    keep_levelone_frame = TRUE,
    metalearner_params = list(
      standardize = TRUE, keep_cross_validation_predictions = TRUE,
      balance_classes = TRUE, class_sampling_factors = samp_factors,
      max_after_balance_size = 0.5
    )
  )
  tmp_path <- h2o.saveModel(h2o.getModel(paste0("superlearner_iter_", sl_iter)), path = exp_dir,
↪  force = TRUE)
  model_id <- paste0("metalearner_glm_superlearner_iter_", sl_iter)
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
  res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_", model_id))[3],
    col.names = paste0(model_id, "_", length(base_models), "_models")
  ))
}
# ----------------------------------------
# super learner for homogeneous base models
# ----------------------------------------
# DeepLearning
base_models <- deeplearning_1@model_ids
cat(paste0("Super learner deep learning (", length(base_models), " models)\n"))
sl <- h2o.stackedEnsemble(
  x = x, y = y, model_id = "superlearner_deeplearning",
  training_frame = tmp, seed = 1,
  base_models = base_models,
  metalearner_algorithm = "glm",
  metalearner_nfolds = nfolds,
  keep_levelone_frame = TRUE,
  metalearner_params = list(
    standardize = TRUE, keep_cross_validation_predictions = TRUE,
```

```
    balance_classes = TRUE, class_sampling_factors = samp_factors,
    max_after_balance_size = 0.5
  )
)
tmp_path <- h2o.saveModel(h2o.getModel("superlearner_deeplearning"), path = exp_dir, force = TRUE)
model_id <- "metalearner_glm_superlearner_deeplearning"
tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_", model_id))[3],
  col.names = paste0(model_id, "_", length(base_models), "_models")
))
# GBM
base_models <- as.list(c(unlist(gbm@model_ids), paste0("GBM_", 1:5)))
cat(paste0("Super learner GBM (", length(base_models), " models)\n"))
sl <- h2o.stackedEnsemble(
  x = x, y = y, model_id = "superlearner_gbm",
  training_frame = tmp, seed = 1,
  base_models = base_models,
  metalearner_algorithm = "glm",
  metalearner_nfolds = nfolds,
  keep_levelone_frame = TRUE,
  metalearner_params = list(
    standardize = TRUE, keep_cross_validation_predictions = TRUE,
    balance_classes = TRUE, class_sampling_factors = samp_factors,
    max_after_balance_size = 0.5
  )
)
tmp_path <- h2o.saveModel(h2o.getModel("superlearner_gbm"), path = exp_dir, force = TRUE)
model_id <- "metalearner_glm_superlearner_gbm"
tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_", model_id))[3],
  col.names = paste0(model_id, "_", length(base_models), "_models")
))
# XGBoost
base_models <- as.list(c(unlist(xgboost@model_ids), paste0("XGBoost_", 1:3)))
cat(paste0("Super learner XGBoost (", length(base_models), " models)\n"))
sl <- h2o.stackedEnsemble(
  x = x, y = y, model_id = "superlearner_xgboost",
  training_frame = tmp, seed = 1,
  base_models = base_models,
  metalearner_algorithm = "glm",
  metalearner_nfolds = nfolds,
```

```
    keep_levelone_frame = TRUE,
    metalearner_params = list(
      standardize = TRUE, keep_cross_validation_predictions = TRUE,
      balance_classes = TRUE, class_sampling_factors = samp_factors,
      max_after_balance_size = 0.5
    )
  )
  tmp_path <- h2o.saveModel(h2o.getModel("superlearner_xgboost"), path = exp_dir, force = TRUE)
  model_id <- "metalearner_glm_superlearner_xgboost"
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
  res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_", model_id))[3],
    col.names = paste0(model_id, "_", length(base_models), "_models")
  ))
  write.csv(res, file = paste0(exp_dir, "/cv_holdout_predictions.csv"), row.names = FALSE)
  cat("\n\n")
  h2o.removeAll()
}
```

3.5.2 Inner cross-validation

```
load(file = "./rdata/var_reduct_tx_train_test_splits.RData")
for (i in 1:10) {
  cat(paste0("Outer training set ", i, "\n"))
  prefix <- paste0("outer_", i)
  exp_dir <- paste0("/Users/renee/Downloads/toxicity/", prefix)
  dir.create(exp_dir)
  train_tx_models(train_set = outer_splits[[i]][["train"]], exp_dir = exp_dir, prefix = prefix)
}
```

3.5.3 Training on whole data set

```
load(file = "./rdata/var_reduct_tx_train_test_splits.RData")
prefix <- "whole_data_set"
exp_dir <- paste0("/Users/renee/Downloads/toxicity/", prefix)
dir.create(exp_dir)
train_tx_models(train_set = tx_data, exp_dir = exp_dir, prefix = prefix)
```

```
# Keep track of grid models
dl_grid <- list()
gbm_grid <- list()
xgboost_grid <- list()
dl_grid_params <- list()
gbm_grid_params <- list()
xgboost_grid_params <- list()
for (i in 1:11) {
  if (i == 11) {
    cat(paste0("Whole data set\n"))
    prefix <- "whole_data_set"
  } else {
    cat(paste0("Outer training set ", i, "\n"))
    prefix <- paste0("outer_", i)
  }
  dir <- paste0("/Users/renee/Downloads/toxicity/", prefix)
  files <- list.files(dir)
  # Deep learning
  dl <- files[str_detect(files, "DeepLearning_model")]
  for (m in dl) {
    model <- h2o.loadModel(paste0(dir, "/", m))
    hs <- sha1(paste(c(
      model@allparameters$epsilon,
      model@allparameters$hidden,
      model@allparameters$hidden_dropout_ratios,
      model@allparameters$input_dropout_ratio,
      model@allparameters$rho
    ), collapse = " "))
    if (hs %in% names(dl_grid)) {
      dl_grid[[hs]] <- c(dl_grid[[hs]], m)
    } else {
      dl_grid[[hs]] <- c(m)
      dl_grid_params <- list.append(
        dl_grid_params,
        c(
          "epsilon" = model@allparameters$epsilon,
          "hidden" = paste0("[", paste(model@allparameters$hidden, collapse = ","), "]"),
          "hidden_dropout_ratios" = paste0(
            "[",
            paste(model@allparameters$hidden_dropout_ratios,
              collapse = ","
```

```
          ), "]"
        ),
        "input_dropout_ratio" = model@allparameters$input_dropout_ratio,
        "rho" = model@allparameters$rho
      )
    )
  }
}
h2o.removeAll()
# GBM
gbm <- files[str_detect(files, "GBM_model")]
for (m in gbm) {
  model <- h2o.loadModel(paste0(dir, "/", m))
  hs <- sha1(paste(c(
    model@allparameters$col_sample_rate,
    model@allparameters$col_sample_rate_per_tree,
    model@allparameters$max_depth,
    model@allparameters$min_rows,
    model@allparameters$min_split_improvement,
    model@allparameters$sample_rate
  ), collapse = " "))
  if (hs %in% names(gbm_grid)) {
    gbm_grid[[hs]] <- c(gbm_grid[[hs]], m)
  } else {
    gbm_grid[[hs]] <- c(m)
    gbm_grid_params <- list.append(
      gbm_grid_params,
      c(
        "col_sample_rate" = model@allparameters$col_sample_rate,
        "col_sample_rate_per_tree" = model@allparameters$col_sample_rate_per_tree,
        "max_depth" = model@allparameters$max_depth,
        "min_rows" = model@allparameters$min_rows,
        "min_split_improvement" = model@allparameters$min_split_improvement,
        "sample_rate" = model@allparameters$sample_rate
      )
    )
  }
}
h2o.removeAll()
# XGBoost
xgboost <- files[str_detect(files, "XGBoost_model")]
```

```
  for (m in xgboost) {
    model <- h2o.loadModel(paste0(dir, "/", m))
    hs <- sha1(paste(c(
      model@allparameters$booster,
      model@allparameters$col_sample_rate,
      model@allparameters$col_sample_rate_per_tree,
      model@allparameters$max_depth,
      model@allparameters$min_rows,
      model@allparameters$reg_alpha,
      model@allparameters$reg_lambda,
      model@allparameters$sample_rate
    ), collapse = " "))
    if (hs %in% names(xgboost_grid)) {
      xgboost_grid[[hs]] <- c(xgboost_grid[[hs]], m)
    } else {
      xgboost_grid[[hs]] <- c(m)
      xgboost_grid_params <- list.append(
        xgboost_grid_params,
        c(
          "booster" = model@allparameters$booster,
          "col_sample_rate" = model@allparameters$col_sample_rate,
          "col_sample_rate_per_tree" = model@allparameters$col_sample_rate_per_tree,
          "max_depth" = model@allparameters$max_depth,
          "min_rows" = model@allparameters$min_rows,
          "reg_alpha" = model@allparameters$reg_alpha,
          "reg_lambda" = model@allparameters$reg_lambda,
          "sample_rate" = model@allparameters$sample_rate
        )
      )
    }
  }
  h2o.removeAll()
}

dl_grid_params <- as.data.frame(t(data.frame(dl_grid_params)))
rownames(dl_grid_params) <- paste("Neural network grid model", 1:nrow(dl_grid_params))

gbm_grid_params <- as.data.frame(t(data.frame(gbm_grid_params)))
rownames(gbm_grid_params) <- paste("GBM grid model", 1:nrow(gbm_grid_params))

xgboost_grid_params <- as.data.frame(t(data.frame(xgboost_grid_params)))
```

```
rownames(xgboost_grid_params) <- paste("XGBoost grid model", 1:nrow(xgboost_grid_params))

write.csv(dl_grid_params, "./other_data/toxicity/neural_network_grid_params.csv")
write.csv(gbm_grid_params, "./other_data/toxicity/gbm_grid_params.csv")
write.csv(xgboost_grid_params, "./other_data/toxicity/xgboost_grid_params.csv")

save(dl_grid, gbm_grid, xgboost_grid, file = "./rdata/model_training_grid_models_tx.RData")
```

3.5.4 Model evaluation

```
model_evaluation <- function(holdout_pred, fold_asign, grid_name, grid_meta = NULL) {
  res_ <- list()
  model_num <- c()
  if (startsWith(grid_name, "Super learner")) {
    if (grid_name == "Super learner all models") {
      m <- colnames(holdout_pred)[startsWith(colnames(holdout_pred),
↪  "metalearner_glm_superlearner_iter_0")]
    } else if (grid_name == "Super learner final") {
      sl_models <- colnames(holdout_pred)[startsWith(colnames(holdout_pred),
↪  "metalearner_glm_superlearner_iter")]
      m <- sl_models[length(sl_models)]
    } else if (grid_name == "Super learner neural network") {
      m <- colnames(holdout_pred)[startsWith(colnames(holdout_pred),
↪  "metalearner_glm_superlearner_deeplearning")]
    } else if (grid_name == "Super learner GBM") {
      m <- colnames(holdout_pred)[startsWith(colnames(holdout_pred),
↪  "metalearner_glm_superlearner_gbm")]
    } else if (grid_name == "Super learner XGBoost") {
      m <- colnames(holdout_pred)[startsWith(colnames(holdout_pred),
↪  "metalearner_glm_superlearner_xgboost")]
    }
    logloss <- c()
    MCC <- c()
    F1 <- c()
    acc <- c()
    EF <- c()
    BEDROC <- c()
    threshold <- 0.5
    for (k in 1:10) {
```

```
    y_true <- holdout_pred[fold_assign == k, "category"]
    y_pred <- holdout_pred[fold_assign == k, m]
    y_pred_cls <- sapply(y_pred, function(x) if (x >= threshold) 1 else 0)
    logloss <- c(logloss, LogLoss(y_pred = y_pred, y_true = y_true))
    MCC <- c(MCC, mcc(preds = y_pred_cls, actuals = y_true))
    F1 <- c(F1, F1_Score(y_pred = y_pred_cls, y_true = y_true))
    acc <- c(acc, balanced_accuracy(y_pred = y_pred_cls, y_true = y_true))
    EF <- c(EF, enrichment_factor(x = y_pred, y = y_true, top = 0.05))
    BEDROC <- c(BEDROC, bedroc(x = y_pred, y = y_true, alpha = 20))
  }
  res_ <- list.append(res_, c(
    mean(logloss, na.rm = TRUE), sem(logloss),
    mean(MCC, na.rm = TRUE), sem(MCC),
    mean(F1, na.rm = TRUE), sem(F1),
    mean(acc, na.rm = TRUE), sem(acc),
    mean(EF, na.rm = TRUE), sem(EF),
    mean(BEDROC, na.rm = TRUE), sem(BEDROC),
    logloss, MCC, F1, acc, EF, BEDROC
  ))
} else {
  for (j in 1:length(grid_meta)) {
    g <- grid_meta[[j]]
    m <- intersect(colnames(holdout_pred), g)
    if (length(m) == 1) {
      model_num <- c(model_num, j)
      logloss <- c()
      MCC <- c()
      F1 <- c()
      acc <- c()
      EF <- c()
      BEDROC <- c()
      threshold <- 0.5
      for (k in 1:10) {
        y_true <- holdout_pred[fold_assign == k, "category"]
        y_pred <- holdout_pred[fold_assign == k, m]
        y_pred_cls <- sapply(y_pred, function(x) if (x >= threshold) 1 else 0)
        logloss <- c(logloss, LogLoss(y_pred = y_pred, y_true = y_true))
        MCC <- c(MCC, mcc(preds = y_pred_cls, actuals = y_true))
        F1 <- c(F1, F1_Score(y_pred = y_pred_cls, y_true = y_true))
        acc <- c(acc, balanced_accuracy(y_pred = y_pred_cls, y_true = y_true))
        EF <- c(EF, enrichment_factor(x = y_pred, y = y_true, top = 0.05))
```

```
          BEDROC <- c(BEDROC, bedroc(x = y_pred, y = y_true, alpha = 20))
        }
        res_ <- list.append(res_, c(
          mean(logloss, na.rm = TRUE), sem(logloss),
          mean(MCC, na.rm = TRUE), sem(MCC),
          mean(F1, na.rm = TRUE), sem(F1),
          mean(acc, na.rm = TRUE), sem(acc),
          mean(EF, na.rm = TRUE), sem(EF),
          mean(BEDROC, na.rm = TRUE), sem(BEDROC),
          logloss, MCC, F1, acc, EF, BEDROC
        ))
      }
    }
  }
  res <- as.data.frame(t(data.frame(res_)))
  colnames(res) <- c(
    "Log loss mean", "Log loss s.e.m.", "MCC mean", "MCC s.e.m.", "F_1 mean", "F_1 s.e.m.",
    "Balanced accuracy mean", "Balanced accuracy s.e.m.", "EF mean", "EF s.e.m.", "BEDROC mean",
    ↪  "BEDROC s.e.m.",
    paste("Log loss CV", 1:10), paste("MCC CV", 1:10), paste("F_1 CV", 1:10),
    paste("Balanced accuracy CV", 1:10), paste("EF CV", 1:10), paste("BEDROC CV", 1:10)
  )
  if (nrow(res) == 1) {
    rownames(res) <- grid_name
  } else {
    rownames(res) <- paste(grid_name, model_num)
  }
  return(res)
}
```

3.5.5 Inner loop model selection

```
load(file = "./rdata/model_training_grid_models_tx.RData")

for (i in 1:10) {
  holdout_pred <- read.csv(paste0("./other_data/toxicity/outer_", i, "/cv_holdout_predictions.csv"))
  fold_assign <- rep(1:10, ceiling(nrow(holdout_pred) / 10))[1:nrow(holdout_pred)]
  data_ <- list()
  # Deep learning grid models
```

```
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
    grid_name = "Neural network grid model",
    grid_meta = dl_grid
  ))
  # GBM grid models
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
    grid_name = "GBM grid model",
    grid_meta = gbm_grid
  ))
  # GBM default models
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
    grid_name = "GBM model",
    grid_meta = as.list(paste0("GBM_", 1:5))
  ))
  # XGBoost grid models
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
    grid_name = "XGBoost grid model",
    grid_meta = xgboost_grid
  ))
  # XGBoost default models
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
    grid_name = "XGBoost model",
    grid_meta = as.list(paste0("XGBoost_", 1:3))
  ))
  # GLM
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "GLM model",
↪   grid_meta = list("GLM")))
  # DRF
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "DRF model",
↪   grid_meta = list("DRF")))
  # XRT
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "XRT model",
↪   grid_meta = list("XRT")))
  # Super learner all
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪   all models"))
  # Super learner final
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪   final"))
  # Super learner deep learning
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪   neural network"))
```

```
  # Super learner GBM
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪  GBM"))
  # Super learner XGBoost
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪  XGBoost"))

  data <- do.call(rbind, data_)
  write.csv(data, paste0("./other_data/toxicity/outer_", i, "/cv_res.csv"))
}
```

3.5.5.1 CV 1

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
Super learner XGBoost	0.6797	0.7672	0.7382	0.6563	6	4	4	5	19

Showing 1 to 1 of 189 entries Previous 1 2 3 4 5 … 189 Next

3.5.5.2 CV 2

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
Super learner final	0.9097	0.3234	0.3442	0.2805	2	19	20	20	61

Showing 1 to 1 of 41 entries Previous 1 2 3 4 5 … 41 Next

3.5.5.3 CV 3

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
Super learner final					1	1	1	1	4

Showing 1 to 1 of 187 entries Previous 1 2 3 4 5 … 187 Next

3.5.5.4 CV 4

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
Super learner final		0.9214	0.8502	0.9698	1	5	10	3	19

Showing 1 to 1 of 74 entries Previous 1 2 3 4 5 … 74 Next

3.5.5.5 CV 5

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
Super learner final		0.8637	0.7502	0.9155	1	4	3	4	12

Showing 1 to 1 of 171 entries Previous 1 2 3 4 5 … 171 Next

3.5.5.6 CV 6

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
Super learner final	0.9698	0.6907	0.6587	0.894	4	2	2	5	13

Showing 1 to 1 of 194 entries Previous 1 2 3 4 5 … 194 Next

3.5.5.7 CV 7

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
Super learner final		0.5726	0.5966	0.6332	1	2	2	4	9

Showing 1 to 1 of 143 entries Previous 1 2 3 4 5 … 143 Next

3.5.5.8 CV 8

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
Super learner final		0.9729		0.748	1	2	1	8	12

Showing 1 to 1 of 176 entries Previous 1 2 3 4 5 … 176 Next

3.5.5.9 CV 9

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
Super learner final	1	0.8528	0.9126	0.4012	2	2	2	7	13

Showing 1 to 1 of 88 entries Previous 1 2 3 4 5 … 88 Next

3.5.5.10 CV 10

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
Super learner final				0.9214	1	1	1	5	8

Showing 1 to 1 of 139 entries Previous 1 2 3 4 5 … 139 Next

3.5.6 Final evaluation

```
load(file = "./rdata/var_reduct_tx_train_test_splits.RData")
load(file = "./rdata/model_training_grid_models_tx.RData")
dir <- "/Users/renee/Downloads/toxicity"
selected_models <- c(
  "Super learner XGBoost", "Super learner final", "Super learner final", "Super learner final",
  "Super learner final", "Super learner final", "Super learner final", "Super learner final",
  "Super learner final", "Super learner final"
)
```

```
logloss <- c()
MCC <- c()
F1 <- c()
acc <- c()
EF <- c()
BEDROC <- c()
threshold <- 0.5
for (i in 1:10) {
  holdout_pred <- read.csv(paste0("./other_data/toxicity/outer_", i, "/cv_holdout_predictions.csv"))
  if (startsWith(selected_models[i], "Neural network grid model")) {
    grid_num <- as.integer(str_extract(selected_models[i], "[0-9]+"))
    m <- intersect(colnames(holdout_pred), dl_grid[[grid_num]])
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/", m))
  } else if (startsWith(selected_models[i], "GBM grid model")) {
    grid_num <- as.integer(str_extract(selected_models[i], "[0-9]+"))
    m <- intersect(colnames(holdout_pred), gbm_grid[[grid_num]])
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/", m))
  } else if (startsWith(selected_models[i], "GBM model")) {
    grid_num <- as.integer(str_extract(selected_models[i], "[0-9]+"))
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/GBM_", grid_num))
  } else if (startsWith(selected_models[i], "XGBoost grid model")) {
    grid_num <- as.integer(str_extract(selected_models[i], "[0-9]+"))
    m <- intersect(colnames(holdout_pred), xgboost_grid[[grid_num]])
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/", m))
  } else if (startsWith(selected_models[i], "XGBoost model")) {
    grid_num <- as.integer(str_extract(selected_models[i], "[0-9]+"))
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/XGBoost_", grid_num))
  } else if (selected_models[i] == "Super learner all models") {
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/superlearner_iter_0"))
  } else if (selected_models[i] == "Super learner final") {
    files <- list.files(paste0(dir, "/outer_", i))
    files <- files[startsWith(files, "superlearner_iter")]
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/", files[length(files)]))
  } else if (selected_models[i] == "Super learner neural network") {
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/superlearner_deeplearning"))
  } else if (selected_models[i] == "Super learner GBM") {
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/superlearner_gbm"))
  } else if (selected_models[i] == "Super learner XGBoost") {
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/superlearner_xgboost"))
  } else {
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/", unlist(strsplit(selected_models[i], "
↪  "))[1]))
```

```
  }
  tmp <- as.h2o(subset(outer_splits[[i]][["test"]], select = -category))
  y_true <- as.numeric(outer_splits[[i]][["test"]]$category) - 1
  y_pred <- as.data.frame(as.data.frame(h2o.predict(model, tmp)))[, 3]
  y_pred_cls <- sapply(y_pred, function(x) if (x >= threshold) 1 else 0)
  logloss <- c(logloss, LogLoss(y_pred = y_pred, y_true = y_true))
  MCC <- c(MCC, mcc(preds = y_pred_cls, actuals = y_true))
  F1 <- c(F1, F1_Score(y_pred = y_pred_cls, y_true = y_true))
  acc <- c(acc, balanced_accuracy(y_pred = y_pred_cls, y_true = y_true))
  EF <- c(EF, enrichment_factor(x = y_pred, y = y_true, top = 0.05))
  BEDROC <- c(BEDROC, bedroc(x = y_pred, y = y_true, alpha = 20))
}
save(logloss, MCC, F1, acc, EF, BEDROC, file = "./rdata/final_evaluation_tx.RData")
```

```
load(file = "./rdata/final_evaluation_tx.RData")
data.frame(
  Metric = c("Log loss", "MCC", "F<sub>1</sub>", "Balanced accuracy", "EF", "BEDROC"),
  `Mean ± s.e.m.` = c(
    paste0(round(mean(logloss), 3), " ± ", round(sem(logloss), 3)),
    paste0(round(mean(MCC), 3), " ± ", round(sem(MCC), 3)),
    paste0(round(mean(F1), 3), " ± ", round(sem(F1), 3)),
    paste0(round(mean(acc), 3), " ± ", round(sem(acc), 3)),
    paste0(round(mean(EF), 3), " ± ", round(sem(EF), 3)),
    paste0(round(mean(BEDROC), 3), " ± ", round(sem(BEDROC), 3))
  ),
  check.names = FALSE
) %>%
  datatable(escape = FALSE, rownames = FALSE)
```

Show 10 entries Search:

Metric	Mean ± s.e.m.
Log loss	0.172 ± 0.008
MCC	0.879 ± 0.004
F_1	0.919 ± 0.002
Balanced accuracy	0.947 ± 0.002
EF	1.519 ± 0.017
BEDROC	0.993 ± 0.005

Showing 1 to 6 of 6 entries Previous 1 Next

3.5.7 Final model selection

```
load(file = "./rdata/model_training_grid_models_tx.RData")

holdout_pred <- read.csv("./other_data/toxicity/whole_data_set/cv_holdout_predictions.csv")
fold_assign <- rep(1:10, ceiling(nrow(holdout_pred) / 10))[1:nrow(holdout_pred)]
data_ <- list()
# Deep learning grid models
data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
  grid_name = "Neural network grid model",
  grid_meta = dl_grid
))
# GBM grid models
data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
  grid_name = "GBM grid model",
  grid_meta = gbm_grid
))
# GBM default models
data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
  grid_name = "GBM model",
  grid_meta = as.list(paste0("GBM_", 1:5))
))
# XGBoost grid models
data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
  grid_name = "XGBoost grid model",
  grid_meta = xgboost_grid
))
# XGBoost default models
data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
  grid_name = "XGBoost model",
  grid_meta = as.list(paste0("XGBoost_", 1:3))
))
# GLM
data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "GLM model",
↪  grid_meta = list("GLM")))
# DRF
data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "DRF model",
↪  grid_meta = list("DRF")))
# XRT
data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "XRT model",
↪  grid_meta = list("XRT")))
```

```
# Super learner all
data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner all
↪  models"))
# Super learner final
data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪  final"))
# Super learner deep learning
data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪  neural network"))
# Super learner GBM
data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪  GBM"))
# Super learner XGBoost
data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪  XGBoost"))

data <- do.call(rbind, data_)
write.csv(data, "./other_data/toxicity/whole_data_set/cv_res.csv")
```

```
data <- read.csv("./other_data/toxicity/whole_data_set/cv_res.csv", row.names = 1, check.names = FALSE)
statistical_testing(data,
  metric_vec = c("Log loss", "MCC", "F_1", "Balanced accuracy"),
  decreasing_vec = c(FALSE, TRUE, TRUE, TRUE)
)
```

Show 10 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
Super learner final		0.9457	0.9488	0.894	1	7	8	5	21
Super learner all models	0.5837	0.812	0.9274	0.9155	7	6	6	8	27
Super learner XGBoost	0.6353	0.8666	0.9274	0.9155	9	10	10	7	36
XGBoost grid model 49	0.6841	0.6543	0.7068	0.8093	4	13	17	6	40
XGBoost grid model 2	0.7939	0.7553	0.7628	0.7653	2	14	13	12	41
XGBoost grid model 71	0.7939	0.5224	0.5635	0.6677	5	22	23	16	66
XGBoost grid model 24	0.5429	0.5224	0.6543	0.7138	14	19	20	14	67
XGBoost grid model 68	0.8501	0.5075	0.4877	0.6894	6	26	29	26	87
XGBoost grid model 63	0.3083	0.5224	0.5261	0.6677	32	20	22	17	91
XGBoost grid model 17	0.461	0.5224	0.4877	0.8093	19	29	31	18	97

Showing 1 to 10 of 144 entries Previous 1 2 3 4 5 … 15 Next

```
# statistical_testing(data, metric_vec=c('Log loss', 'MCC', 'F_1', 'Balanced accuracy'),
#                     decreasing_vec=c(FALSE, TRUE, TRUE, TRUE),
#                   output_path='./other_data/toxicity/whole_data_set/cv_res_statistical_testing.csv')
```

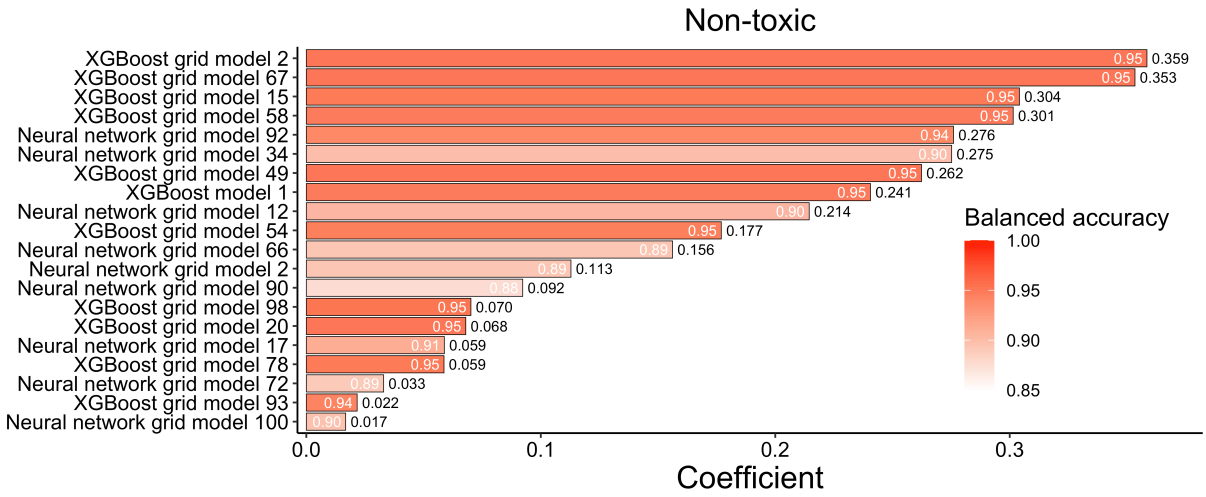
3.5.8 Final reduced SL

```
h2o.init(nthreads = -1)
model_path <- "/Users/renee/Downloads/toxicity/whole_data_set/superlearner_iter_3"
model <- h2o.loadModel(model_path)
load(file = "./rdata/model_training_grid_models_tx.RData")
data <- read.csv("./other_data/toxicity/whole_data_set/cv_res.csv", row.names = 1, check.names = FALSE)
```

```
df <- as.data.frame(model@model$metalearner_model@model$coefficients_table)[c("names",
↪  "standardized_coefficients")][-1, ]

for (i in 1:nrow(df)) {
  name <- df$names[i]
  if (startsWith(name, "DeepLearning_model")) {
    for (j in 1:length(dl_grid)) {
      if (name %in% dl_grid[[j]]) break
    }
    df$names[i] <- paste("Neural network grid model", j)
  } else if (startsWith(name, "GBM_model")) {
    for (j in 1:length(gbm_grid)) {
      if (name %in% gbm_grid[[j]]) break
    }
    df$names[i] <- paste("GBM grid model", j)
  } else if (startsWith(name, "XGBoost_model")) {
    for (j in 1:length(xgboost_grid)) {
      if (name %in% xgboost_grid[[j]]) break
    }
    df$names[i] <- paste("XGBoost grid model", j)
  } else {
    df$names[i] <- paste(strsplit(name, "_")[[1]], collapse = " model ")
  }
}

rownames(df) <- df$names
df <- merge(df, data["Balanced accuracy mean"], by = "row.names")[, -1]
df <- df[df$standardized_coefficients > 0, ]
```

```
p3 <- ggplot(df, aes(x = reorder(names, standardized_coefficients), y = standardized_coefficients,
↪  fill = `Balanced accuracy mean`)) +
  geom_col(color = "black", size = 0.2) +
  scale_fill_gradient(low = "white", high = "red", n.breaks = 4, limits = c(0.85, 1.00)) +
  geom_text(aes(label = sprintf("%.2f", `Balanced accuracy mean`)), nudge_y = -0.008, size = 3, color =
  ↪  "white") +
  geom_text(aes(label = sprintf("%.3f", standardized_coefficients)), nudge_y = 0.01, size = 3) +
  coord_flip() +
  scale_y_continuous(limits = c(0, max(df$standardized_coefficients) + 0.02), expand = c(0.01, 0)) +
  theme_bw() +
```

```
  theme(
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    strip.background = element_blank(),
    panel.border = element_blank(),
    axis.line = element_line(color = "black"),
    legend.text = element_text(colour = "black", size = 10),
    plot.title = element_text(hjust = 0.5, size = 18),
    plot.margin = ggplot2::margin(10, 10, 10, 0, "pt"),
    axis.title.x = element_text(colour = "black", size = 18),
    axis.title.y = element_text(colour = "black", size = 18),
    axis.text.x = element_text(colour = "black", size = 12),
    axis.text.y = element_text(colour = "black", size = 12),
    legend.title = element_text(size = 14),
    legend.position = c(0.85, 0.35)
  ) +
  guides(fill = guide_colorbar(title = "Balanced accuracy")) +
  xlab("") +
  ylab("Coefficient") +
  ggtitle("Non-toxic")
```




```
# Close h2o server
# h2o.shutdown()
sessionInfo()
```

```
## R version 4.2.2 (2022-10-31)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Big Sur ... 10.16
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] reshape2_1.4.4  DT_0.24         digest_0.6.29   stringr_1.4.1
##  [5] rlist_0.4.6.2   enrichvs_0.0.5  scales_1.2.1    mltools_0.3.5
##  [9] MLmetrics_1.1.1 h2o_3.38.0.2
##
## loaded via a namespace (and not attached):
##  [1] styler_1.8.0      xfun_0.32         bslib_0.4.0       purrr_0.3.4
##  [5] lattice_0.20-45   colorspace_2.0-3  vctrs_0.4.1       htmltools_0.5.3
##  [9] yaml_2.3.5        rlang_1.0.4       R.oo_1.25.0       jquerylib_0.1.4
## [13] R.utils_2.12.0    R.cache_0.16.0    lifecycle_1.0.1   plyr_1.8.7
## [17] munsell_0.5.0     R.methodsS3_1.8.2 htmlwidgets_1.5.4 codetools_0.2-18
## [21] evaluate_0.16     knitr_1.40        callr_3.7.2       fastmap_1.1.0
## [25] ps_1.7.1          crosstalk_1.2.0   highr_0.9         Rcpp_1.0.9
## [29] cachem_1.0.6      webshot_0.5.4     jsonlite_1.8.0    stringi_1.7.8
## [33] bookdown_0.28     processx_3.7.0    grid_4.2.2        cli_3.3.0
## [37] tools_4.2.2       bitops_1.0-7      magrittr_2.0.3    sass_0.4.2
## [41] RCurl_1.98-1.8    Matrix_1.5-1      data.table_1.14.2 rmarkdown_2.16
## [45] rstudioapi_0.14   R6_2.5.1          compiler_4.2.2
```

Section 4

Model validation and interpretation

```
library(h2o)
library(ggplot2)
library(ggpubr)
library(ggbeeswarm)
library(reticulate)
use_condaenv("/Users/renee/Library/r-miniconda/envs/peptide_engineering/bin/python")
library(reshape2)
library(scales)
library(ggforce)
library(cowplot)
library(plotly)
library(ggrepel)
```

```
import h2o
import pandas as pd
import numpy as np
import shap
import matplotlib.pyplot as plt
import session_info
```

4.1 Model validation

4.1.1 Melaning binding

```
# ================
# predicted values
# ================
mb_pred_ <- matrix(
  c(
```

```
    -0.0202, 1.3650, 2.1745, 2.3456, 2.5907, 2.6903, 2.9842, 3.1325, 3.5015, 3.6640, 3.7181, 4.0701,
    0.3223, 1.5153, 2.1196, 2.4047, 2.5032, 2.8177, 3.0653, 3.1491, 3.5578, 3.6465, 3.5599, 3.9509,
    0.6579, 2.1797, 1.9685, 2.3709, 2.6702, 3.0694, 2.9323, 3.3150, 3.4798, 3.5661, 3.6981, 3.9730,
    -0.0589, 1.3490, 2.4402, 2.1269, 2.5097, 2.6128, 2.9359, 3.3530, 3.8094, 3.5260, 3.7525, 3.8405,
    0.6532, 1.3684, 2.6170, 2.5270, 2.5228, 2.9967, 3.1125, 3.2106, 3.5952, 3.7345, 3.7353, 3.8400,
    0.2941, 1.1896, 2.1987, 2.7585, 3.1348, 2.9791, 3.2625, 3.0559, 3.5153, 3.6829, 3.9875, 4.0330,
    0.0156, 1.1113, 1.9551, 2.3677, 2.4250, 4.0111, 3.4891, 3.6521, 3.6634, 3.2731, 3.4504, 3.7587,
    0.0209, 1.9146, 2.2865, 2.5308, 2.7193, 3.0322, 3.1020, 3.3582, 3.2993, 3.8534, 4.9542, 3.5464
  ),
  nrow = 8, byrow = TRUE
)

new_mb_pred_ <- c(
  3.9318, 3.9366, 3.5531, 3.4385, 3.8434, 3.3893, 4.4159, 4.7549, 4.0004, 3.2785,
  4.5784, 4.6026, 3.6229, 3.8261, 3.6111, 5.0536, 4.8169, 5.9021, 4.0992, 4.3451,
  5.9626, 4.2730, 4.4115, 4.8889, 4.3264, 4.4830, 4.2847, 4.6585, 4.4568, 4.8513,
  4.1523
)

mb_pred <- c(as.vector(mb_pred_), new_mb_pred_)
for (i in 1:127) {
  if (mb_pred[i] < 0) mb_pred[i] <- NA
}
# Re-scale to 0-100
load(file = "./rdata/var_reduct_mb_train_test_splits.RData")
mb_pred <- rescale(mb_pred, to = c(0, 100), from = range(mb_data$log_intensity))
mb_pred[mb_pred > 100] <- 100

# ===================
# Experimental values
# ===================
replicate_1_1 <- matrix(
  c(
    0.9775, 50.1718, 39.9822, 66.0486, 46.0249, 40.8353, 58.6552, 72.8258, 55.1955, 59.8874, 78.4182,
    ↪  77.8258,
    -3.6197, 25.3377, 56.6884, 58.1339, -0.8945, 26.5936, 64.4609, 63.2050, 68.3472, 71.3566, 80.4325,
    ↪  70.3140,
    -34.7571, 26.0249, 52.3519, 70.4562, 63.1813, 69.1291, 73.1339, 65.6220, 73.9396, 76.5699,
    ↪  75.1955, 76.0249,
    -1.3922, 35.4325, 4.0344, 16.4514, 28.4656, 38.9870, 71.5699, 66.8306, 71.0723, 45.0059, 83.4182,
    ↪  75.9301,
```

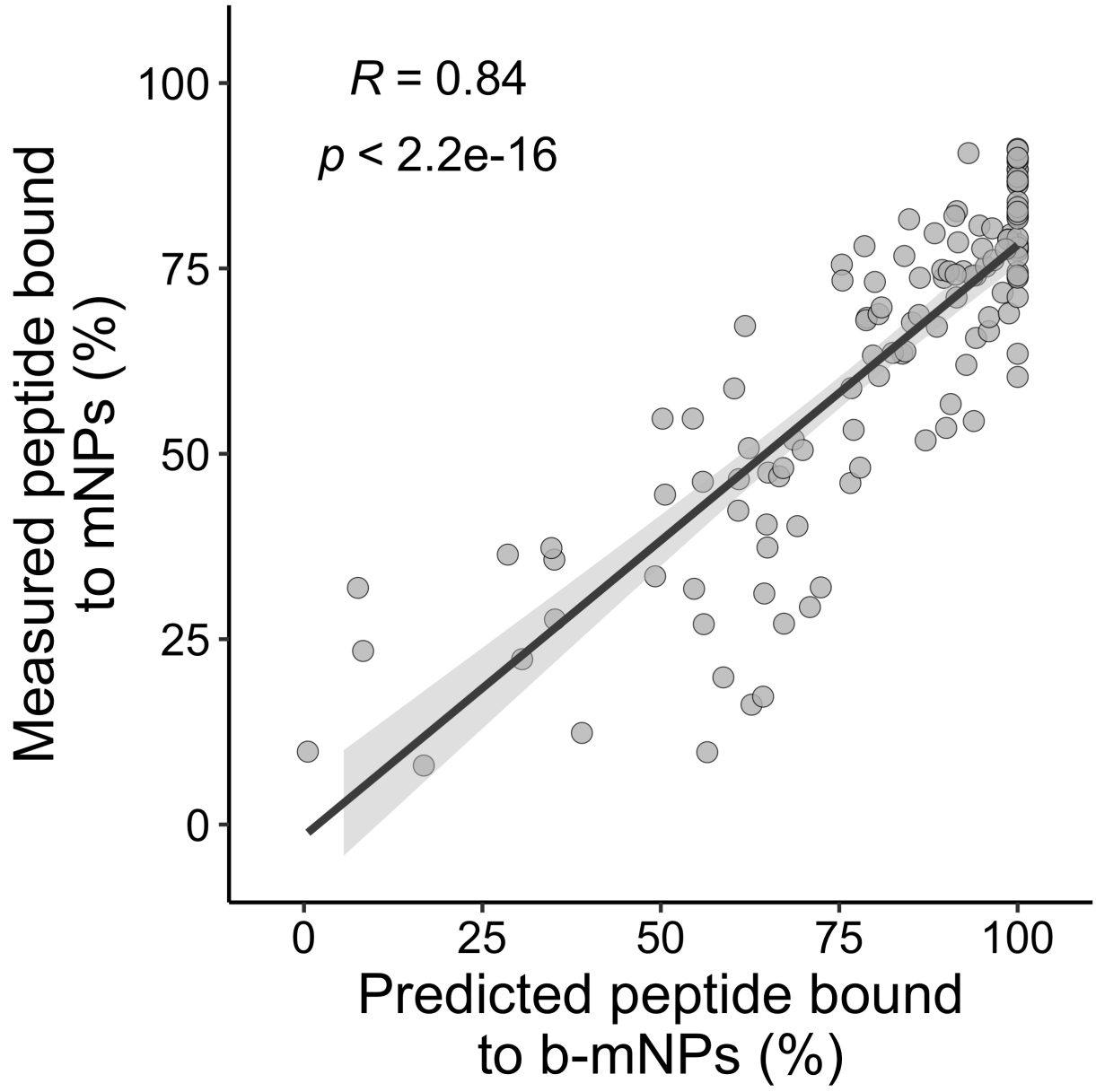
```
    -6.0604, 19.9111, 20.2666, 39.2476, 36.3329, 70.1955, 73.9633, 72.4467, 75.5746, 59.9348, 55.4799,
    ↪  75.8590,
    1.9727, 28.0628, -29.7097, 35.4562, 58.8211, 49.7927, 61.7121, 77.8732, 77.2097, 77.2571, 54.8874,
    ↪  76.0249,
    -23.3353, 33.2998, 45.9064, 36.1671, 36.4751, 74.5557, 70.6220, 48.2998, 69.6742, 76.9727,
    ↪  65.1244, 75.9064,
    2.2571, 27.1860, -1.6055, 42.3519, 48.7500, 47.8969, 62.2334, 70.6220, 79.7927, 75.9538, 77.4467,
    ↪  81.0960
  ),
  nrow = 8, byrow = TRUE
)

replicate_1_2 <- matrix(
  c(
    -7.7192, 54.0818, 45.1007, 62.8495, 50.1481, 37.6126, 59.5083, 71.7121, 57.7073, 59.4135, 77.0201,
    ↪  82.3756,
    21.4277, 10.8353, 54.6268, 65.7168, 15.6220, 35.1007, 64.8637, 67.8258, 66.5225, 72.7073, 83.1576,
    ↪  72.7784,
    -53.7855, 2.6600, 42.8732, 32.9443, 48.1576, 71.0960, 73.4656, 66.9964, 72.1386, 78.6789, 78.7500,
    ↪  77.1386,
    34.9822, 37.1623, 25.8590, 34.7453, 36.9964, 54.2239, 72.8969, 67.5415, 73.5604, 52.8021, 88.1339,
    ↪  80.0296,
    1.1434, 30.6694, 32.8969, 45.5983, 48.6552, 70.5036, 74.7927, 76.0249, 75.5983, 62.6600, 61.3803,
    ↪  81.4277,
    27.4467, 30.3614, 24.8164, 25.8116, 63.9633, 49.7927, 60.6457, 79.8637, 77.3282, 79.8164, 55.0770,
    ↪  77.1386,
    7.9206, 44.0107, 57.2571, 50.1955, 45.6220, 76.2618, 71.7358, 51.6410, 72.9680, 76.6410, 64.1291,
    ↪  76.5225,
    -3.7618, 32.5178, 28.9159, 47.3756, 50.5983, 49.4609, 62.7547, 71.7358, 83.3709, 81.7832, 83.3235,
    ↪  82.9917
  ),
  nrow = 8, byrow = TRUE
)

replicate_2_1 <- matrix(
  c(
    -21.6291, 12.7784, 51.7595, 30.0770, 33.6078, 28.2524, 47.4467, 51.8780, 45.4088, 64.9585,
    ↪  64.9585, 69.3187,
    24.2476, 7.0675, 56.8780, 71.7595, 25.8116, 40.1481, 69.2476, 71.3803, 74.9585, 73.1813, 81.8069,
    ↪  74.8400,
    44.5557, 31.2618, 43.6552, 39.0818, 48.3235, 67.8495, 76.2855, 64.5557, 74.5083, 76.0723, 75.4799,
    ↪  75.5746,
```

```
    17.3045, 45.8590, -7.5533, 34.7690, 28.3945, 41.9491, 73.3709, 69.8874, 70.1244, 62.9206, 71.8306,
    ↪  78.6552,
    17.0675, 34.8874, 19.2002, 35.7168, 43.5130, 2.1386, 70.8353, 50.4325, 73.8922, 68.7974, 76.4277,
    ↪  77.3756,
    51.4751, 2.5652, 13.2998, 25.7405, 58.7500, 50.5036, 64.6979, 76.6410, 67.4467, 80.3614, 63.3945,
    ↪  75.5746,
    -41.2500, 44.1528, 51.4040, 35.3377, 57.3993, 78.3945, 72.7784, 56.4751, 75.5746, 74.5320,
    ↪  68.5604, 75.0533,
    15.5036, 31.6173, 11.4040, 49.9111, 49.6505, 48.0865, 61.4040, 72.7784, 79.9822, 77.9443, 71.5699,
    ↪  79.8164
  ),
  nrow = 8, byrow = TRUE
)

replicate_2_2 <- matrix(
  c(
    13.5367, 25.8827, 48.0628, 76.2855, 58.0628, 54.2713, 69.7927, 78.9633, 55.6457, 78.2287, 80.5983,
    ↪  83.6789,
    51.5699, 6.1671, 50.8116, 73.3472, 28.4419, 26.0960, 73.4893, 76.5699, 74.5320, 78.7263, 85.4799,
    ↪  80.1718,
    41.7595, 48.1339, 39.0818, 43.8211, 47.9917, 65.2429, 79.1765, 73.6078, 78.5604, 82.7547, 81.2855,
    ↪  82.6126,
    38.0154, 30.6931, 42.3045, 41.2145, 30.7642, 57.1386, 75.6220, 70.5746, 72.1386, 66.0960, 78.2287,
    ↪  80.9775,
    19.6742, 25.1481, 36.0012, 28.8685, 33.3472, 70.0533, 73.1102, 55.5509, 73.3945, 74.7216, 80.4325,
    ↪  80.8827,
    46.7832, 28.2050, 30.6220, 30.2903, 60.5036, 34.1291, 66.9964, 77.6363, 76.3803, 85.6694, 68.1102,
    ↪  82.6126,
    6.4040, 24.1528, 64.3661, 47.6126, 63.5604, 81.3566, 79.8164, 61.2382, 78.0628, 78.5604, 70.7405,
    ↪  76.9017,
    25.3140, 42.5889, 40.6694, 50.2192, 53.0154, 47.0912, 66.6173, 79.8164, 83.3709, 82.6363, 77.1623,
    ↪  84.3898
  ),
  nrow = 8, byrow = TRUE
)

replicate_1_1 <- as.vector(replicate_1_1)
replicate_1_2 <- as.vector(replicate_1_2)
replicate_2_1 <- as.vector(replicate_2_1)
replicate_2_2 <- as.vector(replicate_2_2)
```

```
mb_true_ <- c()
for (i in 1:96) {
  res_ <- mean(c(
    mean(c(replicate_1_1[i], replicate_1_2[i])),
    mean(c(replicate_2_1[i], replicate_2_2[i]))
  ))
  if (res_ >= 0) {
    mb_true_ <- c(mb_true_, res_)
  } else {
    mb_true_ <- c(mb_true_, NA)
  }
}

new_replicate_1 <- c(
  79.9064, 80.2179, 75.9474, 79.1975, 71.6238, 53.5235, 83.5678, 81.2452, 74.2512, 68.4291,
  90.2855, 89.2512, 92.8356, 80.2853, 62.0912, 89.2853, 70.1267, 90.2891, 63.2853, 70.6812,
  89.2582, 85.2567, 78.3682, 90.5732, 82.5253, 84.2912, 82.5923, 90.8613, 73.2862, 90.4823,
  59.2715
)

new_replicate_2 <- c(
  77.6257, 82.2692, 71.8684, 79.1338, 67.7994, 52.3258, 88.2912, 79.6823, 78.5257, 62.5212,
  81.2842, 83.5812, 90.2512, 73.4821, 63.6843, 88.2965, 72.0582, 93.1925, 62.9142, 78.0396,
  86.9323, 82.5323, 72.6429, 83.6736, 89.5191, 86.8432, 81.5171, 92.1856, 92.6736, 93.9124,
  95.3929
)

new_replicate_3 <- c(
  79.7602, 83.5192, 74.6645, 80.8854, 67.3918, 49.5821, 93.1263, 84.2590, 76.9253, 60.4827,
  93.4825, 85.9901, 88.5705, 79.0125, 60.1825, 90.5812, 71.2281, 89.9251, 64.2681, 72.5712,
  83.9235, 84.2719, 71.1281, 95.2182, 90.0191, 89.1812, 82.7195, 89.9501, 83.8581, 85.3725,
  93.2801
)

new_mb_true_ <- c()
for (i in 1:31) {
  res_ <- mean(c(new_replicate_1[i], new_replicate_2[i], new_replicate_3[i]))
  if (res_ >= 0) {
    new_mb_true_ <- c(new_mb_true_, res_)
  } else {
    new_mb_true_ <- c(new_mb_true_, NA)
```

```
  }
}

mb_true <- c(mb_true_, new_mb_true_)
```

```
mb_df <- data.frame(mb_pred = mb_pred, mb_true = mb_true)
mb_df <- mb_df[complete.cases(mb_df), ]

p1 <- ggscatter(mb_df,
  x = "mb_pred", y = "mb_true",
  alpha = 0.8, size = 2.5, stroke = 0.2, color = "black", fill = "grey70", shape = 21,
  add = "reg.line", conf.int = TRUE, conf.int.level = 0.95, cor.coef = TRUE,
  add.params = list(color = "grey25", fill = "grey70"),
  cor.coeff.args = list(
    method = "pearson", label.x = 2, label.y = 95, label.sep = "\n",
    size = 4.5
  )
) +
  theme_bw() +
  theme(
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    strip.background = element_blank(),
    panel.border = element_blank(),
    axis.line = element_line(color = "black"),
    axis.title.x = element_text(colour = "black", size = 15),
    axis.title.y = element_text(colour = "black", size = 15),
    axis.text.x = element_text(colour = "black", size = 12),
    axis.text.y = element_text(colour = "black", size = 12)
  ) +
  xlab("Predicted peptide bound\nto b-mNPs (%)") +
  ylab("Measured peptide bound\nto mNPs (%)") +
  xlim(-5, 105) +
  ylim(-5, 105)
```



4.1.2 Melanin binding and cell-penetration

```
# ================
# predicted values
# ================
```

```
cpp_pred_ <- matrix(
  c(
    "Non-CPP", "Non-CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP",
    "Non-CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP",
    "Non-CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP",
    "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP",
    "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP",
    "Non-CPP", "Non-CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP",
    "Non-CPP", "Non-CPP", "Non-CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP",
    "Non-CPP", "Non-CPP", "Non-CPP", "Non-CPP", "Non-CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP",
    ↪  "CPP"
  ),
  nrow = 8, byrow = TRUE
)

new_cpp_pred_ <- c(
  "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP",
  "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP",
  "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP", "CPP",
  "CPP"
)

cpp_pred <- c(as.vector(cpp_pred_), new_cpp_pred_)

# ====================
# Experimental values
# ====================
non_induced_replicate_1 <- matrix(
  c(
    31.3344, 24.1344, 25.3344, 45.4944, 16.0944, 146.6544, 20.2944, 18.9744, 15.3744, 12.0144,
    ↪  19.8144, 15.3744,
    20.8944, 20.1744, 17.2944, 63.1344, 15.6144, 15.6144, 32.4144, 16.5744, 17.7744, 12.1344, 14.7744,
    ↪  55.9344,
    19.4544, 21.3744, 29.1744, 25.9344, 89.5344, 14.8944, 19.9344, 11.4144, 29.5344, 13.5744, 13.3344,
    ↪  33.9744,
    13.2144, 8.1744, 29.7744, -24.8256, 14.2944, 18.4944, 15.7344, 14.6544, 14.1744, 10.9344, 9.1344,
    ↪  55.6944,
    13.9344, 12.8544, 36.7344, 20.8944, 15.2544, 17.5344, 13.3344, 17.4144, 20.7744, 10.9344, 15.8544,
    ↪  19.4544,
    10.4544, 20.2944, 25.9344, 19.9344, 18.8544, 15.8544, 15.7344, 20.2944, 13.4544, 7.5744, 11.7744,
    ↪  33.9744,
```

```
    17.7744, 24.8544, 23.6544, 19.5744, 19.9344, 77.5344, 24.0144, 24.8544, 24.4944, 29.5344, 31.5744,
    ↪  53.8944,
    9.7344, 10.5744, 25.3344, 15.7344, 27.9744, 27.4944, 19.6944, 19.4544, 27.4944, 24.4944, 24.6144,
    ↪  26.2944
  ),
  nrow = 8, byrow = TRUE
)

non_induced_replicate_2 <- matrix(
  c(
    30.3744, 28.4544, 26.1744, 30.0144, 25.6944, 131.5344, 31.8144, 21.9744, 30.4944, 20.2944,
    ↪  17.7744, 26.7744,
    27.4944, 20.6544, 21.3744, 44.2944, 23.6544, 28.5744, 26.5344, 22.0944, 26.7744, 17.5344, 18.9744,
    ↪  63.0144,
    25.6944, 22.8144, 27.6144, 26.7744, 88.9344, 30.8544, 24.1344, 10.9344, 19.9344, 19.5744, 21.4944,
    ↪  25.2144,
    12.9744, 15.0144, 17.7744, 14.8944, 13.5744, 24.4944, 27.6144, 13.9344, 23.1744, 17.7744, 15.7344,
    ↪  45.6144,
    16.0944, 23.1744, 17.7744, 15.7344, 20.6544, 27.6144, 34.0944, 24.6144, 22.8144, 14.4144, 26.7744,
    ↪  25.6944,
    12.0144, 31.8144, 16.4544, 21.6144, 30.2544, 18.7344, 24.9744, 29.7744, 15.1344, 15.1344, 27.8544,
    ↪  25.2144,
    23.2944, 28.2144, 30.6144, 22.0944, 19.9344, 63.0144, 22.8144, 22.2144, 21.6144, 19.9344, 23.0544,
    ↪  30.9744,
    14.0544, 23.7744, 20.6544, 11.0544, 23.8944, 31.4544, 20.6544, 15.4944, 31.4544, 26.0544, 33.0144,
    ↪  37.0944
  ),
  nrow = 8, byrow = TRUE
)

non_induced_replicate_3 <- matrix(
  c(
    27.3744, 16.3344, 17.5344, 33.7344, 13.8144, 110.8944, 33.4944, 37.2144, 34.8144, 20.8944,
    ↪  39.4944, 36.1344,
    15.9744, 13.4544, 18.6144, 28.2144, 13.0944, 12.8544, 46.3344, 27.6144, 64.6944, 3.4944, 25.5744,
    ↪  40.8144,
    13.8144, 13.5744, 35.7744, 16.4544, 54.4944, 14.2944, 30.7344, 23.6544, 41.0544, 8.5344, 23.1744,
    ↪  52.2144,
    32.5344, 19.4544, 27.2544, 27.1344, 20.5344, 27.3744, 30.6144, 25.6944, 27.1344, 28.3344, 29.4144,
    ↪  27.3744,
    30.0144, 19.5744, 31.3344, 31.9344, 25.6944, 26.7744, 36.4944, 25.8144, 35.2944, 7.8144, 45.1344,
    ↪  21.7344,
```

```
    19.4544, 33.4944, 22.8144, 19.8144, 21.9744, 22.8144, 24.8544, 29.4144, 31.3344, 33.3744, 42.9744,
    ↪  52.2144,
    12.4944, 15.3744, 27.3744, 10.4544, 7.9344, 64.3344, 36.0144, 68.7744, 27.6144, 41.0544, 47.5344,
    ↪  70.2144,
    20.4144, 10.8144, 28.0944, 20.5344, 29.0544, 37.2144, 27.9744, 14.1744, 37.2144, 29.4144, 23.1744,
    ↪  25.5744
  ),
  nrow = 8, byrow = TRUE
)

non_induced_true_ <- c()
for (i in 1:96) {
  res_ <- mean(c(non_induced_replicate_1[i], non_induced_replicate_2[i], non_induced_replicate_3[i]))
  non_induced_true_ <- c(non_induced_true_, res_)
}

induced_replicate_1 <- matrix(
  c(
    171.4944, 32.1744, 23.6544, 24.7344, 47.1744, 404.5344, 224.2944, 71.8944, 442.4544, 394.5744,
    ↪  294.8544, 464.8944,
    39.4944, 24.2544, 59.4144, 55.6944, 31.9344, 182.2944, 195.2544, 434.1744, 486.8544, 406.8144,
    ↪  25.4544, 524.5344,
    28.8144, 76.2144, 160.6944, 32.6544, 424.8144, 60.4944, 474.1344, 281.2944, 240.2544, 226.8144,
    ↪  386.1744, 235.2144,
    20.7744, 28.9344, 43.6944, 34.2144, 20.6544, 45.9744, 297.0144, 30.9744, 540.3744, 235.4544,
    ↪  618.3744, 395.7744,
    24.7344, 13.9344, 30.2544, 23.5344, 14.8944, 17.1744, 40.5744, 132.4944, 410.4144, 53.2944,
    ↪  352.8144, 427.2144,
    13.3344, 224.2944, 28.2144, 48.0144, 62.5344, 25.6944, 352.6944, 245.6544, 80.8944, 313.5744,
    ↪  414.6144, 235.2144,
    20.8944, 29.8944, 36.9744, 19.6944, 12.7344, 422.1744, 341.8944, 120.2544, 333.6144, 281.2944,
    ↪  534.7344, 356.2944,
    15.7344, 66.4944, 33.3744, 21.0144, 73.5744, 17.8944, 108.9744, 116.1744, 618.3744, 538.0944,
    ↪  554.0544, 502.4544
  ),
  nrow = 8, byrow = TRUE
)

induced_replicate_2 <- matrix(
  c(
    39.1344, 30.1344, 34.5744, 367.2144, 36.7344, 185.4144, 192.9744, 434.2944, 376.2144, 305.4144,
    ↪  192.9744, 420.6144,
```

```
    41.4144, 19.4544, 101.6544, 370.0944, 43.9344, 231.1344, 110.1744, 476.8944, 395.6544, 315.6144,
     ↪  75.9744, 435.2544,
    26.8944, 60.6144, 29.8944, 200.1744, 460.5744, 265.9344, 541.6944, 310.2144, 282.6144, 404.4144,
     ↪  168.9744, 196.2144,
    12.6144, 137.7744, 16.0944, 16.2144, 47.8944, 24.7344, 274.4544, 62.0544, 247.9344, 209.2944,
     ↪  393.4944, 423.3744,
    22.2144, 21.3744, 11.1744, 9.1344, 128.4144, 19.6944, 58.9344, 233.1744, 437.4144, 58.6944,
     ↪  361.0944, 458.8944,
    16.8144, 33.3744, 12.3744, 24.3744, 134.1744, 25.9344, 364.4544, 207.8544, 91.2144, 383.6544,
     ↪  434.0544, 196.2144,
    11.8944, 21.7344, 17.4144, 16.8144, 19.9344, 525.4944, 422.4144, 519.9744, 379.5744, 282.6144,
     ↪  616.9344, 450.0144,
    19.2144, 42.7344, 19.4544, 25.9344, 54.7344, 42.9744, 73.2144, 135.4944, 524.0544, 471.7344,
     ↪  528.4944, 511.0944
  ),
  nrow = 8, byrow = TRUE
)

induced_replicate_3 <- matrix(
  c(
    30.3744, 20.2944, 233.8944, 83.8944, 24.2544, 25.3344, 136.2144, 374.8944, 334.5744, 399.6144,
     ↪  383.6544, 544.6944,
    18.9744, 21.1344, 20.0544, 401.7744, 22.6944, 62.7744, 51.4944, 430.6944, 303.7344, 466.6944,
     ↪  329.6544, 567.7344,
    21.8544, 75.3744, 51.9744, 150.3744, 411.4944, 122.0544, 458.2944, 127.8144, 38.4144, 269.8944,
     ↪  51.9744, 366.9744,
    12.1344, 6.6144, 8.8944, 16.8144, 13.3344, 24.4944, 276.9744, 77.4144, 163.6944, 297.8544,
     ↪  423.9744, 492.8544,
    13.9344, 1.9344, 10.4544, 10.5744, 19.0944, 17.6544, 33.0144, 86.7744, 43.2144, 316.0944,
     ↪  382.8144, 351.9744,
    1.4544, 136.2144, 9.9744, 13.4544, 101.5344, 18.1344, 403.9344, 183.0144, 389.7744, 70.4544,
     ↪  444.1344, 149.2944,
    7.5744, 15.3744, 141.7344, 15.0144, 12.1344, 423.4944, 422.7744, 94.8144, 269.8944, 289.2144,
     ↪  473.7744, 364.5744,
    0.8544, 25.5744, 14.4144, 8.4144, 37.3344, 37.2144, 250.5744, 159.3744, 608.5344, 466.2144,
     ↪  524.4144, 510.4944
  ),
  nrow = 8, byrow = TRUE
)

induced_true_ <- c()
```

```r
for (i in 1:96) {
  res_ <- mean(c(induced_replicate_1[i], induced_replicate_2[i], induced_replicate_3[i]))
  induced_true_ <- c(induced_true_, res_)
}

new_non_induced_replicate_1 <- c(
  94.6865, 84.2145, 58.6890, 69.1610, 35.1270, 12.4261, 19.8169, 12.4261, 12.4368, 19.8169,
  12.4261, 19.8169, 12.4368, 10.1149, 8.6644,  38.8384, 12.0117, 23.1802, 9.9236,  10.7153,
  55.7665, 14.1476, 24.8167, 11.4644, 21.2568, 41.3144, 11.8948, 17.2878, 21.1505, 12.9894,
  20.8902
)

new_non_induced_replicate_2 <- c(
  88.1415, 77.0150, 52.1440, 43.6355, 35.1270, 11.6132, 28.8335, 11.6132, 13.2019, 28.8335,
  11.6132, 28.8335, 13.2019, 11.6717, 10.0671, 30.3106, 13.4888, 34.5612, 12.9309, 12.6759,
  42.8393, 17.0753, 38.7481, 11.7673, 24.1153, 35.5229, 13.7651, 22.8933, 25.4702, 14.9234,
  29.0832
)

new_non_induced_replicate_3 <- c(
  64.5795, 63.2705, 48.2170, 52.1440, 27.9275, 15.0881, 40.0817, 15.0881, 13.9936, 40.0817,
  15.0881, 40.0817, 13.9936, 8.5847, 13.4835, 64.9266, 13.2444, 56.1065, 10.6090, 13.8660,
  69.1400, 21.0974, 67.0891, 10.8056, 25.8900, 55.3361, 16.4695, 24.6201, 42.7755, 14.1317,
  49.6828
)

new_non_induced_true_ <- c()
for (i in 1:31) {
  res_ <- mean(c(new_non_induced_replicate_1[i], new_non_induced_replicate_2[i],
↪  new_non_induced_replicate_3[i]))
  new_non_induced_true_ <- c(new_non_induced_true_, res_)
}

new_induced_replicate_1 <- c(
  775.3412, 608.3964, 376.3855, 403.2684, 450.2610, 104.3842, 610.9376, 104.3842, 206.0770, 610.9376,
  104.3842, 610.9376, 206.0770, 61.42746, 63.9640, 725.5767, 182.7868, 436.4094, 89.5931, 105.8666,
  643.6163, 295.6471, 536.2245, 141.8066, 415.8205, 549.5332, 258.9823, 532.0408, 386.370, 313.6335,
  596.1610
)

new_induced_replicate_2 <- c(
```

```
  720.8794, 670.3267, 402.6897, 361.0000, 423.5832, 95.4568, 353.5596, 95.4568, 173.0359, 353.5596,
  95.4568, 353.5596, 173.0359, 70.6842, 58.2979, 560.2724, 235.8898, 367.2965, 83.7953, 102.8688,
  552.6956, 249.6267, 461.8738, 114.8928, 335.8037, 455.3512, 205.2205, 327.7658, 417.4017, 186.3116,
  851.0120
)

new_induced_replicate_3 <- c(
  783.3711, 690.3612, 393.6819, 493.6712, 480.9125, 86.2659, 417.0393, 86.2659, 159.3978, 417.0393,
  86.2659, 417.0393, 159.3978, 60.1427, 53.6201, 512.8355, 120.0977, 394.4739, 100.9911, 112.4550,
  577.1388, 243.4006, 552.0697, 118.0552, 376.5862, 513.8896, 224.0635, 387.4242, 380.8028, 221.7246,
  766.4130
)

new_induced_true_ <- c()
for (i in 1:31) {
  res_ <- mean(c(new_induced_replicate_1[i], new_induced_replicate_2[i], new_induced_replicate_3[i]))
  new_induced_true_ <- c(new_induced_true_, res_)
}

cpp_true_non_induced <- c(non_induced_true_, new_non_induced_true_)
cpp_true_induced <- c(induced_true_, new_induced_true_)
```

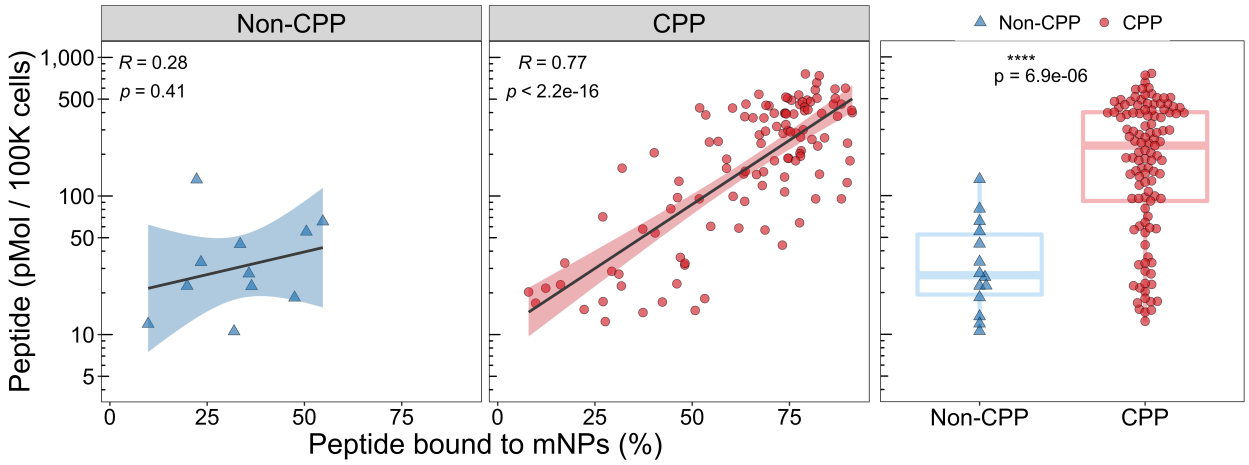
```
mb_cp_df <- data.frame(
  mb_true = rep(mb_true, 2),
  cpp_true = c(cpp_true_non_induced, cpp_true_induced),
  mb_pred = rep(mb_pred, 2),
  cpp_pred = factor(rep(cpp_pred, 2), levels = c("Non-CPP", "CPP")),
  cell_group = as.factor(c(
    rep("Non-induced", length(cpp_true_non_induced)),
    rep("Induced", length(cpp_true_induced))
  ))
)
```

4.1.3 Melanin-induced cells

```
mb_cp_df$log_cpp_true <- log10(mb_cp_df$cpp_true)
p2_1 <- ggscatter(mb_cp_df[mb_cp_df$cell_group == "Induced", ],
  x = "mb_true", y = "log_cpp_true",
```

```
  alpha = 0.7, size = 3, stroke = 0.3, color = "black", fill = "cpp_pred", shape = "cpp_pred",
  add = "reg.line", conf.int = TRUE, conf.int.level = 0.95, cor.coef = TRUE,
  add.params = list(color = "grey25"),
  cor.coeff.args = list(
    method = "pearson", label.x = 2.2, label.y = 2.85, label.sep = "\n",
    size = 5
  )
) +
  facet_grid(. ~ cpp_pred) +
  scale_fill_manual(values = c("#1f77b4", "#d62728")) +
  scale_shape_manual(values = c(24, 21)) +
  scale_y_continuous(
    limits = c(0.63, 3),
    breaks = c(1:3, log10(10^(1:4) / 2)),
    labels = sapply(
      c(10^(1:3), 10^(1:4) / 2),
      function(x) format(x, big.mark = ",", scientific = FALSE)
    )
  ) +
  annotation_logticks(sides = "l") +
  guides(fill = "none", shape = "none") +
  theme_bw() +
  theme(
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    axis.text.x = element_text(colour = "black", size = 17),
    axis.text.y = element_text(colour = "black", size = 17),
    strip.text = element_text(colour = "black", size = 20),
    panel.border = element_rect(colour = "black"),
    strip.text.y = element_blank(),
    axis.title = element_text(colour = "black", size = 22)
  ) +
  xlab("Peptide bound to mNPs (%)") +
  ylab("Peptide (pMol / 100K cells)")

set.seed(1)
pos <- position_jitter(width = 0.1, seed = 16)
p2_2 <- ggplot(mb_cp_df[mb_cp_df$cell_group == "Induced", ], aes(x = cpp_pred, y = log_cpp_true)) +
  geom_boxplot(aes(color = cpp_pred), outlier.color = NA, lwd = 1.5, show.legend = FALSE) +
  geom_beeswarm(aes(fill = cpp_pred, shape = cpp_pred),
    color = "black", alpha = 0.7, size = 3,
```

```
    stroke = 0.3, cex = 2, priority = "random"
  ) +
  stat_compare_means(
    label = "p.signif", size = 5, method = "wilcox.test", label.x.npc = 0.2,
    label.y = 2.92, hide.ns = TRUE
  ) +
  stat_compare_means(
    label = "p.format", size = 5, method = "wilcox.test", label.x.npc = 0.2,
    label.y = 2.82, hide.ns = TRUE
  ) +
  scale_color_manual(values = c("#cae6fa", "#f5b8b8")) +
  scale_fill_manual(values = c("#1f77b4", "#d62728")) +
  scale_shape_manual(values = c(24, 21)) +
  guides(fill = guide_legend(override.aes = list(shape = c(24, 21))), shape = "none") +
  scale_y_continuous(
    limits = c(0.63, 3),
    breaks = c(1:3, log10(10^(1:4) / 2)),
    labels = c(10^(1:3), 10^(1:4) / 2)
  ) +
  annotation_logticks(sides = "l") +
  theme_bw() +
  theme(
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    panel.border = element_rect(colour = "black"),
    strip.text = element_text(colour = "black", size = 20),
    axis.text.x = element_text(colour = "black", size = 18),
    legend.text = element_text(colour = "black", size = 15),
    axis.title.y = element_blank(),
    axis.text.y = element_blank(),
    axis.ticks.y = element_blank(),
    plot.margin = unit(c(1.15, 0.15, 0.53, 0), "cm"),
    legend.position = c(0.5, 1.05),
    legend.direction = "horizontal",
    legend.key = element_rect(fill = NA)
  ) +
  xlab("") +
  labs(fill = "", color = "", shape = "")

p2 <- ggarrange(p2_1, p2_2, widths = c(2.39, 1.02))
```

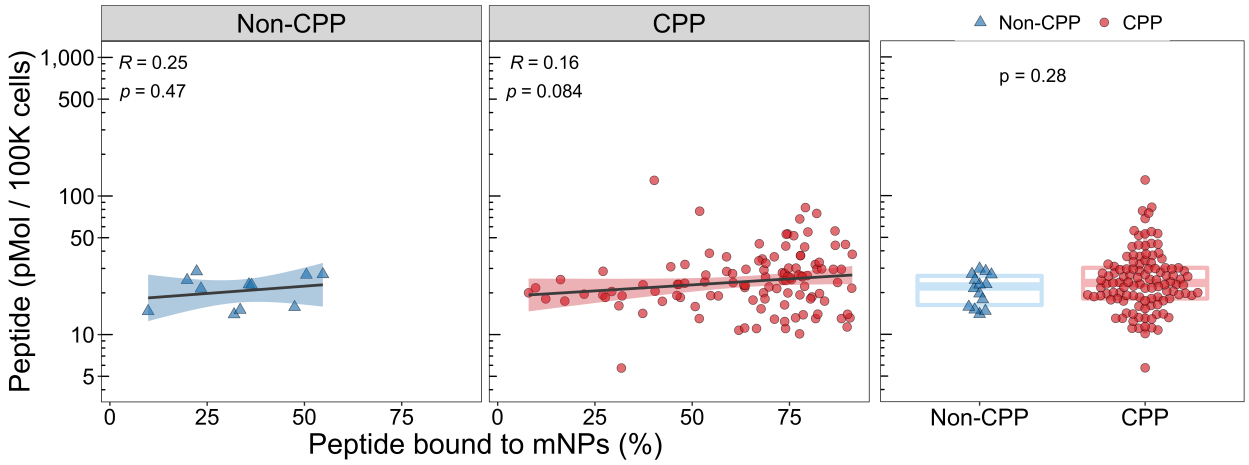


4.1.4 Non-melanin induced cells

```
p3_1 <- ggscatter(mb_cp_df[mb_cp_df$cell_group == "Non-induced", ],
  x = "mb_true", y = "log_cpp_true",
  alpha = 0.7, size = 3, stroke = 0.3, color = "black", fill = "cpp_pred", shape = "cpp_pred",
  add = "reg.line", conf.int = TRUE, conf.int.level = 0.95, cor.coef = TRUE,
  add.params = list(color = "grey25"),
  cor.coeff.args = list(
    method = "pearson", label.x = 2.2, label.y = 2.85, label.sep = "\n",
    size = 5
  )
) +
  facet_grid(. ~ cpp_pred) +
  scale_fill_manual(values = c("#1f77b4", "#d62728")) +
  scale_shape_manual(values = c(24, 21)) +
  scale_y_continuous(
    limits = c(0.63, 3),
    breaks = c(1:3, log10(10^(1:4) / 2)),
    labels = sapply(
      c(10^(1:3), 10^(1:4) / 2),
      function(x) format(x, big.mark = ",", scientific = FALSE)
    )
  ) +
  annotation_logticks(sides = "l") +
  guides(fill = "none", shape = "none") +
  theme_bw() +
  theme(
```

```
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    axis.text.x = element_text(colour = "black", size = 17),
    axis.text.y = element_text(colour = "black", size = 17),
    strip.text = element_text(colour = "black", size = 20),
    panel.border = element_rect(colour = "black"),
    strip.text.y = element_blank(),
    axis.title = element_text(colour = "black", size = 22)
  ) +
  xlab("Peptide bound to mNPs (%)") +
  ylab("Peptide (pMol / 100K cells)")

set.seed(1)
pos <- position_jitter(width = 0.1, seed = 16)
p3_2 <- ggplot(mb_cp_df[mb_cp_df$cell_group == "Non-induced", ], aes(x = cpp_pred, y = log_cpp_true)) +
  geom_boxplot(aes(color = cpp_pred), outlier.color = NA, lwd = 1.5, show.legend = FALSE) +
  geom_beeswarm(aes(fill = cpp_pred, shape = cpp_pred),
    color = "black", alpha = 0.7, size = 3,
    stroke = 0.3, cex = 2, priority = "random"
  ) +
  stat_compare_means(
    label = "p.signif", size = 5, method = "wilcox.test", label.x.npc = 0.2,
    label.y = 2.92, hide.ns = TRUE
  ) +
  stat_compare_means(
    label = "p.format", size = 5, method = "wilcox.test", label.x.npc = 0.2,
    label.y = 2.82, hide.ns = TRUE
  ) +
  scale_color_manual(values = c("#cae6fa", "#f5b8b8")) +
  scale_fill_manual(values = c("#1f77b4", "#d62728")) +
  scale_shape_manual(values = c(24, 21)) +
  guides(fill = guide_legend(override.aes = list(shape = c(24, 21))), shape = "none") +
  scale_y_continuous(
    limits = c(0.63, 3),
    breaks = c(1:3, log10(10^(1:4) / 2)),
    labels = c(10^(1:3), 10^(1:4) / 2)
  ) +
  annotation_logticks(sides = "l") +
  theme_bw() +
  theme(
    panel.grid.major = element_blank(),
```

```
    panel.grid.minor = element_blank(),
    panel.border = element_rect(colour = "black"),
    strip.text = element_text(colour = "black", size = 20),
    axis.text.x = element_text(colour = "black", size = 18),
    legend.text = element_text(colour = "black", size = 15),
    axis.title.y = element_blank(),
    axis.text.y = element_blank(),
    axis.ticks.y = element_blank(),
    plot.margin = unit(c(1.15, 0.15, 0.53, 0), "cm"),
    legend.position = c(0.5, 1.05),
    legend.direction = "horizontal",
    legend.key = element_rect(fill = NA)
  ) +
  xlab("") +
  labs(fill = "", color = "", shape = "")

p3 <- ggarrange(p3_1, p3_2, widths = c(2.39, 1.02))
```



4.2 Overall model interpretation

4.2.1 Melanin binding (regression)

```
# Save train and test sets in csv format
load(file = "./rdata/var_reduct_mb_train_test_splits.RData")
for (i in 1:10) {
```

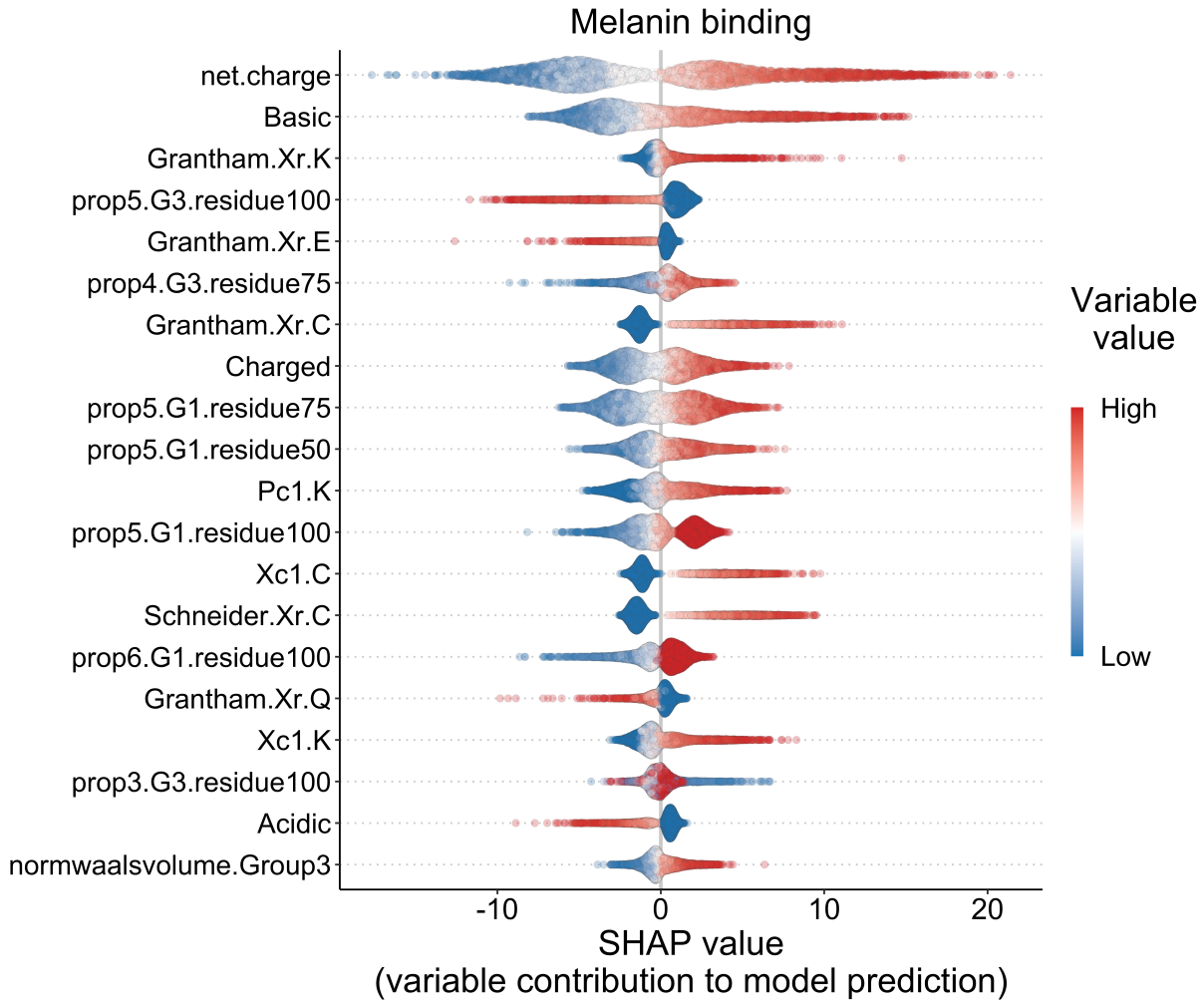
```
  prefix <- paste0("outer_", i)
  exp_dir <- paste0("/Users/renee/Downloads/melanin_binding/", prefix)
  write.csv(outer_splits[[i]]$train, file = paste0(exp_dir, "/train_set.csv"))
  write.csv(outer_splits[[i]]$test, file = paste0(exp_dir, "/test_set.csv"))
}
exp_dir <- paste0("/Users/renee/Downloads/melanin_binding/whole_data_set")
write.csv(mb_data, file = paste0(exp_dir, "/train_set.csv"))
```

```
h2o.init(nthreads=-1)

dir_path = '/Users/renee/Downloads/melanin_binding'
model_names = ['superlearner_iter_3', 'superlearner_iter_2', 'superlearner_iter_2',
↪  'superlearner_iter_3',
               'superlearner_iter_3', 'superlearner_iter_3', 'superlearner_iter_3',
                ↪  'superlearner_iter_3',
               'superlearner_iter_4', 'superlearner_iter_3']
shap_res = []
for i in range(10):
    print('Iter ' + str(i + 1) + ':')
    train = pd.read_csv(dir_path + '/outer_' + str(i + 1) + '/train_set.csv', index_col=0)
    X_train = train.iloc[:, :-1]
    test = pd.read_csv(dir_path + '/outer_' + str(i + 1) + '/test_set.csv', index_col=0)
    X_test = test.iloc[:, :-1]
    model = h2o.load_model(dir_path + '/outer_' + str(i + 1) + '/' + model_names[i])
    def model_predict_(data):
        data = pd.DataFrame(data, columns=X_train.columns)
        h2o_data = h2o.H2OFrame(data)
        res = model.predict(h2o_data)
        res = res.as_data_frame()
        return(res)
    np.random.seed(1)
    X_train_ = X_train.iloc[np.random.choice(X_train.shape[0], 100, replace=False), :]
    explainer = shap.KernelExplainer(model_predict_, X_train_, link='identity')
    shap_values = explainer.shap_values(X_test, nsamples=1000)
    shap_res.append(shap_values)
    h2o.remove_all()

shap_res = pd.DataFrame(np.vstack(shap_res), columns=X_train.columns)
shap_res.to_csv('./other_data/mb_shap_values_cv_data_sets.csv', index=False)
```

```
load(file = "./rdata/var_reduct_mb_train_test_splits.RData")
data <- do.call(rbind, lapply(outer_splits, function(x) x$test[-ncol(mb_data)]))
data <- apply(data, 2, function(x) rank(x))
data <- apply(data, 2, function(x) rescale(x, to = c(0, 100)))
shap_values <- as.matrix(read.csv("./other_data/mb_shap_values_cv_data_sets.csv", check.names =
↪  FALSE))
shap_values <- as.data.frame(rescale(shap_values, to = c(0, 100), from = range(mb_data$log_intensity)))

var_diff <- apply(shap_values, 2, function(x) max(x) - min(x))
var_diff <- var_diff[order(-var_diff)]
df <- data.frame(
  variable = melt(shap_values)$variable,
  shap = melt(shap_values)$value,
  variable_value = melt(data)$value
)
```

```
top_vars <- names(var_diff)[1:20]
df <- df[df$variable %in% top_vars, ]
df$variable <- factor(df$variable, levels = rev(top_vars), labels = rev(top_vars))

p1 <- ggplot(df, aes(x = shap, y = variable)) +
  geom_hline(yintercept = top_vars, linetype = "dotted", color = "grey80") +
  geom_vline(xintercept = 0, color = "grey80", size = 1) +
  geom_point(aes(fill = variable_value), color = "grey30", shape = 21, alpha = 0.3, size = 2, position
  ↪  = "auto", stroke = 0.1) +
  scale_fill_gradient2(low = "#1f77b4", mid = "white", high = "#d62728", midpoint = 50, breaks = c(0,
  ↪  100), labels = c("Low", "High")) +
  theme_bw() +
  theme(
    plot.margin = ggplot2::margin(1.5, 8, 1.5, -3),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    strip.background = element_blank(),
    panel.border = element_blank(),
    axis.line = element_line(color = "black"),
    plot.title = element_text(hjust = 0.5, size = 20),
    axis.title.x = element_text(hjust = 0.5, colour = "black", size = 20),
    axis.title.y = element_text(colour = "black", size = 20),
    axis.text.x = element_text(colour = "black", size = 17),
```

```
  axis.text.y = element_text(colour = "black", size = 16),
  legend.title = element_text(size = 20),
  legend.text = element_text(colour = "black", size = 16),
  legend.title.align = 0.5
) +
guides(
  fill = guide_colourbar("Variable\nvalue\n", ticks = FALSE, barheight = 10, barwidth = 0.5),
  color = "none"
) +
xlab("SHAP value\n(variable contribution to model prediction)") +
ylab("") +
ggtitle("Melanin binding")
```



4.2.2 Cell-penetration (classification)

```r
# Save train and test sets in csv format
load(file = "./rdata/var_reduct_cpp_train_test_splits.RData")
for (i in 1:10) {
  prefix <- paste0("outer_", i)
  exp_dir <- paste0("/Users/renee/Downloads/cell_penetration/", prefix)
  write.csv(outer_splits[[i]]$train, file = paste0(exp_dir, "/train_set.csv"))
  write.csv(outer_splits[[i]]$test, file = paste0(exp_dir, "/test_set.csv"))
}
exp_dir <- paste0("/Users/renee/Downloads/cell_penetration/whole_data_set")
write.csv(cpp_data, file = paste0(exp_dir, "/train_set.csv"))
```

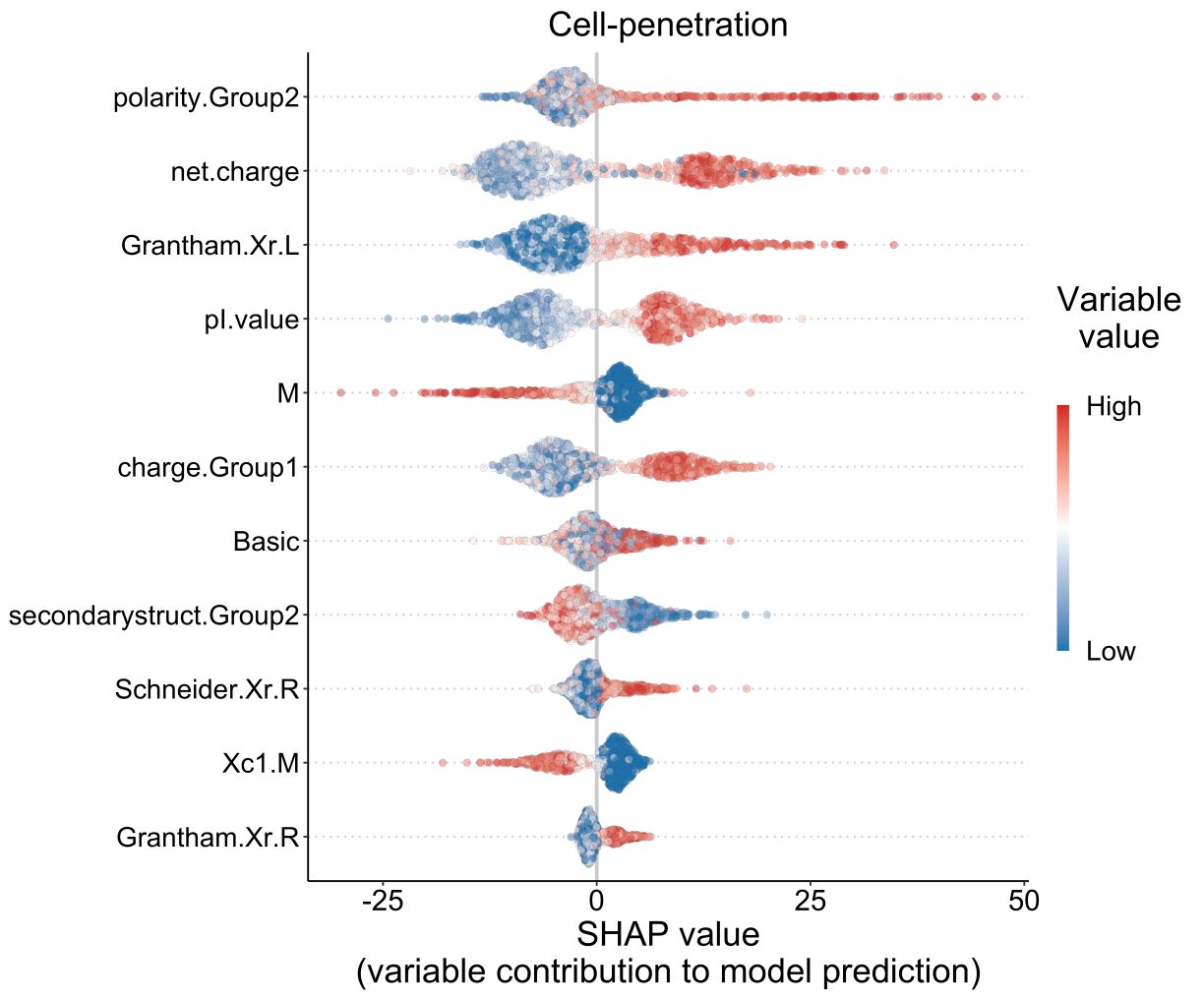
```python
h2o.init(nthreads=-1)

dir_path = '/Users/renee/Downloads/cell_penetration'
model_names = ['GBM_model_1669777615732_57632', 'GBM_model_1669777615732_225117',
↪  'GBM_model_1669777615732_387368',
               'GBM_model_1669777615732_552677', 'GBM_model_1669777615732_717390',
                ↪  'GBM_model_1669777538393_67146',
               'GBM_model_1669777538393_246060', 'GBM_model_1669777538393_420437',
                ↪  'GBM_model_1669777538393_602358',
               'GBM_model_1669777538393_782798']
shap_res = []
for i in range(10):
    print('Iter ' + str(i + 1) + ':')
    train = pd.read_csv(dir_path + '/outer_' + str(i + 1) + '/train_set.csv', index_col=0)
    X_train = train.iloc[:, :-1]
    test = pd.read_csv(dir_path + '/outer_' + str(i + 1) + '/test_set.csv', index_col=0)
    X_test = test.iloc[:, :-1]
    model = h2o.load_model(dir_path + '/outer_' + str(i + 1) + '/' + model_names[i])
    def model_predict_(data):
        data = pd.DataFrame(data, columns=X_train.columns)
        h2o_data = h2o.H2OFrame(data)
        res = model.predict(h2o_data)
        res = res.as_data_frame().iloc[:,2]
        return(res)
    np.random.seed(1)
    X_train_ = X_train.iloc[np.random.choice(X_train.shape[0], 100, replace=False), :]
    explainer = shap.KernelExplainer(model_predict_, X_train_, link='identity')
```

```
    shap_values = explainer.shap_values(X_test, nsamples=1000)
    shap_res.append(shap_values)
    h2o.remove_all()

shap_res = pd.DataFrame(np.vstack(shap_res), columns=X_train.columns)
shap_res.to_csv('./other_data/cpp_shap_values_cv_data_sets.csv', index=False)
```

```
load(file = "./rdata/var_reduct_cpp_train_test_splits.RData")
data <- do.call(rbind, lapply(outer_splits, function(x) x$test[-ncol(cpp_data)]))
data <- apply(data, 2, function(x) rank(x))
data <- apply(data, 2, function(x) rescale(x, to = c(0, 100)))
shap_values <- read.csv("./other_data/cpp_shap_values_cv_data_sets.csv", check.names = FALSE) * 100

var_diff <- apply(shap_values, 2, function(x) max(x) - min(x))
var_diff <- var_diff[order(-var_diff)]
df <- data.frame(
  variable = melt(shap_values)$variable,
  shap = melt(shap_values)$value,
  variable_value = melt(data)$value
)
```

```
top_vars <- names(var_diff)[1:11]
df <- df[df$variable %in% top_vars, ]
df$variable <- factor(df$variable, levels = rev(top_vars), labels = rev(top_vars))

p2 <- ggplot(df, aes(x = shap, y = variable)) +
  geom_hline(yintercept = top_vars, linetype = "dotted", color = "grey80") +
  geom_vline(xintercept = 0, color = "grey80", size = 1) +
  geom_point(aes(fill = variable_value), color = "grey30", shape = 21, alpha = 0.5, size = 2, position
  ↪  = "auto", stroke = 0.1) +
  scale_fill_gradient2(low = "#1f77b4", mid = "white", high = "#d62728", midpoint = 50, breaks = c(0,
  ↪  100), labels = c("Low", "High")) +
  theme_bw() +
  theme(
    plot.margin = ggplot2::margin(1.5, 8, 1.5, -3),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    strip.background = element_blank(),
```

```
  panel.border = element_blank(),
  axis.line = element_line(color = "black"),
  plot.title = element_text(hjust = 0.5, size = 20),
  axis.title.x = element_text(hjust = 0.5, colour = "black", size = 20),
  axis.title.y = element_text(colour = "black", size = 20),
  axis.text.x = element_text(colour = "black", size = 17),
  axis.text.y = element_text(colour = "black", size = 16),
  legend.title = element_text(size = 20),
  legend.text = element_text(colour = "black", size = 16),
  legend.title.align = 0.5
) +
guides(
  fill = guide_colourbar("Variable\nvalue\n", ticks = FALSE, barheight = 10, barwidth = 0.5),
  color = "none"
) +
xlab("SHAP value\n(variable contribution to model prediction)") +
ylab("") +
ggtitle("Cell-penetration")
```



4.2.3 Toxicity (classification)

```
# Save train and test sets in csv format
load(file = "./rdata/var_reduct_tx_train_test_splits.RData")
for (i in 1:10) {
  prefix <- paste0("outer_", i)
  exp_dir <- paste0("/Users/renee/Downloads/toxicity/", prefix)
  write.csv(outer_splits[[i]]$train, file = paste0(exp_dir, "/train_set.csv"))
  write.csv(outer_splits[[i]]$test, file = paste0(exp_dir, "/test_set.csv"))
}
exp_dir <- paste0("/Users/renee/Downloads/toxicity/whole_data_set")
write.csv(tx_data, file = paste0(exp_dir, "/train_set.csv"))
```

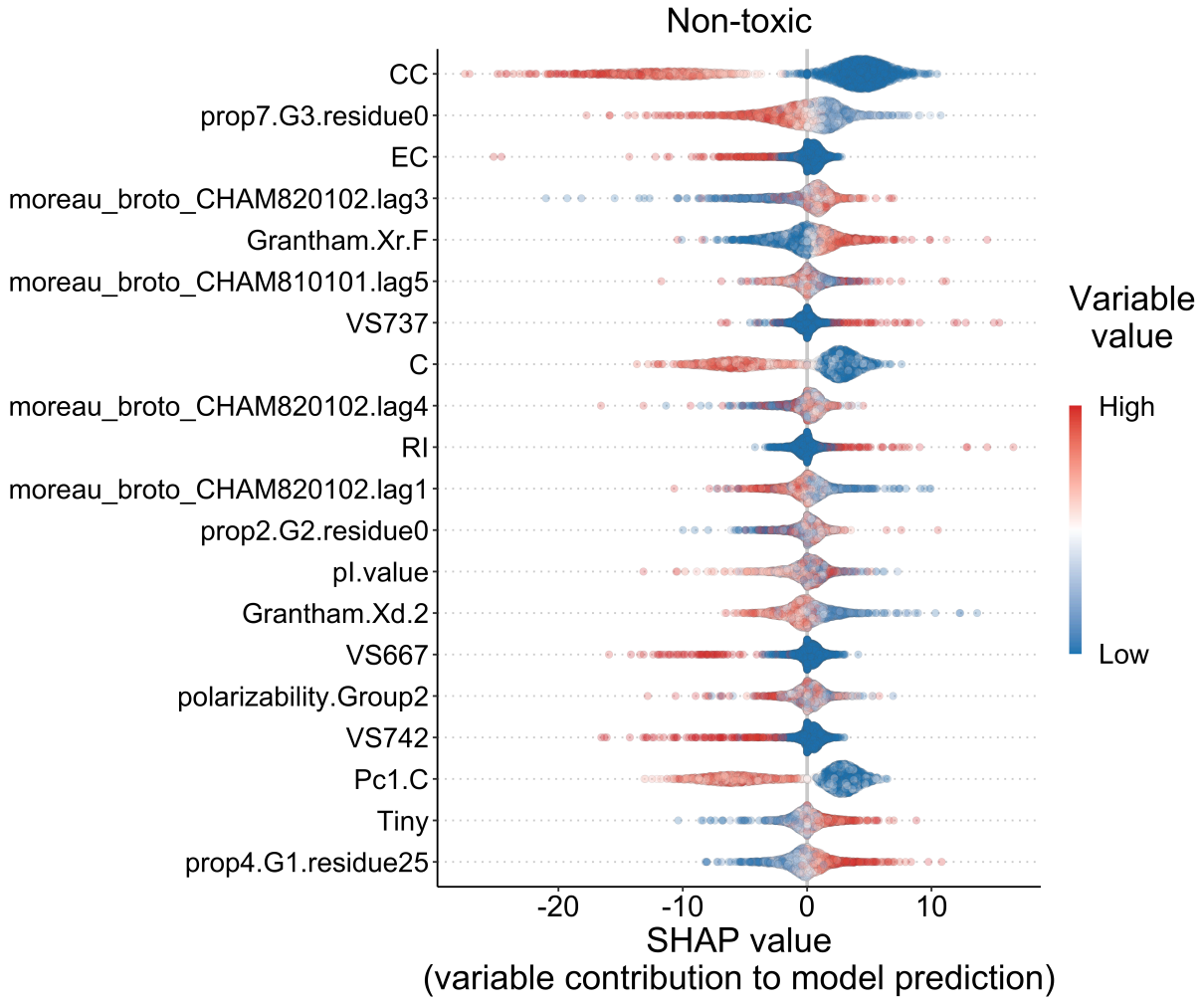
```
h2o.init(nthreads=-1)

dir_path = '/Users/renee/Downloads/toxicity'
model_names = ['superlearner_iter_4', 'superlearner_iter_3', 'superlearner_iter_3',
↪  'superlearner_iter_3',
               'superlearner_iter_3', 'superlearner_iter_3', 'superlearner_iter_3',
                ↪  'superlearner_iter_4',
               'superlearner_iter_3', 'superlearner_iter_4']
shap_res = []
for i in range(10):
    print('Iter ' + str(i + 1) + ':')
    train = pd.read_csv(dir_path + '/outer_' + str(i + 1) + '/train_set.csv', index_col=0)
    X_train = train.iloc[:, :-1]
    test = pd.read_csv(dir_path + '/outer_' + str(i + 1) + '/test_set.csv', index_col=0)
    X_test = test.iloc[:, :-1]
    model = h2o.load_model(dir_path + '/outer_' + str(i + 1) + '/' + model_names[i])
    def model_predict_(data):
        data = pd.DataFrame(data, columns=X_train.columns)
        h2o_data = h2o.H2OFrame(data)
        res = model.predict(h2o_data)
        res = res.as_data_frame().iloc[:,2] # 0:toxic; 1:non-toxic
        return(res)
    np.random.seed(1)
    X_train_ = X_train.iloc[np.random.choice(X_train.shape[0], 100, replace=False), :]
    explainer = shap.KernelExplainer(model_predict_, X_train_, link='identity')
    shap_values = explainer.shap_values(X_test, nsamples=1000)
    shap_res.append(shap_values)
    h2o.remove_all()

shap_res = pd.DataFrame(np.vstack(shap_res), columns=X_train.columns)
shap_res.to_csv('./other_data/tx_shap_values_cv_data_sets.csv', index=False)
```

```
load(file = "./rdata/var_reduct_tx_train_test_splits.RData")
data <- do.call(rbind, lapply(outer_splits, function(x) x$test[-ncol(tx_data)]))
data <- apply(data, 2, function(x) rank(x))
data <- apply(data, 2, function(x) rescale(x, to = c(0, 100)))
shap_values <- read.csv("./other_data/tx_shap_values_cv_data_sets.csv", check.names = FALSE) * 100

set.seed(12)
```

```r
ind <- sample(1:nrow(data), 2500)
data <- data[ind, ]
shap_values <- shap_values[ind, ]

var_diff <- apply(shap_values, 2, function(x) max(x) - min(x))
var_diff <- var_diff[order(-var_diff)]
df <- data.frame(
  variable = melt(shap_values)$variable,
  shap = melt(shap_values)$value,
  variable_value = melt(data)$value
)
```

```r
top_vars <- names(var_diff)[1:20]
df <- df[df$variable %in% top_vars, ]
df$variable <- factor(df$variable, levels = rev(top_vars), labels = rev(top_vars))

p3 <- ggplot(df, aes(x = shap, y = variable)) +
  geom_hline(yintercept = top_vars, linetype = "dotted", color = "grey80") +
  geom_vline(xintercept = 0, color = "grey80", size = 1) +
  geom_point(aes(fill = variable_value), color = "grey30", shape = 21, alpha = 0.3, size = 2, position
  ↪  = "auto", stroke = 0.1) +
  scale_fill_gradient2(low = "#1f77b4", mid = "white", high = "#d62728", midpoint = 50, breaks = c(0,
  ↪  100), labels = c("Low", "High")) +
  theme_bw() +
  theme(
    plot.margin = ggplot2::margin(1.5, 8, 1.5, -3),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    strip.background = element_blank(),
    panel.border = element_blank(),
    axis.line = element_line(color = "black"),
    plot.title = element_text(hjust = 0.5, size = 20),
    axis.title.x = element_text(hjust = 0.5, colour = "black", size = 20),
    axis.title.y = element_text(colour = "black", size = 20),
    axis.text.x = element_text(colour = "black", size = 17),
    axis.text.y = element_text(colour = "black", size = 16),
    legend.title = element_text(size = 20),
    legend.text = element_text(colour = "black", size = 16),
    legend.title.align = 0.5
```

```
) +
guides(
  fill = guide_colourbar("Variable\nvalue\n", ticks = FALSE, barheight = 10, barwidth = 0.5),
  color = "none"
) +
xlab("SHAP value\n(variable contribution to model prediction)") +
ylab("") +
ggtitle("Non-toxic")
```

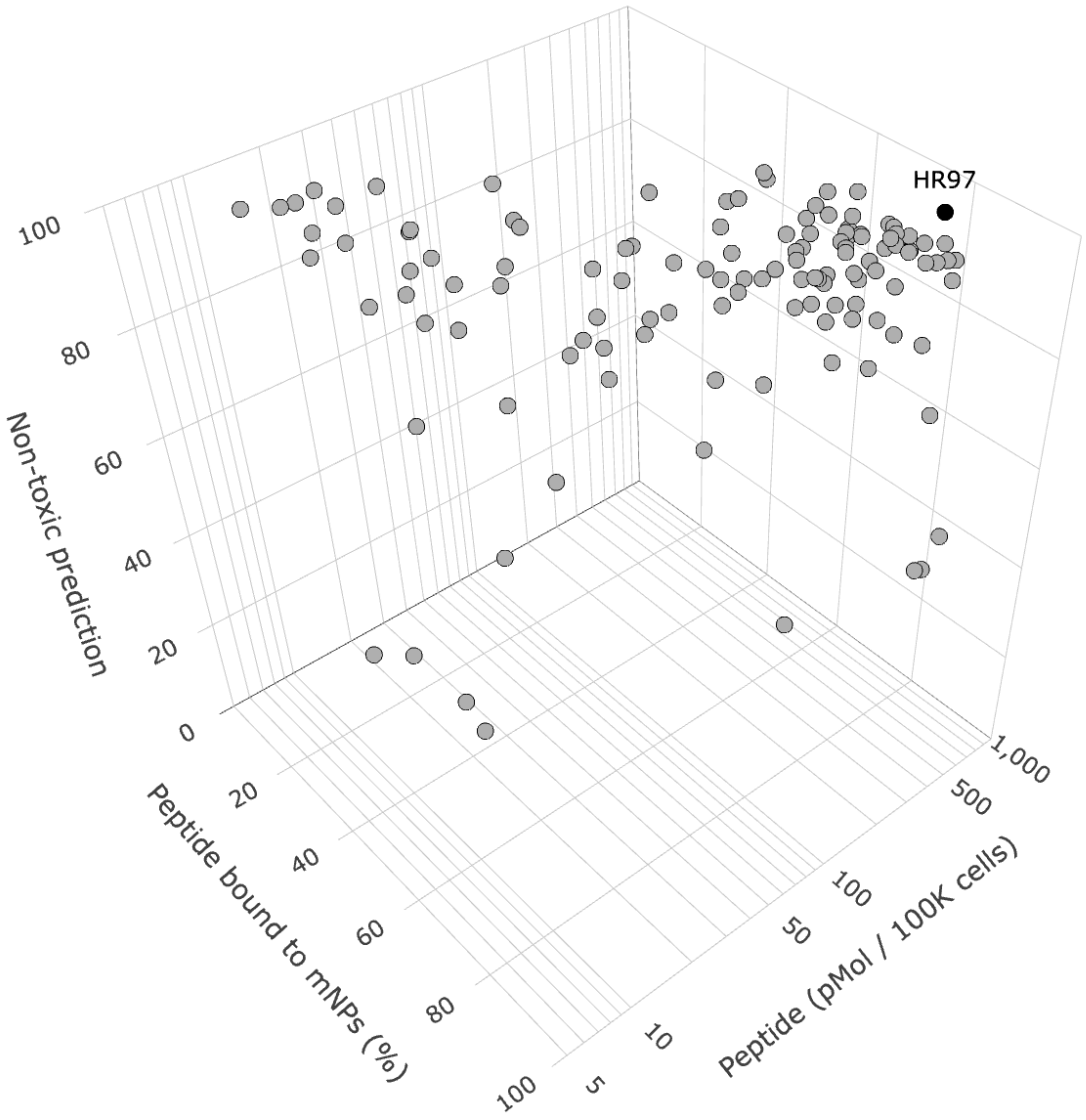


4.3 Multifunctional peptide selection

```
load(file = paste0("./rdata/tsne_res.RData"))
load(file = paste0("./rdata/var_reduct_mb_train_test_splits.RData"))
tsne_df$mb_predict <- rescale(tsne_df$mb_predict, to = c(0, 100), from = range(mb_data$log_intensity))
tsne_df$mb_predict[tsne_df$mb_predict > 100] <- 100
```

```
tsne_df_ <- tsne_df[tsne_df$source == "Validation control" | tsne_df$source == "Validation set", ]
tsne_df_$note[tsne_df_$note == "TAT"] <- NA
tsne_df_$color <- factor(sapply(tsne_df_$note, function(x) if (is.na(x)) 0 else 1))
p <- plot_ly(tsne_df_, x = ~in_vitro_mb_value, y = ~in_vitro_cp_value, z = ~non_tx_prob, text = ~note,
↪  color = ~color, colors = c("grey70", "black"), width = 700, height = 600) %>%
  add_trace(
    marker = list(size = 5, line = list(color = "black", width = 0.5)),
    type = "scatter3d", mode = "text+markers"
  ) %>%
  layout(
    scene = list(
      xaxis = list(
        title = "Peptide bound to mNPs (%)",
        range = c(0, 100)
      ),
      yaxis = list(
        title = "Peptide (pMol / 100K cells)",
        type = "log",
        ticktext = list(
          "", "5", "", "", "", "", "10",
          "", "", "", "50", "", "", "", "", "100",
          "", "", "", "500", "", "", "", "", "1,000"
        ),
        tickvals = list(
          4, 5, 6, 7, 8, 9, 10,
          20, 30, 40, 50, 60, 70, 80, 90, 100,
          200, 300, 400, 500, 600, 700, 800, 900, 1000
        ),
        range = c(0.63, 3)
      ),
      zaxis = list(
        title = "Non-toxic prediction",
```

```
        range = c(0, 100)
      )
    ),
    showlegend = FALSE, autosize = TRUE
  )
p
```

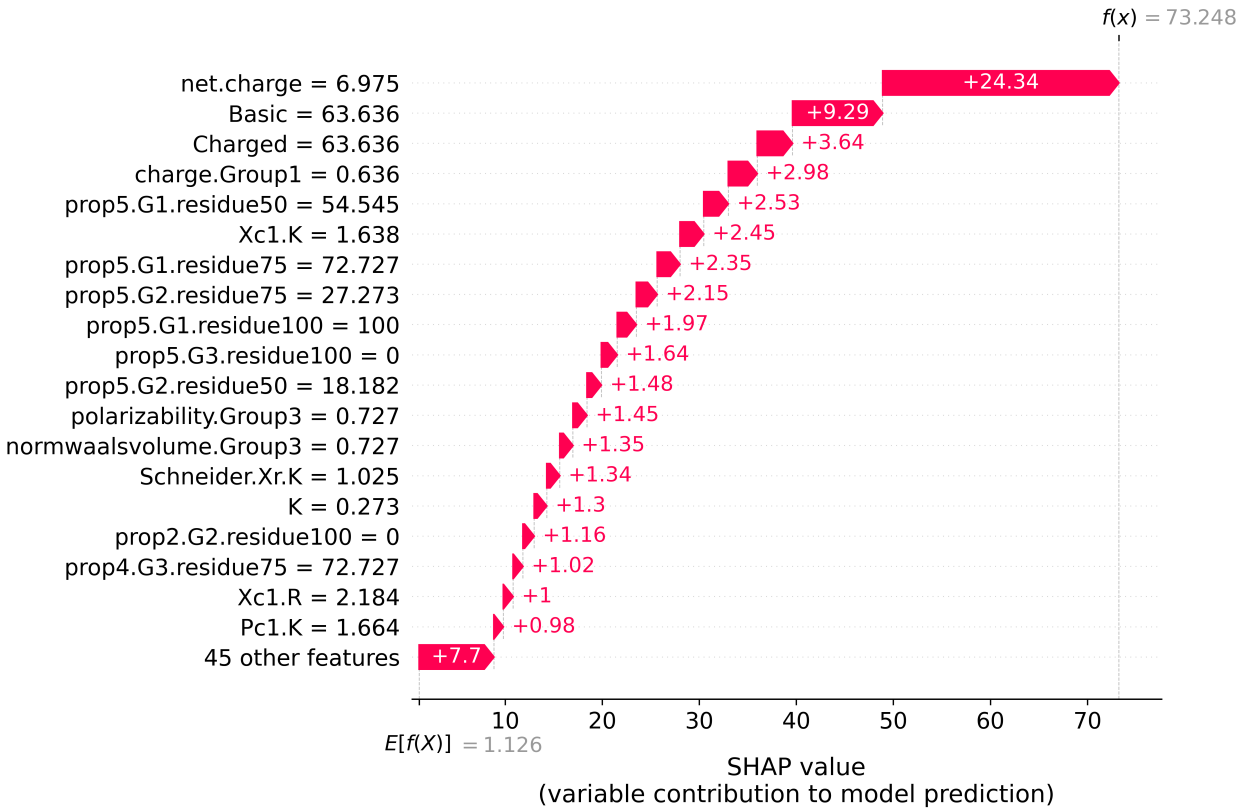


4.4 Explanation of HR97 predictions

4.4.1 Melanin binding

```
h2o.init(nthreads=-1)

dir_path = '/Users/renee/Downloads/melanin_binding'
model_name = 'superlearner_iter_3'
train = pd.read_csv(dir_path + '/whole_data_set/train_set.csv', index_col=0)
X_train = train.iloc[:, :-1]
X_test = pd.read_csv('./other_data/HR97_ml_input.csv', index_col=0).loc[:,X_train.columns]
model = h2o.load_model(dir_path + '/whole_data_set/' + model_name)
def model_predict_(data):
    data = pd.DataFrame(data, columns=X_train.columns)
    h2o_data = h2o.H2OFrame(data)
    res = model.predict(h2o_data)
    res = res.as_data_frame()
    return(res)

np.random.seed(1)
X_train_ = X_train.iloc[np.random.choice(X_train.shape[0], 100, replace=False), :]
explainer = shap.KernelExplainer(model_predict_, X_train_, link='identity')
shap_res = explainer.shap_values(X_test, nsamples=1000)
h2o.remove_all()

exp = shap.Explanation(shap_res * 100 / 3.89063, explainer.expected_value, data=X_test.iloc[0:1, :], \
                       feature_names=X_train.columns)

fig, ax = plt.subplots(figsize=(10, 7))
shap.plots.waterfall(exp[0], max_display=20, show=False)

yticklabels = plt.gca().get_yticklabels()
for i in range(int(len(yticklabels) / 2)):
    label = yticklabels[i]
    label_ = label.get_text().split(' = ')
    if len(label_) > 1:
        yticklabels[i].set_text(' = ' + label_[0])
    yticklabels[int(i + len(yticklabels) / 2)].set_text(' = '.join(label_[::-1]))

plt.gca().set_yticklabels(yticklabels)
fig.subplots_adjust(left=0.33, bottom=0.167, right=0.93, top=0.920)
```

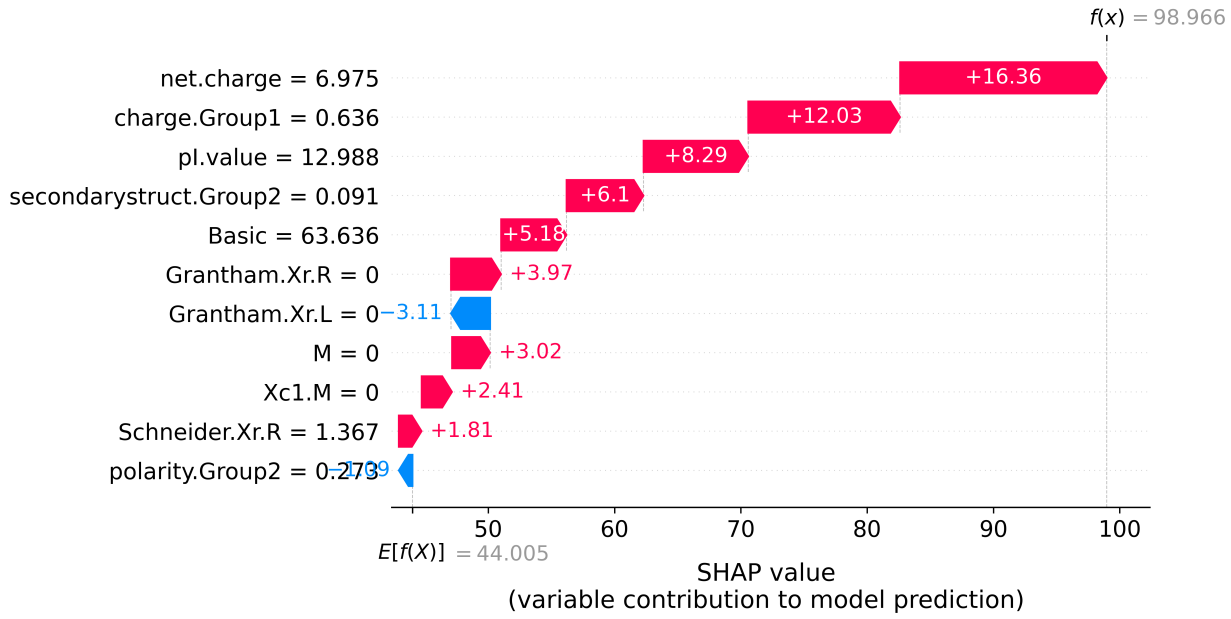
```
fig.set_figwidth(10)
fig.set_figheight(7)
plt.text(40, -4.8, 'SHAP value\n(variable contribution to model prediction)',
         horizontalalignment='center', fontsize=14)
plt.show()
```



4.4.2 Cell-penetration

```
h2o.init(nthreads=-1)

dir_path = '/Users/renee/Downloads/cell_penetration'
model_name = 'GBM_model_1669777615732_886421'
train = pd.read_csv(dir_path + '/whole_data_set/train_set.csv', index_col=0)
```

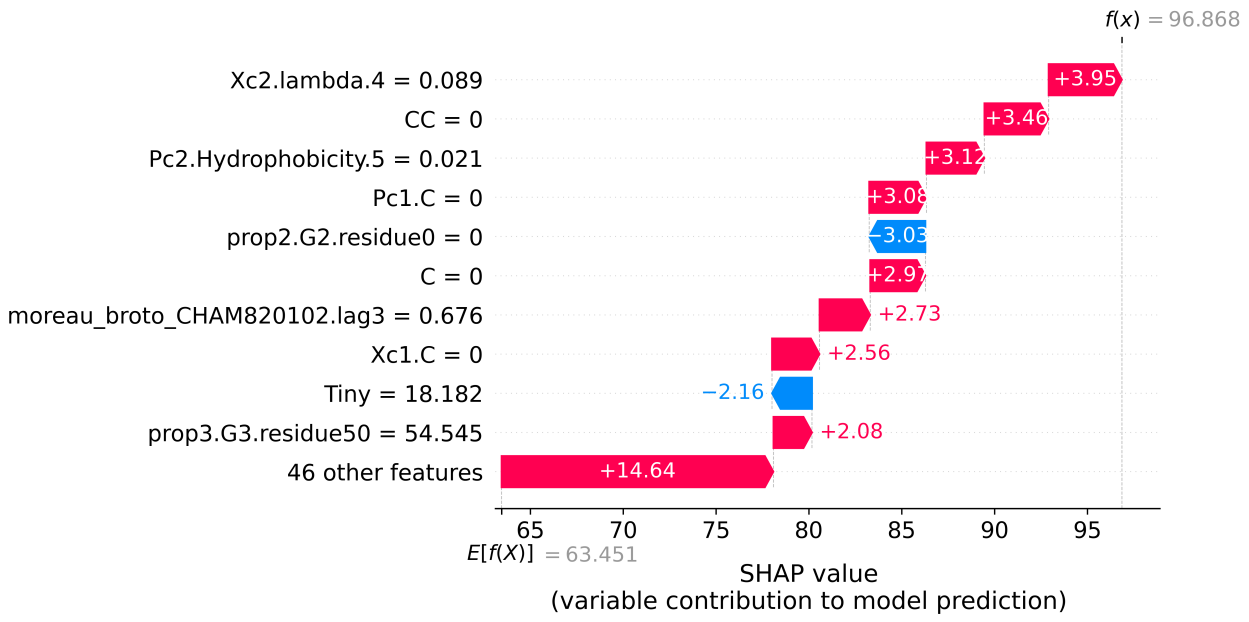
```
X_train = train.iloc[:, :-1]
X_test = pd.read_csv('./other_data/HR97_ml_input.csv', index_col=0).loc[:,X_train.columns]
model = h2o.load_model(dir_path + '/whole_data_set/' + model_name)
def model_predict_(data):
    data = pd.DataFrame(data, columns=X_train.columns)
    h2o_data = h2o.H2OFrame(data)
    res = model.predict(h2o_data)
    res = res.as_data_frame().iloc[:,2]
    return(res)

np.random.seed(1)
X_train_ = X_train.iloc[np.random.choice(X_train.shape[0], 100, replace=False), :]
explainer = shap.KernelExplainer(model_predict_, X_train_, link='identity')
shap_res = explainer.shap_values(X_test, nsamples=1000)
h2o.remove_all()

exp = shap.Explanation(shap_res * 100, explainer.expected_value * 100, data=X_test.iloc[0:1, :], \
                       feature_names=X_train.columns)

fig, ax = plt.subplots(figsize=(10, 5))
shap.plots.waterfall(exp[0], max_display=11, show=False)

yticklabels = plt.gca().get_yticklabels()
for i in range(int(len(yticklabels) / 2)):
    label = yticklabels[i]
    label_ = label.get_text().split(' = ')
    if len(label_) > 1:
        yticklabels[i].set_text(' = ' + label_[0])
    yticklabels[int(i + len(yticklabels) / 2)].set_text(' = '.join(label_[::-1]))

plt.gca().set_yticklabels(yticklabels)
fig.subplots_adjust(left=0.32, bottom=0.18, right=0.927, top=0.925)
fig.set_figwidth(10)
fig.set_figheight(5)
plt.text(72, -3.5, 'SHAP value\n(variable contribution to model prediction)',
         horizontalalignment='center', fontsize=14)
plt.show()
```



4.4.3 Toxicity

```
h2o.init(nthreads=-1)

dir_path = '/Users/renee/Downloads/toxicity'
model_name = 'superlearner_iter_3'
train = pd.read_csv(dir_path + '/whole_data_set/train_set.csv', index_col=0)
X_train = train.iloc[:, :-1]
X_test = pd.read_csv('./other_data/HR97_ml_input.csv', index_col=0).loc[:,X_train.columns]
model = h2o.load_model(dir_path + '/whole_data_set/' + model_name)
def model_predict_(data):
    data = pd.DataFrame(data, columns=X_train.columns)
    h2o_data = h2o.H2OFrame(data)
    res = model.predict(h2o_data)
    res = res.as_data_frame().iloc[:,2]
    return(res)

np.random.seed(1)
X_train_ = X_train.iloc[np.random.choice(X_train.shape[0], 100, replace=False), :]
explainer = shap.KernelExplainer(model_predict_, X_train_, link='identity')
shap_res = explainer.shap_values(X_test, nsamples=1000)
h2o.remove_all()
```

```
exp = shap.Explanation(shap_res * 100, explainer.expected_value * 100, data=X_test.iloc[0:1, :], \
                       feature_names=X_train.columns)

fig, ax = plt.subplots(figsize=(10, 5))
shap.plots.waterfall(exp[0], max_display=11, show=False)

yticklabels = plt.gca().get_yticklabels()
for i in range(int(len(yticklabels) / 2)):
    label = yticklabels[i]
    label_ = label.get_text().split(' = ')
    if len(label_) > 1:
        yticklabels[i].set_text(' = ' + label_[0])
    yticklabels[int(i + len(yticklabels) / 2)].set_text(' = '.join(label_[::-1]))

plt.gca().set_yticklabels(yticklabels)
fig.subplots_adjust(left=0.395, bottom=0.18, right=0.927, top=0.925)
fig.set_figwidth(10)
fig.set_figheight(5)
plt.text(80, -3.5, 'SHAP value\n(variable contribution to model prediction)',
         horizontalalignment='center', fontsize=14)
plt.show()
```



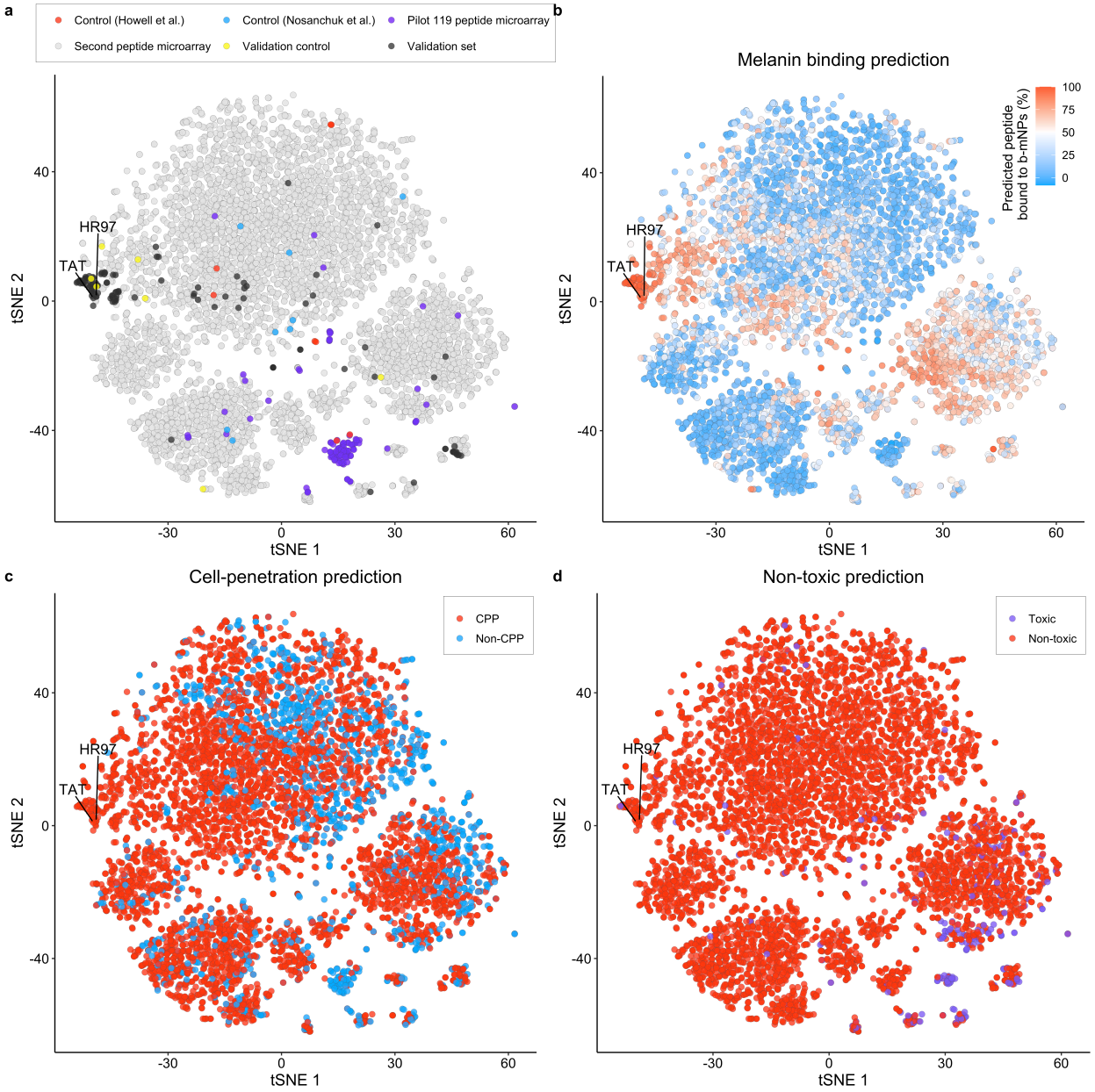
4.5 Peptide design space visualization

```
p1 <- ggplot(tsne_df, aes(x = `tSNE 1`, y = `tSNE 2`, fill = source)) +
  geom_point(color = "grey30", shape = 21, alpha = 0.8, size = 2.5, stroke = 0.1) +
  geom_text_repel(aes(label = note), color = "black", size = 4.5, nudge_x = -10, nudge_y = 10,
  ↪  point.padding = 0.1, force = 100) +
  scale_fill_manual(values = c("#FF380D", "#0DACFF", "#7134F7", "grey90", "#FEFA0A", "grey20")) +
  theme_bw() +
  theme(
    plot.margin = ggplot2::margin(5, 10, 5, 10),
    plot.background = element_rect(fill = "transparent", color = NA),
    panel.background = element_rect(fill = "transparent"),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    strip.background = element_blank(),
    panel.border = element_blank(),
    axis.line = element_line(color = "black"),
    plot.title = element_text(hjust = 0.5, size = 16),
    axis.title.x = element_text(hjust = 0.5, colour = "black", size = 14),
    axis.title.y = element_text(colour = "black", size = 14),
    axis.text.x = element_text(colour = "black", size = 12),
    axis.text.y = element_text(colour = "black", size = 12),
    legend.title = element_blank(),
    legend.text = element_text(colour = "black", size = 10),
    legend.title.align = 0.5,
    legend.position = "top",
    legend.background = element_rect(fill = "transparent", color = "black", linetype = "solid", size =
    ↪  0.1)
  ) +
  guides(fill = guide_legend(nrow = 2, byrow = TRUE))

p2 <- ggplot(tsne_df, aes(x = `tSNE 1`, y = `tSNE 2`, fill = mb_predict)) +
  geom_point(color = "grey30", shape = 21, alpha = 0.8, size = 2.5, stroke = 0.1) +
  geom_text_repel(aes(label = note), color = "black", size = 4.5, nudge_x = -10, nudge_y = 10,
  ↪  point.padding = 0.1, force = 100) +
  scale_fill_gradient2(low = "#0DACFF", mid = "white", high = "#FF380D", midpoint = 50) +
  theme_bw() +
  theme(
    plot.margin = ggplot2::margin(5, 10, 5, 10),
    plot.background = element_rect(fill = "transparent", color = NA),
```

```
    panel.background = element_rect(fill = "transparent"),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    strip.background = element_blank(),
    panel.border = element_blank(),
    axis.line = element_line(color = "black"),
    plot.title = element_text(hjust = 0.5, size = 16),
    axis.title.x = element_text(hjust = 0.5, colour = "black", size = 14),
    axis.title.y = element_text(colour = "black", size = 14),
    axis.text.x = element_text(colour = "black", size = 12),
    axis.text.y = element_text(colour = "black", size = 12),
    legend.title = element_text(colour = "black", size = 12, angle = 90),
    legend.text = element_text(colour = "black", size = 10),
    legend.title.align = 0.5,
    legend.position = c(0.91, 0.82),
    legend.background = element_blank()
  ) +
  guides(fill = guide_colorbar(
    title = "Predicted peptide \n bound to b-mNPs (%)",
    title.position = "left"
  )) +
  ggtitle("Melanin binding prediction")

p3 <- ggplot(tsne_df, aes(x = `tSNE 1`, y = `tSNE 2`, fill = cpp_predict)) +
  geom_point(color = "grey30", shape = 21, alpha = 0.8, size = 2.5, stroke = 0.1) +
  geom_text_repel(aes(label = note), color = "black", size = 4.5, nudge_x = -10, nudge_y = 10,
  ↪  point.padding = 0.1, force = 100) +
  scale_fill_manual(values = c("#FF380D", "#0DACFF")) +
  theme_bw() +
  theme(
    plot.margin = ggplot2::margin(5, 10, 5, 10),
    plot.background = element_rect(fill = "transparent", color = NA),
    panel.background = element_rect(fill = "transparent"),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    strip.background = element_blank(),
    panel.border = element_blank(),
    axis.line = element_line(color = "black"),
    plot.title = element_text(hjust = 0.5, size = 16),
    axis.title.x = element_text(hjust = 0.5, colour = "black", size = 14),
    axis.title.y = element_text(colour = "black", size = 14),
```

```
    axis.text.x = element_text(colour = "black", size = 12),
    axis.text.y = element_text(colour = "black", size = 12),
    legend.title = element_blank(),
    legend.text = element_text(colour = "black", size = 10),
    legend.title.align = 0.5,
    legend.position = c(0.90, 0.93),
    legend.background = element_rect(fill = "transparent", color = "black", linetype = "solid", size =
    ↪  0.1)
  ) +
  ggtitle("Cell-penetration prediction")

p4 <- ggplot(tsne_df, aes(x = `tSNE 1`, y = `tSNE 2`, fill = tx_predict)) +
  geom_point(color = "grey30", shape = 21, alpha = 0.8, size = 2.5, stroke = 0.1) +
  geom_text_repel(aes(label = note), color = "black", size = 4.5, nudge_x = -10, nudge_y = 10,
  ↪  point.padding = 0.1, force = 100) +
  scale_fill_manual(values = c("#7d5ef7", "#FF380D")) +
  theme_bw() +
  theme(
    plot.margin = ggplot2::margin(5, 5, 5, 5),
    plot.background = element_rect(fill = "transparent", color = NA),
    panel.background = element_rect(fill = "transparent"),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    strip.background = element_blank(),
    panel.border = element_blank(),
    axis.line = element_line(color = "black"),
    plot.title = element_text(hjust = 0.5, size = 16),
    axis.title.x = element_text(hjust = 0.5, colour = "black", size = 14),
    axis.title.y = element_text(colour = "black", size = 14),
    axis.text.x = element_text(colour = "black", size = 12),
    axis.text.y = element_text(colour = "black", size = 12),
    legend.title = element_blank(),
    legend.text = element_text(colour = "black", size = 10),
    legend.title.align = 0.5,
    legend.position = c(0.90, 0.93),
    legend.background = element_rect(fill = "transparent", color = "black", linetype = "solid", size =
    ↪  0.1)
  ) +
  ggtitle("Non-toxic prediction")
```

```
combined <- plot_grid(p1,
  plot_grid(NULL, p2, nrow = 2, rel_heights = c(0.08, 0.85)),
  p3, p4,
  nrow = 2, rel_heights = c(0.48, 0.45),
  labels = c("a", "b", "c", "d")
)
```




```
sessionInfo()
```

```
## R version 4.2.2 (2022-10-31)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Big Sur ... 10.16
##
```

```
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] ggrepel_0.9.1    plotly_4.10.0    cowplot_1.1.1    ggforce_0.3.4
##  [5] scales_1.2.1     reshape2_1.4.4   reticulate_1.25  ggbeeswarm_0.6.0
##  [9] ggpubr_0.4.0     ggplot2_3.3.6    h2o_3.38.0.2
##
## loaded via a namespace (and not attached):
##  [1] httr_1.4.4        tidyr_1.2.0       jsonlite_1.8.0    viridisLite_0.4.1
##  [5] carData_3.0-5     R.utils_2.12.0    here_1.0.1        assertthat_0.2.1
##  [9] vipor_0.4.5       yaml_2.3.5        pillar_1.8.1      backports_1.4.1
## [13] lattice_0.20-45   glue_1.6.2        digest_0.6.29     ggsignif_0.6.3
## [17] polyclip_1.10-0   colorspace_2.0-3  htmltools_0.5.3   Matrix_1.5-1
## [21] R.oo_1.25.0       plyr_1.8.7        pkgconfig_2.0.3   broom_1.0.0
## [25] bookdown_0.28     purrr_0.3.4       tweenr_2.0.1      tibble_3.1.8
## [29] styler_1.8.0      generics_0.1.3    farver_2.1.1      car_3.1-0
## [33] withr_2.5.0       lazyeval_0.2.2    cli_3.3.0         magrittr_2.0.3
## [37] evaluate_0.16     R.methodsS3_1.8.2 fansi_1.0.3       R.cache_0.16.0
## [41] MASS_7.3-58.1     rstatix_0.7.0     beeswarm_0.4.0    tools_4.2.2
## [45] data.table_1.14.2 lifecycle_1.0.1   stringr_1.4.1     munsell_0.5.0
## [49] compiler_4.2.2    rlang_1.0.4       grid_4.2.2        RCurl_1.98-1.8
## [53] rstudioapi_0.14   htmlwidgets_1.5.4 bitops_1.0-7      rmarkdown_2.16
## [57] gtable_0.3.0      codetools_0.2-18  abind_1.4-5       DBI_1.1.3
## [61] R6_2.5.1          knitr_1.40        dplyr_1.0.9       fastmap_1.1.0
## [65] utf8_1.2.2        rprojroot_2.0.3   stringi_1.7.8     Rcpp_1.0.9
## [69] vctrs_0.4.1       png_0.1-7         tidyselect_1.1.2  xfun_0.32
```

```
session_info.show()
```

```
## -----
## h2o                 3.38.0.2
## matplotlib          3.6.2
## numpy               1.23.5
## pandas              1.5.2
```

```
## session_info        1.0.0
## shap                0.41.0
## -----
## Python 3.8.15 | packaged by conda-forge | (default, Nov 22 2022, 09:04:40) [Clang 14.0.6 ]
## macOS-10.16-x86_64-i386-64bit
## -----
## Session information updated at 2023-03-12 23:18
```

Section 5

Adversarial computational control

```
library(h2o)
library(MLmetrics)
library(mltools)
library(scales)
library(enrichvs)
library(rlist)
library(stringr)
library(digest)
library(DT)
library(reshape2)
library(ggplot2)
library(ggforce)
library(reticulate)
use_condaenv("/Users/renee/Library/r-miniconda/envs/peptide_engineering/bin/python")
```

5.1 General code/functions

```
# h2o.init(nthreads=-1, max_mem_size='100G', port=54321)
h2o.init(nthreads = -1)
h2o.removeAll()
# DeepLearning Grid 1
deeplearning_params_1 <- list(
  activation = "RectifierWithDropout",
  epochs = 10000, # early stopping
  epsilon = c(1e-6, 1e-7, 1e-8, 1e-9),
  hidden = list(c(50), c(200), c(500)),
  hidden_dropout_ratios = list(c(0.1), c(0.2), c(0.3), c(0.4), c(0.5)),
  input_dropout_ratio = c(0, 0.05, 0.1, 0.15, 0.2),
  rho = c(0.9, 0.95, 0.99)
)
```

```
# DeepLearning Grid 2
deeplearning_params_2 <- list(
  activation = "RectifierWithDropout",
  epochs = 10000, # early stopping
  epsilon = c(1e-6, 1e-7, 1e-8, 1e-9),
  hidden = list(c(50, 50), c(200, 200), c(500, 500)),
  hidden_dropout_ratios = list(
    c(0.1, 0.1), c(0.2, 0.2), c(0.3, 0.3),
    c(0.4, 0.4), c(0.5, 0.5)
  ),
  input_dropout_ratio = c(0, 0.05, 0.1, 0.15, 0.2),
  rho = c(0.9, 0.95, 0.99)
)
# DeepLearning Grid 3
deeplearning_params_3 <- list(
  activation = "RectifierWithDropout",
  epochs = 10000, # early stopping
  epsilon = c(1e-6, 1e-7, 1e-8, 1e-9),
  hidden = list(c(50, 50, 50), c(200, 200, 200), c(500, 500, 500)),
  hidden_dropout_ratios = list(
    c(0.1, 0.1, 0.1), c(0.2, 0.2, 0.2),
    c(0.3, 0.3, 0.3), c(0.4, 0.4, 0.4),
    c(0.5, 0.5, 0.5)
  ),
  input_dropout_ratio = c(0, 0.05, 0.1, 0.15, 0.2),
  rho = c(0.9, 0.95, 0.99)
)
# GBM
gbm_params <- list(
  col_sample_rate = c(0.4, 0.7, 1.0),
  col_sample_rate_per_tree = c(0.4, 0.7, 1.0),
  learn_rate = 0.1,
  max_depth = c(3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17),
  min_rows = c(1, 5, 10, 15, 30, 100),
  min_split_improvement = c(1e-4, 1e-5),
  ntrees = 10000, # early stopping
  sample_rate = c(0.5, 0.6, 0.7, 0.8, 0.9, 1.0)
)
# XGBoost
xgboost_params <- list(
  booster = c("gbtree", "dart"),
```

```
  col_sample_rate = c(0.6, 0.8, 1.0),
  col_sample_rate_per_tree = c(0.7, 0.8, 0.9, 1.0),
  max_depth = c(5, 10, 15, 20),
  min_rows = c(0.01, 0.1, 1.0, 3.0, 5.0, 10.0, 15.0, 20.0),
  ntrees = 10000, # early stopping
  reg_alpha = c(0.001, 0.01, 0.1, 1, 10, 100),
  reg_lambda = c(0.001, 0.01, 0.1, 0.5, 1),
  sample_rate = c(0.6, 0.8, 1.0)
)
```

```
balanced_accuracy <- function(y_pred, y_true) {
  return(mean(c(Sensitivity(y_pred, y_true), Specificity(y_pred, y_true)), na.rm = TRUE))
}

sem <- function(x, na.rm = TRUE) sd(x, na.rm) / sqrt(length(na.omit(x)))

statistical_testing <- function(data, metric_vec, decreasing_vec, p_value_threshold = 0.05,
↪  num_entries = 10,
                                output_path = NULL) {
  for (i in 1:length(metric_vec)) {
    metric <- metric_vec[i]
    ranked_models <- rownames(data[order(data[, paste(metric, "mean")], decreasing =
↪  decreasing_vec[i]), ])
    pval <- c()
    for (j in 2:length(ranked_models)) {
      if (sum(is.na(data[ranked_models[j], paste(metric, "CV", 1:10)]))) {
        pval <- c(pval, NA)
      } else {
        pval <- c(pval, wilcox.test(as.numeric(data[ranked_models[1], paste(metric, "CV", 1:10)]),
          as.numeric(data[ranked_models[j], paste(metric, "CV", 1:10)]),
          exact = FALSE
        )$p.value)
      }
    }
    adj_pval <- c(NA, p.adjust(pval, method = "BH", n = length(pval)))
    df <- data.frame(adj_pval)
    rownames(df) <- ranked_models
    colnames(df) <- paste(metric, "adj. <i>p</i> value")
    data <- merge(data, df, by = "row.names", all = TRUE)
```

```
    rownames(data) <- data$Row.names
    data$Row.names <- NULL
  }
  for (i in 1:length(metric_vec)) {
    if (decreasing_vec[i]) {
      data[paste(metric_vec[i], "rank")] <- rank(-data[, paste(metric_vec[i], "mean")], ties.method =
↪  "average")
    } else {
      data[paste(metric_vec[i], "rank")] <- rank(data[, paste(metric_vec[i], "mean")], ties.method =
↪  "average")
    }
  }
  data["Rank sum"] <- rowSums(data[(ncol(data) - length(metric_vec) + 1):ncol(data)])
  data <- data[order(data$`Rank sum`), ]
  competitive <- rowSums(data[paste(metric_vec, "adj. <i>p</i> value")] < p_value_threshold) == 0
  competitive[is.na(competitive)] <- TRUE
  data <- data[competitive, ]
  if (!is.null(output_path)) {
    data[c(
      paste(metric_vec, "adj. <i>p</i> value"), paste(metric_vec, "rank"), "Rank sum",
      melt(sapply(metric_vec, function(x) paste(x, c("mean", "s.e.m."))))$value
    )] %>%
      round(digits = 4) %>%
      write.csv(., file = output_path)
  }
  data[c(paste(metric_vec, "adj. <i>p</i> value"), paste(metric_vec, "rank"), "Rank sum")] %>%
    round(digits = 4) %>%
    datatable(options = list(pageLength = num_entries), escape = FALSE)
}
```

5.2 Melanin binding (regression)

```
load(file = "./rdata/var_reduct_mb_train_test_splits.RData")
set.seed(32)

# Shuffle the response variable of cross-validation training sets
for (i in 1:10) {
  response <- outer_splits[[i]]$train$log_intensity
```

```
  response <- sample(response)
  outer_splits[[i]]$train$log_intensity <- response
}

# Shuffle the response variable of the whole data set
response <- mb_data$log_intensity
response <- sample(response)
mb_data$log_intensity <- response

save(mb_data, outer_splits,
  file = "./rdata/var_reduct_mb_train_test_splits_y_shuffled.RData"
)
```

5.2.1 Model training

```
train_mb_models <- function(train_set, exp_dir, prefix, nfolds = 10, grid_seed = 1) {
  tmp <- as.h2o(train_set, destination_frame = prefix)
  y <- "log_intensity"
  x <- setdiff(names(tmp), y)
  res <- as.data.frame(tmp$log_intensity)
  # --------------------
  # base model training
  # --------------------
  cat("Deep learning grid 1\n")
  deeplearning_1 <- h2o.grid(
    algorithm = "deeplearning", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
    keep_cross_validation_predictions = TRUE, hyper_params = deeplearning_params_1,
    stopping_rounds = 3, keep_cross_validation_models = FALSE,
    search_criteria = list(
      strategy = "RandomDiscrete",
      max_models = 100, seed = grid_seed
    ),
    fold_assignment = "Modulo", parallelism = 0
  )
  for (model_id in deeplearning_1@model_ids) {
    tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
  }
  cat("Deep learning grid 2\n")
  deeplearning_2 <- h2o.grid(
```

```
  algorithm = "deeplearning", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, hyper_params = deeplearning_params_2,
  stopping_rounds = 3, keep_cross_validation_models = FALSE,
  search_criteria = list(
    strategy = "RandomDiscrete",
    max_models = 100, seed = grid_seed
  ),
  fold_assignment = "Modulo", parallelism = 0
)
for (model_id in deeplearning_2@model_ids) {
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
}
cat("Deep learning grid 3\n")
deeplearning_3 <- h2o.grid(
  algorithm = "deeplearning", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, hyper_params = deeplearning_params_3,
  stopping_rounds = 3, keep_cross_validation_models = FALSE,
  search_criteria = list(
    strategy = "RandomDiscrete",
    max_models = 100, seed = grid_seed
  ),
  fold_assignment = "Modulo", parallelism = 0
)
for (model_id in deeplearning_3@model_ids) {
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
}
cat("GBM grid\n")
gbm <- h2o.grid(
  algorithm = "gbm", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, hyper_params = gbm_params, stopping_rounds = 3,
  search_criteria = list(strategy = "RandomDiscrete", max_models = 300, seed = grid_seed),
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo", parallelism = 0
)
for (model_id in gbm@model_ids) {
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
}
cat("GBM 5 default models\n")
gbm_1 <- h2o.gbm(
  model_id = "GBM_1", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8, learn_rate = 0.1,
```

```
  max_depth = 6, min_rows = 1, min_split_improvement = 1e-5, ntrees = 10000, sample_rate = 0.8,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
gbm_2 <- h2o.gbm(
  model_id = "GBM_2", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8, learn_rate = 0.1,
  max_depth = 7, min_rows = 10, min_split_improvement = 1e-5, ntrees = 10000, sample_rate = 0.8,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
gbm_3 <- h2o.gbm(
  model_id = "GBM_3", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8, learn_rate = 0.1,
  max_depth = 8, min_rows = 10, min_split_improvement = 1e-5, ntrees = 10000, sample_rate = 0.8,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
gbm_4 <- h2o.gbm(
  model_id = "GBM_4", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8, learn_rate = 0.1,
  max_depth = 10, min_rows = 10, min_split_improvement = 1e-5, ntrees = 10000, sample_rate = 0.8,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
gbm_5 <- h2o.gbm(
  model_id = "GBM_5", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8, learn_rate = 0.1,
  max_depth = 15, min_rows = 100, min_split_improvement = 1e-5, ntrees = 10000, sample_rate = 0.8,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
for (model_id in paste0("GBM_", 1:5)) {
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
}
cat("XGBoost grid\n")
xgboost <- h2o.grid(
  algorithm = "xgboost", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, hyper_params = xgboost_params, stopping_rounds = 3,
  search_criteria = list(strategy = "RandomDiscrete", max_models = 300, seed = grid_seed),
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo", parallelism = 0
)
```

```
for (model_id in xgboost@model_ids) {
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
}
cat("XGBoost 3 default models\n")
xgboost_1 <- h2o.xgboost(
  model_id = "XGBoost_1", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  booster = "gbtree", col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8,
  max_depth = 10, min_rows = 5, ntrees = 10000, reg_alpha = 0, reg_lambda = 1,
  sample_rate = 0.6, keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
xgboost_2 <- h2o.xgboost(
  model_id = "XGBoost_2", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  booster = "gbtree", col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8,
  max_depth = 20, min_rows = 10, ntrees = 10000, reg_alpha = 0, reg_lambda = 1,
  sample_rate = 0.6, keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
xgboost_3 <- h2o.xgboost(
  model_id = "XGBoost_3", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  booster = "gbtree", col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8,
  max_depth = 5, min_rows = 3, ntrees = 10000, reg_alpha = 0, reg_lambda = 1,
  sample_rate = 0.8, keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
for (model_id in paste0("XGBoost_", 1:3)) {
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
}
cat("GLM\n")
glm <- h2o.glm(
  model_id = "GLM", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, alpha = c(0.0, 0.2, 0.4, 0.6, 0.8, 1.0),
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
tmp_path <- h2o.saveModel(h2o.getModel("GLM"), path = exp_dir, force = TRUE)
cat("DRF\n")
drf <- h2o.randomForest(
  model_id = "DRF", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, ntrees = 10000,
  score_tree_interval = 5, stopping_rounds = 3,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
```

```
)
tmp_path <- h2o.saveModel(h2o.getModel("DRF"), path = exp_dir, force = TRUE)
cat("XRT\n")
xrt <- h2o.randomForest(
  model_id = "XRT", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, ntrees = 10000, histogram_type = "Random",
  score_tree_interval = 5, stopping_rounds = 3,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
tmp_path <- h2o.saveModel(h2o.getModel("XRT"), path = exp_dir, force = TRUE)
# ----------------------
# get holdout predictions
# ----------------------
base_models <- as.list(c(
  unlist(deeplearning_1@model_ids),
  unlist(deeplearning_2@model_ids),
  unlist(deeplearning_3@model_ids),
  unlist(gbm@model_ids), paste0("GBM_", 1:5),
  unlist(xgboost@model_ids), paste0("XGBoost_", 1:3),
  "GLM",
  "DRF",
  "XRT"
))
for (model_id in base_models) {
  res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_", model_id)),
    col.names = model_id
  ))
}
# ----------------------
# super learner training
# ----------------------
sl_iter <- 0
cat(paste0("Super learner iteration ", sl_iter, " (", length(base_models), " models)\n"))
sl <- h2o.stackedEnsemble(
  x = x, y = y, model_id = paste0("superlearner_iter_", sl_iter),
  training_frame = tmp, seed = 1,
  base_models = base_models,
  metalearner_algorithm = "glm",
  metalearner_nfolds = nfolds,
  keep_levelone_frame = TRUE,
  metalearner_params = list(standardize = TRUE, keep_cross_validation_predictions = TRUE)
```

```
  )
  tmp_path <- h2o.saveModel(h2o.getModel(paste0("superlearner_iter_", sl_iter)), path = exp_dir, force
↪  = TRUE)
  model_id <- paste0("metalearner_glm_superlearner_iter_", sl_iter)
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
  res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_", model_id)),
    col.names = paste0(model_id, "_", length(base_models), "_models")
  ))
  # ---------------------------------
  # super learner base model reduction
  # ---------------------------------
  while (TRUE) {
    meta <- h2o.getModel(paste0("metalearner_glm_superlearner_iter_", sl_iter, "_cv_1"))
    names <- meta@model$coefficients_table[, "names"]
    coeffs <- (meta@model$coefficients_table[, "standardized_coefficients"] > 0)
    for (j in 2:nfolds) {
      meta <- h2o.getModel(paste0("metalearner_glm_superlearner_iter_", sl_iter, "_cv_", j))
      names <- meta@model$coefficients_table[, "names"]
      coeffs <- coeffs + (meta@model$coefficients_table[, "standardized_coefficients"] > 0)
    }
    base_models_ <- as.list(names[coeffs >= ceiling(nfolds / 2) & names != "Intercept"])
    if (length(base_models_) == 0) {
      cat("No base models passing the threshold\n\n")
      break
    }
    if (sum(base_models %in% base_models_) == length(base_models)) {
      cat("No further reduction of base models\n\n")
      break
    }
    sl_iter <- sl_iter + 1
    base_models <- base_models_
    cat(paste0("Super learner iteration ", sl_iter, " (", length(base_models), " models)\n"))
    sl <- h2o.stackedEnsemble(
      x = x, y = y, model_id = paste0("superlearner_iter_", sl_iter),
      training_frame = tmp, seed = 1,
      base_models = base_models,
      metalearner_algorithm = "glm",
      metalearner_nfolds = nfolds,
      keep_levelone_frame = TRUE,
      metalearner_params = list(standardize = TRUE, keep_cross_validation_predictions = TRUE)
    )
```

```
    tmp_path <- h2o.saveModel(h2o.getModel(paste0("superlearner_iter_", sl_iter)), path = exp_dir,
↪   force = TRUE)
    model_id <- paste0("metalearner_glm_superlearner_iter_", sl_iter)
    tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
    res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_", model_id)),
      col.names = paste0(model_id, "_", length(base_models), "_models")
    ))
  }
  # ----------------------------------------
  # super learner for homogeneous base models
  # ----------------------------------------
  # DeepLearning
  base_models <- as.list(c(
    unlist(deeplearning_1@model_ids),
    unlist(deeplearning_2@model_ids),
    unlist(deeplearning_3@model_ids)
  ))
  cat(paste0("Super learner deep learning (", length(base_models), " models)\n"))
  sl <- h2o.stackedEnsemble(
    x = x, y = y, model_id = "superlearner_deeplearning",
    training_frame = tmp, seed = 1,
    base_models = base_models,
    metalearner_algorithm = "glm",
    metalearner_nfolds = nfolds,
    keep_levelone_frame = TRUE,
    metalearner_params = list(standardize = TRUE, keep_cross_validation_predictions = TRUE)
  )
  tmp_path <- h2o.saveModel(h2o.getModel("superlearner_deeplearning"), path = exp_dir, force = TRUE)
  model_id <- "metalearner_glm_superlearner_deeplearning"
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
  res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_", model_id)),
    col.names = paste0(model_id, "_", length(base_models), "_models")
  ))
  # GBM
  base_models <- as.list(c(unlist(gbm@model_ids), paste0("GBM_", 1:5)))
  cat(paste0("Super learner GBM (", length(base_models), " models)\n"))
  sl <- h2o.stackedEnsemble(
    x = x, y = y, model_id = "superlearner_gbm",
    training_frame = tmp, seed = 1,
    base_models = base_models,
    metalearner_algorithm = "glm",
```

```
    metalearner_nfolds = nfolds,
    keep_levelone_frame = TRUE,
    metalearner_params = list(standardize = TRUE, keep_cross_validation_predictions = TRUE)
  )
  tmp_path <- h2o.saveModel(h2o.getModel("superlearner_gbm"), path = exp_dir, force = TRUE)
  model_id <- "metalearner_glm_superlearner_gbm"
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
  res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_", model_id)),
    col.names = paste0(model_id, "_", length(base_models), "_models")
  ))
  # XGBoost
  base_models <- as.list(c(unlist(xgboost@model_ids), paste0("XGBoost_", 1:3)))
  cat(paste0("Super learner XGBoost (", length(base_models), " models)\n"))
  sl <- h2o.stackedEnsemble(
    x = x, y = y, model_id = "superlearner_xgboost",
    training_frame = tmp, seed = 1,
    base_models = base_models,
    metalearner_algorithm = "glm",
    metalearner_nfolds = nfolds,
    keep_levelone_frame = TRUE,
    metalearner_params = list(standardize = TRUE, keep_cross_validation_predictions = TRUE)
  )
  tmp_path <- h2o.saveModel(h2o.getModel("superlearner_xgboost"), path = exp_dir, force = TRUE)
  model_id <- "metalearner_glm_superlearner_xgboost"
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
  res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_", model_id)),
    col.names = paste0(model_id, "_", length(base_models), "_models")
  ))
  write.csv(res, file = paste0(exp_dir, "/cv_holdout_predictions.csv"), row.names = FALSE)
  cat("\n\n")
  h2o.removeAll()
}
```

5.2.2 Inner cross-validation

```
load(file = "./rdata/var_reduct_mb_train_test_splits_y_shuffled.RData")
for (i in 1:10) {
  cat(paste0("Outer training set ", i, "\n"))
  prefix <- paste0("outer_", i)
```

```
  exp_dir <- paste0("/Users/renee/Downloads/melanin_binding_adversarial/", prefix)
  dir.create(exp_dir)
  train_mb_models(train_set = outer_splits[[i]][["train"]], exp_dir = exp_dir, prefix = prefix)
}
```

```
# Keep track of grid models
dl_grid <- list()
gbm_grid <- list()
xgboost_grid <- list()
dl_grid_params <- list()
gbm_grid_params <- list()
xgboost_grid_params <- list()
for (i in 1:10) {
  cat(paste0("Outer training set ", i, "\n"))
  prefix <- paste0("outer_", i)
  dir <- paste0("/Users/renee/Downloads/melanin_binding_adversarial/", prefix)
  files <- list.files(dir)
  # Deep learning
  dl <- files[str_detect(files, "DeepLearning_model")]
  for (m in dl) {
    model <- h2o.loadModel(paste0(dir, "/", m))
    hs <- sha1(paste(c(
      model@allparameters$epsilon,
      model@allparameters$hidden,
      model@allparameters$hidden_dropout_ratios,
      model@allparameters$input_dropout_ratio,
      model@allparameters$rho
    ), collapse = " "))
    if (hs %in% names(dl_grid)) {
      dl_grid[[hs]] <- c(dl_grid[[hs]], m)
    } else {
      dl_grid[[hs]] <- c(m)
      dl_grid_params <- list.append(
        dl_grid_params,
        c(
          "epsilon" = model@allparameters$epsilon,
          "hidden" = paste0("[", paste(model@allparameters$hidden, collapse = ","), "]"),
          "hidden_dropout_ratios" = paste0(
            "[",
```

```
          paste(model@allparameters$hidden_dropout_ratios,
            collapse = ","
          ), "]"
        ),
        "input_dropout_ratio" = model@allparameters$input_dropout_ratio,
        "rho" = model@allparameters$rho
      )
    )
  }
}
h2o.removeAll()
# GBM
gbm <- files[str_detect(files, "GBM_model")]
for (m in gbm) {
  model <- h2o.loadModel(paste0(dir, "/", m))
  hs <- sha1(paste(c(
    model@allparameters$col_sample_rate,
    model@allparameters$col_sample_rate_per_tree,
    model@allparameters$max_depth,
    model@allparameters$min_rows,
    model@allparameters$min_split_improvement,
    model@allparameters$sample_rate
  ), collapse = " "))
  if (hs %in% names(gbm_grid)) {
    gbm_grid[[hs]] <- c(gbm_grid[[hs]], m)
  } else {
    gbm_grid[[hs]] <- c(m)
    gbm_grid_params <- list.append(
      gbm_grid_params,
      c(
        "col_sample_rate" = model@allparameters$col_sample_rate,
        "col_sample_rate_per_tree" = model@allparameters$col_sample_rate_per_tree,
        "max_depth" = model@allparameters$max_depth,
        "min_rows" = model@allparameters$min_rows,
        "min_split_improvement" = model@allparameters$min_split_improvement,
        "sample_rate" = model@allparameters$sample_rate
      )
    )
  }
}
h2o.removeAll()
```

```
  # XGBoost
  xgboost <- files[str_detect(files, "XGBoost_model")]
  for (m in xgboost) {
    model <- h2o.loadModel(paste0(dir, "/", m))
    hs <- sha1(paste(c(
      model@allparameters$booster,
      model@allparameters$col_sample_rate,
      model@allparameters$col_sample_rate_per_tree,
      model@allparameters$max_depth,
      model@allparameters$min_rows,
      model@allparameters$reg_alpha,
      model@allparameters$reg_lambda,
      model@allparameters$sample_rate
    ), collapse = " "))
    if (hs %in% names(xgboost_grid)) {
      xgboost_grid[[hs]] <- c(xgboost_grid[[hs]], m)
    } else {
      xgboost_grid[[hs]] <- c(m)
      xgboost_grid_params <- list.append(
        xgboost_grid_params,
        c(
          "booster" = model@allparameters$booster,
          "col_sample_rate" = model@allparameters$col_sample_rate,
          "col_sample_rate_per_tree" = model@allparameters$col_sample_rate_per_tree,
          "max_depth" = model@allparameters$max_depth,
          "min_rows" = model@allparameters$min_rows,
          "reg_alpha" = model@allparameters$reg_alpha,
          "reg_lambda" = model@allparameters$reg_lambda,
          "sample_rate" = model@allparameters$sample_rate
        )
      )
    }
  }
  h2o.removeAll()
}

dl_grid_params <- as.data.frame(t(data.frame(dl_grid_params)))
rownames(dl_grid_params) <- paste("Neural network grid model", 1:nrow(dl_grid_params))

gbm_grid_params <- as.data.frame(t(data.frame(gbm_grid_params)))
rownames(gbm_grid_params) <- paste("GBM grid model", 1:nrow(gbm_grid_params))
```

```
xgboost_grid_params <- as.data.frame(t(data.frame(xgboost_grid_params)))
rownames(xgboost_grid_params) <- paste("XGBoost grid model", 1:nrow(xgboost_grid_params))

write.csv(dl_grid_params, "./other_data/melanin_binding_adversarial/neural_network_grid_params.csv")
write.csv(gbm_grid_params, "./other_data/melanin_binding_adversarial/gbm_grid_params.csv")
write.csv(xgboost_grid_params, "./other_data/melanin_binding_adversarial/xgboost_grid_params.csv")

save(dl_grid, gbm_grid, xgboost_grid, file = "./rdata/model_training_grid_models_mb_ad.RData")
```

5.2.3 Model evaluation

```
model_evaluation <- function(holdout_pred, fold_asign, grid_name, grid_meta = NULL) {
  load(file = "./rdata/var_reduct_mb_train_test_splits_y_shuffled.RData")
  res_ <- list()
  model_num <- c()
  if (startsWith(grid_name, "Super learner")) {
    if (grid_name == "Super learner all models") {
      m <- colnames(holdout_pred)[startsWith(colnames(holdout_pred),
↪  "metalearner_glm_superlearner_iter_0")]
    } else if (grid_name == "Super learner final") {
      sl_models <- colnames(holdout_pred)[startsWith(colnames(holdout_pred),
↪  "metalearner_glm_superlearner_iter")]
      m <- sl_models[length(sl_models)]
    } else if (grid_name == "Super learner neural network") {
      m <- colnames(holdout_pred)[startsWith(colnames(holdout_pred),
↪  "metalearner_glm_superlearner_deeplearning")]
    } else if (grid_name == "Super learner GBM") {
      m <- colnames(holdout_pred)[startsWith(colnames(holdout_pred),
↪  "metalearner_glm_superlearner_gbm")]
    } else if (grid_name == "Super learner XGBoost") {
      m <- colnames(holdout_pred)[startsWith(colnames(holdout_pred),
↪  "metalearner_glm_superlearner_xgboost")]
    }
    mae <- c()
    rmse <- c()
    R2 <- c()
    for (k in 1:10) {
      y_true <- holdout_pred[fold_assign == k, "log_intensity"]
```

```
    y_pred <- holdout_pred[fold_assign == k, m]
    mae <- c(mae, MAE(y_pred = y_pred, y_true = y_true))
    rmse <- c(rmse, RMSE(y_pred = y_pred, y_true = y_true))
    R2 <- c(R2, R2_Score(y_pred = y_pred, y_true = y_true))
  }
  # Re-scale to 0-100
  mae <- rescale(mae, to = c(0, 100), from = range(mb_data$log_intensity))
  rmse <- rescale(rmse, to = c(0, 100), from = range(mb_data$log_intensity))
  res_ <- list.append(res_, c(
    mean(mae, na.rm = TRUE), sem(mae),
    mean(rmse, na.rm = TRUE), sem(rmse),
    mean(R2, na.rm = TRUE), sem(R2),
    mae, rmse, R2
  ))
} else {
  for (j in 1:length(grid_meta)) {
    g <- grid_meta[[j]]
    m <- intersect(colnames(holdout_pred), g)
    if (length(m) == 1) {
      model_num <- c(model_num, j)
      mae <- c()
      rmse <- c()
      R2 <- c()
      for (k in 1:10) {
        y_true <- holdout_pred[fold_assign == k, "log_intensity"]
        y_pred <- holdout_pred[fold_assign == k, m]
        mae <- c(mae, MAE(y_pred = y_pred, y_true = y_true))
        rmse <- c(rmse, RMSE(y_pred = y_pred, y_true = y_true))
        R2 <- c(R2, R2_Score(y_pred = y_pred, y_true = y_true))
      }
      # Re-scale to 0-100
      mae <- rescale(mae, to = c(0, 100), from = range(mb_data$log_intensity))
      rmse <- rescale(rmse, to = c(0, 100), from = range(mb_data$log_intensity))
      res_ <- list.append(res_, c(
        mean(mae, na.rm = TRUE), sem(mae),
        mean(rmse, na.rm = TRUE), sem(rmse),
        mean(R2, na.rm = TRUE), sem(R2),
        mae, rmse, R2
      ))
    }
  }
```

```
  }
  res <- as.data.frame(t(data.frame(res_)))
  colnames(res) <- c(
    "Norm. MAE mean", "Norm. MAE s.e.m.", "Norm. RMSE mean", "Norm. RMSE s.e.m.",
    "R^2 mean", "R^2 s.e.m.",
    paste("Norm. MAE CV", 1:10), paste("Norm. RMSE CV", 1:10), paste("R^2 CV", 1:10)
  )
  if (nrow(res) == 1) {
    rownames(res) <- grid_name
  } else {
    rownames(res) <- paste(grid_name, model_num)
  }
  return(res)
}
```

5.2.4 Inner loop model selection

```
load(file = "./rdata/model_training_grid_models_mb_ad.RData")

for (i in 1:10) {
  holdout_pred <- read.csv(paste0("./other_data/melanin_binding_adversarial/outer_", i,
↪  "/cv_holdout_predictions.csv"))
  fold_assign <- rep(1:10, ceiling(nrow(holdout_pred) / 10))[1:nrow(holdout_pred)]
  data_ <- list()
  # Deep learning grid models
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
    grid_name = "Neural network grid model",
    grid_meta = dl_grid
  ))
  # GBM grid models
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
    grid_name = "GBM grid model",
    grid_meta = gbm_grid
  ))
  # GBM default models
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
    grid_name = "GBM model",
    grid_meta = as.list(paste0("GBM_", 1:5))
  ))
```

```
  # XGBoost grid models
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
    grid_name = "XGBoost grid model",
    grid_meta = xgboost_grid
  ))
  # XGBoost default models
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
    grid_name = "XGBoost model",
    grid_meta = as.list(paste0("XGBoost_", 1:3))
  ))
  # GLM
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "GLM model",
↪  grid_meta = list("GLM")))
  # DRF
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "DRF model",
↪  grid_meta = list("DRF")))
  # XRT
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "XRT model",
↪  grid_meta = list("XRT")))
  # Super learner all
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪  all models"))
  # Super learner final
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪  final"))
  # Super learner deep learning
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪  neural network"))
  # Super learner GBM
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪  GBM"))
  # Super learner XGBoost
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪  XGBoost"))

  data <- do.call(rbind, data_)
  write.csv(data, paste0("./other_data/melanin_binding_adversarial/outer_", i, "/cv_res.csv"))
}
```

5.2.4.1 CV 1

Show 1 entries Search:

	Norm. MAE adj. *p* value	Norm. RMSE adj. *p* value	R^2 adj. *p* value	Norm. MAE rank	Norm. RMSE rank	R^2 rank	Rank sum
Neural network grid model 113	0.1888			108	1	1	110

Showing 1 to 1 of 296 entries Previous 1 2 3 4 5 … 296 Next

5.2.4.2 CV 2

Show 1 entries Search:

	Norm. MAE adj. *p* value	Norm. RMSE adj. *p* value	R^2 adj. *p* value	Norm. MAE rank	Norm. RMSE rank	R^2 rank	Rank sum
Neural network grid model 187	0.5018	1	0.8102	137	5	4	146

Showing 1 to 1 of 277 entries Previous 1 2 3 4 5 … 277 Next

5.2.4.3 CV 3

Show 1 entries Search:

	Norm. MAE adj. *p* value	Norm. RMSE adj. *p* value	R^2 adj. *p* value	Norm. MAE rank	Norm. RMSE rank	R^2 rank	Rank sum
Neural network grid model 145	0.0039			144	1	1	146

Showing 1 to 1 of 2 entries Previous 1 2 Next

5.2.4.4 CV 4

Show 1 entries Search:

	Norm. MAE adj. *p* value	Norm. RMSE adj. *p* value	R^2 adj. *p* value	Norm. MAE rank	Norm. RMSE rank	R^2 rank	Rank sum
Neural network grid model 215	0.159	0.8037	0.5733	44	9	9	62

Showing 1 to 1 of 118 entries Previous 1 2 3 4 5 … 118 Next

5.2.4.5 CV 5

Show 1 entries Search:

	Norm. MAE adj. *p* value	Norm. RMSE adj. *p* value	R^2 adj. *p* value	Norm. MAE rank	Norm. RMSE rank	R^2 rank	Rank sum
Super learner final	0.8501			4	1	1	6

Showing 1 to 1 of 285 entries Previous 1 2 3 4 5 … 285 Next

5.2.4.6 CV 6

Show 1 entries Search:

	Norm. MAE adj. *p* value	Norm. RMSE adj. *p* value	R^2 adj. *p* value	Norm. MAE rank	Norm. RMSE rank	R^2 rank	Rank sum
Neural network grid model 188	0.1155	0.9698	0.9711	138	4	4	146

Showing 1 to 1 of 120 entries Previous 1 2 3 4 5 … 120 Next

5.2.4.7 CV 7

Show 1 entries Search:

	Norm. MAE adj. *p* value	Norm. RMSE adj. *p* value	R^2 adj. *p* value	Norm. MAE rank	Norm. RMSE rank	R^2 rank	Rank sum
Neural network grid model 176	0.3253	0.857	1	28	4	4	36

Showing 1 to 1 of 86 entries Previous 1 2 3 4 5 … 86 Next

5.2.4.8 CV 8

Show 1 entries Search:

	Norm. MAE adj. *p* value	Norm. RMSE adj. *p* value	R^2 adj. *p* value	Norm. MAE rank	Norm. RMSE rank	R^2 rank	Rank sum
Neural network grid model 201	0.1795			148	1	1	150

Showing 1 to 1 of 66 entries Previous 1 2 3 4 5 … 66 Next

5.2.4.9 CV 9

Show 1 entries Search:

	Norm. MAE adj. *p* value	Norm. RMSE adj. *p* value	R^2 adj. *p* value	Norm. MAE rank	Norm. RMSE rank	R^2 rank	Rank sum
Neural network grid model 139	0.1085	0.7419	0.354	104	10	10	124

Showing 1 to 1 of 53 entries Previous 1 2 3 4 5 … 53 Next

5.2.4.10 CV 10

Show 1 entries Search:

	Norm. MAE adj. *p* value	Norm. RMSE adj. *p* value	R^2 adj. *p* value	Norm. MAE rank	Norm. RMSE rank	R^2 rank	Rank sum
Neural network grid model 160	0.3142	0.8091	0.5281	95	3	3	101

Showing 1 to 1 of 146 entries Previous 1 2 3 4 5 … 146 Next

5.2.5 Final evaluation

```
load(file = "./rdata/var_reduct_mb_train_test_splits_y_shuffled.RData")
load(file = "./rdata/model_training_grid_models_mb_ad.RData")
dir <- "/Users/renee/Downloads/melanin_binding_adversarial"
selected_models <- c(
  "Neural network grid model 113", "Neural network grid model 187",
  "Neural network grid model 145", "Neural network grid model 215",
  "Super learner final", "Neural network grid model 188",
  "Neural network grid model 176", "Neural network grid model 201",
  "Neural network grid model 139", "Neural network grid model 160"
)
mae <- c()
rmse <- c()
R2 <- c()
for (i in 1:10) {
  holdout_pred <- read.csv(paste0("./other_data/melanin_binding_adversarial/outer_", i,
↪  "/cv_holdout_predictions.csv"))
  if (startsWith(selected_models[i], "Neural network grid model")) {
    grid_num <- as.integer(str_extract(selected_models[i], "[0-9]+"))
    m <- intersect(colnames(holdout_pred), dl_grid[[grid_num]])
```

```
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/", m))
  } else if (startsWith(selected_models[i], "GBM grid model")) {
    grid_num <- as.integer(str_extract(selected_models[i], "[0-9]+"))
    m <- intersect(colnames(holdout_pred), gbm_grid[[grid_num]])
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/", m))
  } else if (startsWith(selected_models[i], "GBM model")) {
    grid_num <- as.integer(str_extract(selected_models[i], "[0-9]+"))
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/GBM_", grid_num))
  } else if (startsWith(selected_models[i], "XGBoost grid model")) {
    grid_num <- as.integer(str_extract(selected_models[i], "[0-9]+"))
    m <- intersect(colnames(holdout_pred), xgboost_grid[[grid_num]])
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/", m))
  } else if (startsWith(selected_models[i], "XGBoost model")) {
    grid_num <- as.integer(str_extract(selected_models[i], "[0-9]+"))
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/XGBoost_", grid_num))
  } else if (selected_models[i] == "Super learner all models") {
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/superlearner_iter_0"))
  } else if (selected_models[i] == "Super learner final") {
    files <- list.files(paste0(dir, "/outer_", i))
    files <- files[startsWith(files, "superlearner_iter")]
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/", files[length(files)]))
  } else if (selected_models[i] == "Super learner neural network") {
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/superlearner_deeplearning"))
  } else if (selected_models[i] == "Super learner GBM") {
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/superlearner_gbm"))
  } else if (selected_models[i] == "Super learner XGBoost") {
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/superlearner_xgboost"))
  } else {
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/", unlist(strsplit(selected_models[i], "
↪  "))[1]))
  }
  tmp <- as.h2o(subset(outer_splits[[i]][["test"]], select = -log_intensity))
  y_true <- outer_splits[[i]][["test"]]$log_intensity
  y_pred <- as.data.frame(as.data.frame(h2o.predict(model, tmp)))[, 1]
  mae <- c(mae, MAE(y_pred = y_pred, y_true = y_true))
  rmse <- c(rmse, RMSE(y_pred = y_pred, y_true = y_true))
  R2 <- c(R2, R2_Score(y_pred = y_pred, y_true = y_true))
}

# Re-scale to 0-100
mae <- rescale(mae, to = c(0, 100), from = range(mb_data$log_intensity))
```

```
rmse <- rescale(rmse, to = c(0, 100), from = range(mb_data$log_intensity))

h2o.removeAll()
save(mae, rmse, R2, file = "./rdata/final_evaluation_mb_ad.RData")
```

```
load(file = "./rdata/final_evaluation_mb_ad.RData")
data.frame(
  Metric = c("Norm. MAE", "Norm. RMSE", "R<sup>2</sup>"),
  `Mean ± s.e.m.` = c(
    paste0(round(mean(mae), 3), " ± ", round(sem(mae), 3)),
    paste0(round(mean(rmse), 3), " ± ", round(sem(rmse), 3)),
    paste0(round(mean(R2), 3), " ± ", round(sem(R2), 3))
  ),
  check.names = FALSE
) %>%
  datatable(escape = FALSE, rownames = FALSE)
```

Show 10 entries Search:

Metric	Mean ± s.e.m.
Norm. MAE	32.857 ± 0.173
Norm. RMSE	34.344 ± 0.226
R^2	-0.038 ± 0.015

Showing 1 to 3 of 3 entries Previous 1 Next

5.3 Cell-penetration (classification)

```
load(file = "./rdata/var_reduct_cpp_train_test_splits.RData")
set.seed(32)

# Shuffle the response variable of cross-validation training sets
for (i in 1:10) {
  response <- outer_splits[[i]]$train$category
  response <- sample(response)
  outer_splits[[i]]$train$category <- response
}
```

```
# Shuffle the response variable of the whole data set
response <- cpp_data$category
response <- sample(response)
cpp_data$category <- response

save(cpp_data, outer_splits,
  file = "./rdata/var_reduct_cpp_train_test_splits_y_shuffled.RData"
)
```

5.3.1 Model training

```
train_cpp_models <- function(train_set, exp_dir, prefix, nfolds = 10, grid_seed = 1) {
  tmp <- as.h2o(train_set, destination_frame = prefix)
  tmp["category"] <- as.factor(tmp["category"])
  y <- "category"
  x <- setdiff(names(tmp), y)
  res <- as.data.frame(tmp$category)
  samp_factors <- as.vector(mean(table(train_set$category)) / table(train_set$category))
  # -------------------
  # base model training
  # -------------------
  cat("Deep learning grid 1\n")
  deeplearning_1 <- h2o.grid(
    algorithm = "deeplearning", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
    keep_cross_validation_predictions = TRUE, hyper_params = deeplearning_params_1,
    stopping_rounds = 3, balance_classes = TRUE, class_sampling_factors = samp_factors,
    search_criteria = list(
      strategy = "RandomDiscrete",
      max_models = 100, seed = grid_seed
    ),
    keep_cross_validation_models = FALSE, fold_assignment = "Modulo", parallelism = 0
  )
  for (model_id in deeplearning_1@model_ids) {
    tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
  }
  cat("GBM grid\n")
  gbm <- h2o.grid(
    algorithm = "gbm", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
    keep_cross_validation_predictions = TRUE, hyper_params = gbm_params, stopping_rounds = 3,
```

```
  balance_classes = TRUE, class_sampling_factors = samp_factors,
  search_criteria = list(strategy = "RandomDiscrete", max_models = 100, seed = grid_seed),
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo", parallelism = 0
)
for (model_id in gbm@model_ids) {
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
}
cat("GBM 5 default models\n")
gbm_1 <- h2o.gbm(
  model_id = "GBM_1", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  balance_classes = TRUE, class_sampling_factors = samp_factors,
  col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8, learn_rate = 0.1,
  max_depth = 6, min_rows = 1, min_split_improvement = 1e-5, ntrees = 10000, sample_rate = 0.8,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
gbm_2 <- h2o.gbm(
  model_id = "GBM_2", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  balance_classes = TRUE, class_sampling_factors = samp_factors,
  col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8, learn_rate = 0.1,
  max_depth = 7, min_rows = 10, min_split_improvement = 1e-5, ntrees = 10000, sample_rate = 0.8,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
gbm_3 <- h2o.gbm(
  model_id = "GBM_3", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  balance_classes = TRUE, class_sampling_factors = samp_factors,
  col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8, learn_rate = 0.1,
  max_depth = 8, min_rows = 10, min_split_improvement = 1e-5, ntrees = 10000, sample_rate = 0.8,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
gbm_4 <- h2o.gbm(
  model_id = "GBM_4", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  balance_classes = TRUE, class_sampling_factors = samp_factors,
  col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8, learn_rate = 0.1,
  max_depth = 10, min_rows = 10, min_split_improvement = 1e-5, ntrees = 10000, sample_rate = 0.8,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
gbm_5 <- h2o.gbm(
```

```
  model_id = "GBM_5", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  balance_classes = TRUE, class_sampling_factors = samp_factors,
  col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8, learn_rate = 0.1,
  max_depth = 15, min_rows = 100, min_split_improvement = 1e-5, ntrees = 10000, sample_rate = 0.8,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
for (model_id in paste0("GBM_", 1:5)) {
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
}
cat("XGBoost grid\n")
xgboost <- h2o.grid(
  algorithm = "xgboost", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, hyper_params = xgboost_params, stopping_rounds = 3,
  search_criteria = list(strategy = "RandomDiscrete", max_models = 100, seed = grid_seed),
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo", parallelism = 0
)
for (model_id in xgboost@model_ids) {
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
}
cat("XGBoost 3 default models\n")
xgboost_1 <- h2o.xgboost(
  model_id = "XGBoost_1", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  booster = "gbtree", col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8,
  max_depth = 10, min_rows = 5, ntrees = 10000, reg_alpha = 0, reg_lambda = 1,
  sample_rate = 0.6, keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
xgboost_2 <- h2o.xgboost(
  model_id = "XGBoost_2", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  booster = "gbtree", col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8,
  max_depth = 20, min_rows = 10, ntrees = 10000, reg_alpha = 0, reg_lambda = 1,
  sample_rate = 0.6, keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
xgboost_3 <- h2o.xgboost(
  model_id = "XGBoost_3", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  booster = "gbtree", col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8,
  max_depth = 5, min_rows = 3, ntrees = 10000, reg_alpha = 0, reg_lambda = 1,
  sample_rate = 0.8, keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
```

```
)
for (model_id in paste0("XGBoost_", 1:3)) {
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
}
cat("DRF\n")
drf <- h2o.randomForest(
  model_id = "DRF", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, ntrees = 10000,
  score_tree_interval = 5, stopping_rounds = 3,
  balance_classes = TRUE, class_sampling_factors = samp_factors,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
tmp_path <- h2o.saveModel(h2o.getModel("DRF"), path = exp_dir, force = TRUE)
cat("XRT\n")
xrt <- h2o.randomForest(
  model_id = "XRT", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, ntrees = 10000, histogram_type = "Random",
  score_tree_interval = 5, stopping_rounds = 3,
  balance_classes = TRUE, class_sampling_factors = samp_factors,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
tmp_path <- h2o.saveModel(h2o.getModel("XRT"), path = exp_dir, force = TRUE)
# ----------------------
# get holdout predictions
# ----------------------
base_models <- as.list(c(
  unlist(deeplearning_1@model_ids),
  unlist(gbm@model_ids), paste0("GBM_", 1:5),
  unlist(xgboost@model_ids), paste0("XGBoost_", 1:3),
  "DRF",
  "XRT"
))
for (model_id in base_models) {
  res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_",
↪ model_id))["penetrating"],
    col.names = model_id
  ))
}
# ----------------------
# super learner training
# ----------------------
```

```
  sl_iter <- 0
  cat(paste0("Super learner iteration ", sl_iter, " (", length(base_models), " models)\n"))
  sl <- h2o.stackedEnsemble(
    x = x, y = y, model_id = paste0("superlearner_iter_", sl_iter),
    training_frame = tmp, seed = 1,
    base_models = base_models,
    metalearner_algorithm = "glm",
    metalearner_nfolds = nfolds,
    keep_levelone_frame = TRUE,
    metalearner_params = list(
      standardize = TRUE, keep_cross_validation_predictions = TRUE,
      balance_classes = TRUE, class_sampling_factors = samp_factors
    )
  )
  tmp_path <- h2o.saveModel(h2o.getModel(paste0("superlearner_iter_", sl_iter)), path = exp_dir, force
↪  = TRUE)
  model_id <- paste0("metalearner_glm_superlearner_iter_", sl_iter)
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
  res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_",
↪  model_id))["penetrating"],
    col.names = paste0(model_id, "_", length(base_models), "_models")
  ))
  # ---------------------------------
  # super learner base model reduction
  # ---------------------------------
  while (TRUE) {
    meta <- h2o.getModel(paste0("metalearner_glm_superlearner_iter_", sl_iter, "_cv_1"))
    names <- meta@model$coefficients_table[, "names"]
    coeffs <- (meta@model$coefficients_table[, "standardized_coefficients"] > 0)
    for (j in 2:nfolds) {
      meta <- h2o.getModel(paste0("metalearner_glm_superlearner_iter_", sl_iter, "_cv_", j))
      names <- meta@model$coefficients_table[, "names"]
      coeffs <- coeffs + (meta@model$coefficients_table[, "standardized_coefficients"] > 0)
    }
    base_models_ <- as.list(names[coeffs >= ceiling(nfolds / 2) & names != "Intercept"])
    if (length(base_models_) == 0) {
      cat("No base models passing the threshold\n\n")
      break
    }
    if (sum(base_models %in% base_models_) == length(base_models)) {
      cat("No further reduction of base models\n\n")
```

```
    break
  }
  sl_iter <- sl_iter + 1
  base_models <- base_models_
  cat(paste0("Super learner iteration ", sl_iter, " (", length(base_models), " models)\n"))
  sl <- h2o.stackedEnsemble(
    x = x, y = y, model_id = paste0("superlearner_iter_", sl_iter),
    training_frame = tmp, seed = 1,
    base_models = base_models,
    metalearner_algorithm = "glm",
    metalearner_nfolds = nfolds,
    keep_levelone_frame = TRUE,
    metalearner_params = list(
      standardize = TRUE, keep_cross_validation_predictions = TRUE,
      balance_classes = TRUE, class_sampling_factors = samp_factors
    )
  )
  tmp_path <- h2o.saveModel(h2o.getModel(paste0("superlearner_iter_", sl_iter)), path = exp_dir,
↪  force = TRUE)
  model_id <- paste0("metalearner_glm_superlearner_iter_", sl_iter)
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
  res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_",
↪  model_id))["penetrating"],
    col.names = paste0(model_id, "_", length(base_models), "_models")
  ))
}
# ----------------------------------------
# super learner for homogeneous base models
# ----------------------------------------
# DeepLearning
base_models <- deeplearning_1@model_ids
cat(paste0("Super learner deep learning (", length(base_models), " models)\n"))
sl <- h2o.stackedEnsemble(
  x = x, y = y, model_id = "superlearner_deeplearning",
  training_frame = tmp, seed = 1,
  base_models = base_models,
  metalearner_algorithm = "glm",
  metalearner_nfolds = nfolds,
  keep_levelone_frame = TRUE,
  metalearner_params = list(
    standardize = TRUE, keep_cross_validation_predictions = TRUE,
```

```
    balance_classes = TRUE, class_sampling_factors = samp_factors
  )
)
tmp_path <- h2o.saveModel(h2o.getModel("superlearner_deeplearning"), path = exp_dir, force = TRUE)
model_id <- "metalearner_glm_superlearner_deeplearning"
tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_",
↪  model_id))["penetrating"],
  col.names = paste0(model_id, "_", length(base_models), "_models")
))
# GBM
base_models <- as.list(c(unlist(gbm@model_ids), paste0("GBM_", 1:5)))
cat(paste0("Super learner GBM (", length(base_models), " models)\n"))
sl <- h2o.stackedEnsemble(
  x = x, y = y, model_id = "superlearner_gbm",
  training_frame = tmp, seed = 1,
  base_models = base_models,
  metalearner_algorithm = "glm",
  metalearner_nfolds = nfolds,
  keep_levelone_frame = TRUE,
  metalearner_params = list(
    standardize = TRUE, keep_cross_validation_predictions = TRUE,
    balance_classes = TRUE, class_sampling_factors = samp_factors
  )
)
tmp_path <- h2o.saveModel(h2o.getModel("superlearner_gbm"), path = exp_dir, force = TRUE)
model_id <- "metalearner_glm_superlearner_gbm"
tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_",
↪  model_id))["penetrating"],
  col.names = paste0(model_id, "_", length(base_models), "_models")
))
# XGBoost
base_models <- as.list(c(unlist(xgboost@model_ids), paste0("XGBoost_", 1:3)))
cat(paste0("Super learner XGBoost (", length(base_models), " models)\n"))
sl <- h2o.stackedEnsemble(
  x = x, y = y, model_id = "superlearner_xgboost",
  training_frame = tmp, seed = 1,
  base_models = base_models,
  metalearner_algorithm = "glm",
  metalearner_nfolds = nfolds,
```

```
    keep_levelone_frame = TRUE,
    metalearner_params = list(
      standardize = TRUE, keep_cross_validation_predictions = TRUE,
      balance_classes = TRUE, class_sampling_factors = samp_factors
    )
  )
  tmp_path <- h2o.saveModel(h2o.getModel("superlearner_xgboost"), path = exp_dir, force = TRUE)
  model_id <- "metalearner_glm_superlearner_xgboost"
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
  res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_",
↪  model_id))["penetrating"],
    col.names = paste0(model_id, "_", length(base_models), "_models")
  ))
  write.csv(res, file = paste0(exp_dir, "/cv_holdout_predictions.csv"), row.names = FALSE)
  cat("\n\n")
  h2o.removeAll()
}
```

5.3.2 Inner cross-validation

```
load(file = "./rdata/var_reduct_cpp_train_test_splits_y_shuffled.RData")
for (i in 1:10) {
  cat(paste0("Outer training set ", i, "\n"))
  prefix <- paste0("outer_", i)
  exp_dir <- paste0("/Users/renee/Downloads/cell_penetration_adversarial/", prefix)
  dir.create(exp_dir)
  train_cpp_models(train_set = outer_splits[[i]][["train"]], exp_dir = exp_dir, prefix = prefix)
}
```

```
# Keep track of grid models
dl_grid <- list()
gbm_grid <- list()
xgboost_grid <- list()
dl_grid_params <- list()
gbm_grid_params <- list()
xgboost_grid_params <- list()
for (i in 1:10) {
  cat(paste0("Outer training set ", i, "\n"))
```

```
prefix <- paste0("outer_", i)
dir <- paste0("/Users/renee/Downloads/cell_penetration_adversarial/", prefix)
files <- list.files(dir)
# Deep learning
dl <- files[str_detect(files, "DeepLearning_model")]
for (m in dl) {
  model <- h2o.loadModel(paste0(dir, "/", m))
  hs <- sha1(paste(c(
    model@allparameters$epsilon,
    model@allparameters$hidden,
    model@allparameters$hidden_dropout_ratios,
    model@allparameters$input_dropout_ratio,
    model@allparameters$rho
  ), collapse = " "))
  if (hs %in% names(dl_grid)) {
    dl_grid[[hs]] <- c(dl_grid[[hs]], m)
  } else {
    dl_grid[[hs]] <- c(m)
    dl_grid_params <- list.append(
      dl_grid_params,
      c(
        "epsilon" = model@allparameters$epsilon,
        "hidden" = paste0("[", paste(model@allparameters$hidden, collapse = ","), "]"),
        "hidden_dropout_ratios" = paste0(
          "[",
          paste(model@allparameters$hidden_dropout_ratios,
            collapse = ","
          ), "]"
        ),
        "input_dropout_ratio" = model@allparameters$input_dropout_ratio,
        "rho" = model@allparameters$rho
      )
    )
  }
}
h2o.removeAll()
# GBM
gbm <- files[str_detect(files, "GBM_model")]
for (m in gbm) {
  model <- h2o.loadModel(paste0(dir, "/", m))
  hs <- sha1(paste(c(
```

```
    model@allparameters$col_sample_rate,
    model@allparameters$col_sample_rate_per_tree,
    model@allparameters$max_depth,
    model@allparameters$min_rows,
    model@allparameters$min_split_improvement,
    model@allparameters$sample_rate
  ), collapse = " "))
  if (hs %in% names(gbm_grid)) {
    gbm_grid[[hs]] <- c(gbm_grid[[hs]], m)
  } else {
    gbm_grid[[hs]] <- c(m)
    gbm_grid_params <- list.append(
      gbm_grid_params,
      c(
        "col_sample_rate" = model@allparameters$col_sample_rate,
        "col_sample_rate_per_tree" = model@allparameters$col_sample_rate_per_tree,
        "max_depth" = model@allparameters$max_depth,
        "min_rows" = model@allparameters$min_rows,
        "min_split_improvement" = model@allparameters$min_split_improvement,
        "sample_rate" = model@allparameters$sample_rate
      )
    )
  }
}
h2o.removeAll()
# XGBoost
xgboost <- files[str_detect(files, "XGBoost_model")]
for (m in xgboost) {
  model <- h2o.loadModel(paste0(dir, "/", m))
  hs <- sha1(paste(c(
    model@allparameters$booster,
    model@allparameters$col_sample_rate,
    model@allparameters$col_sample_rate_per_tree,
    model@allparameters$max_depth,
    model@allparameters$min_rows,
    model@allparameters$reg_alpha,
    model@allparameters$reg_lambda,
    model@allparameters$sample_rate
  ), collapse = " "))
  if (hs %in% names(xgboost_grid)) {
    xgboost_grid[[hs]] <- c(xgboost_grid[[hs]], m)
```

```
    } else {
      xgboost_grid[[hs]] <- c(m)
      xgboost_grid_params <- list.append(
        xgboost_grid_params,
        c(
          "booster" = model@allparameters$booster,
          "col_sample_rate" = model@allparameters$col_sample_rate,
          "col_sample_rate_per_tree" = model@allparameters$col_sample_rate_per_tree,
          "max_depth" = model@allparameters$max_depth,
          "min_rows" = model@allparameters$min_rows,
          "reg_alpha" = model@allparameters$reg_alpha,
          "reg_lambda" = model@allparameters$reg_lambda,
          "sample_rate" = model@allparameters$sample_rate
        )
      )
    }
  }
  h2o.removeAll()
}

dl_grid_params <- as.data.frame(t(data.frame(dl_grid_params)))
rownames(dl_grid_params) <- paste("Neural network grid model", 1:nrow(dl_grid_params))

gbm_grid_params <- as.data.frame(t(data.frame(gbm_grid_params)))
rownames(gbm_grid_params) <- paste("GBM grid model", 1:nrow(gbm_grid_params))

xgboost_grid_params <- as.data.frame(t(data.frame(xgboost_grid_params)))
rownames(xgboost_grid_params) <- paste("XGBoost grid model", 1:nrow(xgboost_grid_params))

write.csv(dl_grid_params, "./other_data/cell_penetration_adversarial/neural_network_grid_params.csv")
write.csv(gbm_grid_params, "./other_data/cell_penetration_adversarial/gbm_grid_params.csv")
write.csv(xgboost_grid_params, "./other_data/cell_penetration_adversarial/xgboost_grid_params.csv")

save(dl_grid, gbm_grid, xgboost_grid, file = "./rdata/model_training_grid_models_cpp_ad.RData")
```

5.3.3 Model evaluation

```
model_evaluation <- function(holdout_pred, fold_asign, grid_name, grid_meta = NULL) {
  res_ <- list()
```

```
  model_num <- c()
  if (startsWith(grid_name, "Super learner")) {
    if (grid_name == "Super learner all models") {
      m <- colnames(holdout_pred)[startsWith(colnames(holdout_pred),
↪  "metalearner_glm_superlearner_iter_0")]
    } else if (grid_name == "Super learner final") {
      sl_models <- colnames(holdout_pred)[startsWith(colnames(holdout_pred),
↪  "metalearner_glm_superlearner_iter")]
      m <- sl_models[length(sl_models)]
    } else if (grid_name == "Super learner neural network") {
      m <- colnames(holdout_pred)[startsWith(colnames(holdout_pred),
↪  "metalearner_glm_superlearner_deeplearning")]
    } else if (grid_name == "Super learner GBM") {
      m <- colnames(holdout_pred)[startsWith(colnames(holdout_pred),
↪  "metalearner_glm_superlearner_gbm")]
    } else if (grid_name == "Super learner XGBoost") {
      m <- colnames(holdout_pred)[startsWith(colnames(holdout_pred),
↪  "metalearner_glm_superlearner_xgboost")]
    }
    logloss <- c()
    MCC <- c()
    F1 <- c()
    acc <- c()
    EF <- c()
    BEDROC <- c()
    threshold <- 0.5
    for (k in 1:10) {
      y_true <- sapply(holdout_pred[fold_assign == k, "category"], function(x) if (x == "penetrating")
↪  1 else 0)
      y_pred <- holdout_pred[fold_assign == k, m]
      y_pred_cls <- sapply(y_pred, function(x) if (x >= threshold) 1 else 0)
      logloss <- c(logloss, LogLoss(y_pred = y_pred, y_true = y_true))
      MCC <- c(MCC, mcc(preds = y_pred_cls, actuals = y_true))
      F1 <- c(F1, F1_Score(y_pred = y_pred_cls, y_true = y_true))
      acc <- c(acc, balanced_accuracy(y_pred = y_pred_cls, y_true = y_true))
      EF <- c(EF, enrichment_factor(x = y_pred, y = y_true, top = 0.05))
      BEDROC <- c(BEDROC, bedroc(x = y_pred, y = y_true, alpha = 20))
    }
    res_ <- list.append(res_, c(
      mean(logloss, na.rm = TRUE), sem(logloss),
      mean(MCC, na.rm = TRUE), sem(MCC),
```

```
      mean(F1, na.rm = TRUE), sem(F1),
      mean(acc, na.rm = TRUE), sem(acc),
      mean(EF, na.rm = TRUE), sem(EF),
      mean(BEDROC, na.rm = TRUE), sem(BEDROC),
      logloss, MCC, F1, acc, EF, BEDROC
    ))
  } else {
    for (j in 1:length(grid_meta)) {
      g <- grid_meta[[j]]
      m <- intersect(colnames(holdout_pred), g)
      if (length(m) == 1) {
        model_num <- c(model_num, j)
        logloss <- c()
        MCC <- c()
        F1 <- c()
        acc <- c()
        EF <- c()
        BEDROC <- c()
        threshold <- 0.5
        for (k in 1:10) {
          y_true <- sapply(holdout_pred[fold_assign == k, "category"], function(x) if (x ==
↪  "penetrating") 1 else 0)
          y_pred <- holdout_pred[fold_assign == k, m]
          y_pred_cls <- sapply(y_pred, function(x) if (x >= threshold) 1 else 0)
          logloss <- c(logloss, LogLoss(y_pred = y_pred, y_true = y_true))
          MCC <- c(MCC, mcc(preds = y_pred_cls, actuals = y_true))
          F1 <- c(F1, F1_Score(y_pred = y_pred_cls, y_true = y_true))
          acc <- c(acc, balanced_accuracy(y_pred = y_pred_cls, y_true = y_true))
          EF <- c(EF, enrichment_factor(x = y_pred, y = y_true, top = 0.05))
          BEDROC <- c(BEDROC, bedroc(x = y_pred, y = y_true, alpha = 20))
        }
        res_ <- list.append(res_, c(
          mean(logloss, na.rm = TRUE), sem(logloss),
          mean(MCC, na.rm = TRUE), sem(MCC),
          mean(F1, na.rm = TRUE), sem(F1),
          mean(acc, na.rm = TRUE), sem(acc),
          mean(EF, na.rm = TRUE), sem(EF),
          mean(BEDROC, na.rm = TRUE), sem(BEDROC),
          logloss, MCC, F1, acc, EF, BEDROC
        ))
      }
```

```
    }
  }
  res <- as.data.frame(t(data.frame(res_)))
  colnames(res) <- c(
    "Log loss mean", "Log loss s.e.m.", "MCC mean", "MCC s.e.m.", "F_1 mean", "F_1 s.e.m.",
    "Balanced accuracy mean", "Balanced accuracy s.e.m.", "EF mean", "EF s.e.m.", "BEDROC mean",
    ↪  "BEDROC s.e.m.",
    paste("Log loss CV", 1:10), paste("MCC CV", 1:10), paste("F_1 CV", 1:10),
    paste("Balanced accuracy CV", 1:10), paste("EF CV", 1:10), paste("BEDROC CV", 1:10)
  )
  if (nrow(res) == 1) {
    rownames(res) <- grid_name
  } else {
    rownames(res) <- paste(grid_name, model_num)
  }
  return(res)
}
```

5.3.4 Inner loop model selection

```
load(file = "./rdata/model_training_grid_models_cpp_ad.RData")

for (i in 1:10) {
  holdout_pred <- read.csv(paste0("./other_data/cell_penetration_adversarial/outer_", i,
↪  "/cv_holdout_predictions.csv"))
  fold_assign <- rep(1:10, ceiling(nrow(holdout_pred) / 10))[1:nrow(holdout_pred)]
  data_ <- list()
  # Deep learning grid models
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
    grid_name = "Neural network grid model",
    grid_meta = dl_grid
  ))
  # GBM grid models
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
    grid_name = "GBM grid model",
    grid_meta = gbm_grid
  ))
  # GBM default models
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
```

```
    grid_name = "GBM model",
    grid_meta = as.list(paste0("GBM_", 1:5))
  ))
  # XGBoost grid models
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
    grid_name = "XGBoost grid model",
    grid_meta = xgboost_grid
  ))
  # XGBoost default models
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
    grid_name = "XGBoost model",
    grid_meta = as.list(paste0("XGBoost_", 1:3))
  ))
  # DRF
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "DRF model",
↪   grid_meta = list("DRF")))
  # XRT
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "XRT model",
↪   grid_meta = list("XRT")))
  # Super learner all
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪   all models"))
  # Super learner final
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪   final"))
  # Super learner deep learning
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪   neural network"))
  # Super learner GBM
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪   GBM"))
  # Super learner XGBoost
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪   XGBoost"))

  data <- do.call(rbind, data_)
  write.csv(data, paste0("./other_data/cell_penetration_adversarial/outer_", i, "/cv_res.csv"))
}
```

5.3.4.1 CV 1

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
GBM grid model 89	0.9698	0.6864	0.2194	1	6	3	8	8	25

Showing 1 to 1 of 82 entries Previous 1 2 3 4 5 … 82 Next

5.3.4.2 CV 2

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
Super learner XGBoost					1	1	1	1	4

Showing 1 to 1 of 26 entries Previous 1 2 3 4 5 … 26 Next

5.3.4.3 CV 3

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
GBM grid model 75	0.9618	0.5239	0.9716	0.8583	2	2	6	7	17

Showing 1 to 1 of 65 entries Previous 1 2 3 4 5 … 65 Next

5.3.4.4 CV 4

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
GBM grid model 94		0.4627	0.0039	0.2956	1	15	131	15	162

Showing 1 to 1 of 25 entries Previous 1 2 3 4 5 … 25 Next

5.3.4.5 CV 5

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
GBM grid model 21	0.9618	0.6841	0.0845	0.4819	3.5	4.5	15.5	4.5	28

Showing 1 to 1 of 26 entries Previous 1 2 3 4 5 … 26 Next

5.3.4.6 CV 6

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
Super learner final	0.4829	0.2775	0.0567	0.215	24	10	54	12	100

Showing 1 to 1 of 26 entries Previous 1 2 3 4 5 … 26 Next

5.3.4.7 CV 7

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
GBM grid model 15	0.4605	0.4329	0.1477	0.3105	7	3	31	3	44

Showing 1 to 1 of 28 entries Previous 1 2 3 4 5 … 28 Next

5.3.4.8 CV 8

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
GBM grid model 6		0.5324	0.0182	0.4503	1	15	46	20	82

Showing 1 to 1 of 28 entries Previous 1 2 3 4 5 … 28 Next

5.3.4.9 CV 9

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
GBM grid model 76	0.8215	0.1933		0.9729	8	3	1	3	15

Showing 1 to 1 of 30 entries Previous 1 2 3 4 5 … 30 Next

5.3.4.10 CV 10

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
Neural network grid model 86	0.1252	0.7432	0.1261	0.8501	78	6	49	4	137

Showing 1 to 1 of 33 entries Previous 1 2 3 4 5 … 33 Next

5.3.5 Final evaluation

```
load(file = "./rdata/var_reduct_cpp_train_test_splits_y_shuffled.RData")
load(file = "./rdata/model_training_grid_models_cpp_ad.RData")
dir <- "/Users/renee/Downloads/cell_penetration_adversarial"
selected_models <- c(
  "GBM grid model 68", "Super learner XGBoost", "GBM grid model 75",
  "Super learner XGBoost", "GBM grid model 21", "Super learner final",
  "GBM grid model 15", "GBM grid model 6", "GBM grid model 76",
  "Neural network grid model 86"
)
logloss <- c()
MCC <- c()
F1 <- c()
acc <- c()
EF <- c()
BEDROC <- c()
threshold <- 0.5
for (i in 1:10) {
  holdout_pred <- read.csv(paste0("./other_data/cell_penetration_adversarial/outer_", i,
↪  "/cv_holdout_predictions.csv"))
```

```
  if (startsWith(selected_models[i], "Neural network grid model")) {
    grid_num <- as.integer(str_extract(selected_models[i], "[0-9]+"))
    m <- intersect(colnames(holdout_pred), dl_grid[[grid_num]])
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/", m))
  } else if (startsWith(selected_models[i], "GBM grid model")) {
    grid_num <- as.integer(str_extract(selected_models[i], "[0-9]+"))
    m <- intersect(colnames(holdout_pred), gbm_grid[[grid_num]])
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/", m))
  } else if (startsWith(selected_models[i], "GBM model")) {
    grid_num <- as.integer(str_extract(selected_models[i], "[0-9]+"))
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/GBM_", grid_num))
  } else if (startsWith(selected_models[i], "XGBoost grid model")) {
    grid_num <- as.integer(str_extract(selected_models[i], "[0-9]+"))
    m <- intersect(colnames(holdout_pred), xgboost_grid[[grid_num]])
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/", m))
  } else if (startsWith(selected_models[i], "XGBoost model")) {
    grid_num <- as.integer(str_extract(selected_models[i], "[0-9]+"))
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/XGBoost_", grid_num))
  } else if (selected_models[i] == "Super learner all models") {
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/superlearner_iter_0"))
  } else if (selected_models[i] == "Super learner final") {
    files <- list.files(paste0(dir, "/outer_", i))
    files <- files[startsWith(files, "superlearner_iter")]
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/", files[length(files)]))
  } else if (selected_models[i] == "Super learner neural network") {
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/superlearner_deeplearning"))
  } else if (selected_models[i] == "Super learner GBM") {
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/superlearner_gbm"))
  } else if (selected_models[i] == "Super learner XGBoost") {
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/superlearner_xgboost"))
  } else {
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/", unlist(strsplit(selected_models[i], "
↪  "))[1]))
  }
  tmp <- as.h2o(subset(outer_splits[[i]][["test"]], select = -category))
  y_true <- sapply(outer_splits[[i]][["test"]]$category, function(x) if (x == "penetrating") 1 else 0)
  y_pred <- as.data.frame(as.data.frame(h2o.predict(model, tmp)))[, "penetrating"]
  y_pred_cls <- sapply(y_pred, function(x) if (x >= threshold) 1 else 0)
  logloss <- c(logloss, LogLoss(y_pred = y_pred, y_true = y_true))
  MCC <- c(MCC, mcc(preds = y_pred_cls, actuals = y_true))
  F1 <- c(F1, F1_Score(y_pred = y_pred_cls, y_true = y_true))
```

```
  acc <- c(acc, balanced_accuracy(y_pred = y_pred_cls, y_true = y_true))
  EF <- c(EF, enrichment_factor(x = y_pred, y = y_true, top = 0.05))
  BEDROC <- c(BEDROC, bedroc(x = y_pred, y = y_true, alpha = 20))
}
save(logloss, MCC, F1, acc, EF, BEDROC, file = "./rdata/final_evaluation_cpp_ad.RData")
```

```
load(file = "./rdata/final_evaluation_cpp_ad.RData")
data.frame(
  Metric = c("Log loss", "MCC", "F<sub>1</sub>", "Balanced accuracy", "EF", "BEDROC"),
  `Mean ± s.e.m.` = c(
    paste0(round(mean(logloss), 3), " ± ", round(sem(logloss), 3)),
    paste0(round(mean(MCC), 3), " ± ", round(sem(MCC), 3)),
    paste0(round(mean(F1), 3), " ± ", round(sem(F1), 3)),
    paste0(round(mean(acc), 3), " ± ", round(sem(acc), 3)),
    paste0(round(mean(EF), 3), " ± ", round(sem(EF), 3)),
    paste0(round(mean(BEDROC), 3), " ± ", round(sem(BEDROC), 3))
  ),
  check.names = FALSE
) %>%
  datatable(escape = FALSE, rownames = FALSE)
```

Show 10 entries Search:

Metric	Mean ± s.e.m.
Log loss	0.715 ± 0.013
MCC	-0.002 ± 0.054
F_1	0.522 ± 0.031
Balanced accuracy	0.502 ± 0.028
EF	1.09 ± 0.159
BEDROC	0.529 ± 0.058

Showing 1 to 6 of 6 entries Previous 1 Next

5.4 Toxicity (classification)

```
load(file = "./rdata/var_reduct_tx_train_test_splits.RData")
set.seed(32)
```

```
# Shuffle the response variable of cross-validation training sets
for (i in 1:10) {
  response <- outer_splits[[i]]$train$category
  response <- sample(response)
  outer_splits[[i]]$train$category <- response
}

# Shuffle the response variable of the whole data set
response <- tx_data$category
response <- sample(response)
tx_data$category <- response

save(tx_data, outer_splits,
  file = "./rdata/var_reduct_tx_train_test_splits_y_shuffled.RData"
)
```

5.4.1 Model training

```
train_tx_models <- function(train_set, exp_dir, prefix, nfolds = 10, grid_seed = 1) {
  tmp <- as.h2o(train_set, destination_frame = prefix)
  tmp["category"] <- as.factor(tmp["category"])
  y <- "category"
  x <- setdiff(names(tmp), y)
  res <- as.data.frame(tmp$category)
  samp_factors <- as.vector(mean(table(train_set$category)) / table(train_set$category))
  # --------------------
  # base model training
  # --------------------
  cat("Deep learning grid 1\n")
  deeplearning_1 <- h2o.grid(
    algorithm = "deeplearning", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
    keep_cross_validation_predictions = TRUE, hyper_params = deeplearning_params_1,
    stopping_rounds = 3, balance_classes = TRUE, class_sampling_factors = samp_factors,
    max_after_balance_size = 0.5,
    search_criteria = list(
      strategy = "RandomDiscrete",
      max_models = 100, seed = grid_seed
    ),
```

```
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo", parallelism = 0
)
for (model_id in deeplearning_1@model_ids) {
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
}
cat("GBM grid\n")
gbm <- h2o.grid(
  algorithm = "gbm", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, hyper_params = gbm_params, stopping_rounds = 3,
  balance_classes = TRUE, class_sampling_factors = samp_factors, max_after_balance_size = 0.5,
  search_criteria = list(strategy = "RandomDiscrete", max_models = 100, seed = grid_seed),
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo", parallelism = 0
)
for (model_id in gbm@model_ids) {
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
}
cat("GBM 5 default models\n")
gbm_1 <- h2o.gbm(
  model_id = "GBM_1", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  balance_classes = TRUE, class_sampling_factors = samp_factors, max_after_balance_size = 0.5,
  col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8, learn_rate = 0.1,
  max_depth = 6, min_rows = 1, min_split_improvement = 1e-5, ntrees = 10000, sample_rate = 0.8,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
gbm_2 <- h2o.gbm(
  model_id = "GBM_2", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  balance_classes = TRUE, class_sampling_factors = samp_factors, max_after_balance_size = 0.5,
  col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8, learn_rate = 0.1,
  max_depth = 7, min_rows = 10, min_split_improvement = 1e-5, ntrees = 10000, sample_rate = 0.8,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
gbm_3 <- h2o.gbm(
  model_id = "GBM_3", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  balance_classes = TRUE, class_sampling_factors = samp_factors, max_after_balance_size = 0.5,
  col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8, learn_rate = 0.1,
  max_depth = 8, min_rows = 10, min_split_improvement = 1e-5, ntrees = 10000, sample_rate = 0.8,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
```

```
gbm_4 <- h2o.gbm(
  model_id = "GBM_4", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  balance_classes = TRUE, class_sampling_factors = samp_factors, max_after_balance_size = 0.5,
  col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8, learn_rate = 0.1,
  max_depth = 10, min_rows = 10, min_split_improvement = 1e-5, ntrees = 10000, sample_rate = 0.8,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
gbm_5 <- h2o.gbm(
  model_id = "GBM_5", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  balance_classes = TRUE, class_sampling_factors = samp_factors, max_after_balance_size = 0.5,
  col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8, learn_rate = 0.1,
  max_depth = 15, min_rows = 100, min_split_improvement = 1e-5, ntrees = 10000, sample_rate = 0.8,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
for (model_id in paste0("GBM_", 1:5)) {
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
}
cat("XGBoost grid\n")
xgboost <- h2o.grid(
  algorithm = "xgboost", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, hyper_params = xgboost_params, stopping_rounds = 3,
  search_criteria = list(strategy = "RandomDiscrete", max_models = 100, seed = grid_seed),
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo", parallelism = 0
)
for (model_id in xgboost@model_ids) {
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
}
cat("XGBoost 3 default models\n")
xgboost_1 <- h2o.xgboost(
  model_id = "XGBoost_1", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  booster = "gbtree", col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8,
  max_depth = 10, min_rows = 5, ntrees = 10000, reg_alpha = 0, reg_lambda = 1,
  sample_rate = 0.6, keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
xgboost_2 <- h2o.xgboost(
  model_id = "XGBoost_2", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  booster = "gbtree", col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8,
```

```
  max_depth = 20, min_rows = 10, ntrees = 10000, reg_alpha = 0, reg_lambda = 1,
  sample_rate = 0.6, keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
xgboost_3 <- h2o.xgboost(
  model_id = "XGBoost_3", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  booster = "gbtree", col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8,
  max_depth = 5, min_rows = 3, ntrees = 10000, reg_alpha = 0, reg_lambda = 1,
  sample_rate = 0.8, keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
for (model_id in paste0("XGBoost_", 1:3)) {
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
}
cat("GLM\n")
glm <- h2o.glm(
  model_id = "GLM", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, alpha = c(0.0, 0.2, 0.4, 0.6, 0.8, 1.0),
  balance_classes = TRUE, class_sampling_factors = samp_factors, max_after_balance_size = 0.5,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
tmp_path <- h2o.saveModel(h2o.getModel("GLM"), path = exp_dir, force = TRUE)
cat("DRF\n")
drf <- h2o.randomForest(
  model_id = "DRF", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, ntrees = 10000,
  score_tree_interval = 5, stopping_rounds = 3,
  balance_classes = TRUE, class_sampling_factors = samp_factors, max_after_balance_size = 0.5,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
tmp_path <- h2o.saveModel(h2o.getModel("DRF"), path = exp_dir, force = TRUE)
cat("XRT\n")
xrt <- h2o.randomForest(
  model_id = "XRT", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, ntrees = 10000, histogram_type = "Random",
  score_tree_interval = 5, stopping_rounds = 3,
  balance_classes = TRUE, class_sampling_factors = samp_factors, max_after_balance_size = 0.5,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
tmp_path <- h2o.saveModel(h2o.getModel("XRT"), path = exp_dir, force = TRUE)
# ----------------------
# get holdout predictions
```

```
# -----------------------
base_models <- as.list(c(
  unlist(deeplearning_1@model_ids),
  unlist(gbm@model_ids), paste0("GBM_", 1:5),
  unlist(xgboost@model_ids), paste0("XGBoost_", 1:3),
  "GLM",
  "DRF",
  "XRT"
))
for (model_id in base_models) {
  res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_", model_id))[3],
    col.names = model_id
  ))
}
# -----------------------
# super learner training
# -----------------------
sl_iter <- 0
cat(paste0("Super learner iteration ", sl_iter, " (", length(base_models), " models)\n"))
sl <- h2o.stackedEnsemble(
  x = x, y = y, model_id = paste0("superlearner_iter_", sl_iter),
  training_frame = tmp, seed = 1,
  base_models = base_models,
  metalearner_algorithm = "glm",
  metalearner_nfolds = nfolds,
  keep_levelone_frame = TRUE,
  metalearner_params = list(
    standardize = TRUE, keep_cross_validation_predictions = TRUE,
    balance_classes = TRUE, class_sampling_factors = samp_factors,
    max_after_balance_size = 0.5
  )
)
tmp_path <- h2o.saveModel(h2o.getModel(paste0("superlearner_iter_", sl_iter)), path = exp_dir, force
↪  = TRUE)
model_id <- paste0("metalearner_glm_superlearner_iter_", sl_iter)
tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_", model_id))[3],
  col.names = paste0(model_id, "_", length(base_models), "_models")
))
# ---------------------------------
# super learner base model reduction
```

```
# ---------------------------------
while (TRUE) {
  meta <- h2o.getModel(paste0("metalearner_glm_superlearner_iter_", sl_iter, "_cv_1"))
  names <- meta@model$coefficients_table[, "names"]
  coeffs <- (meta@model$coefficients_table[, "standardized_coefficients"] > 0)
  for (j in 2:nfolds) {
    meta <- h2o.getModel(paste0("metalearner_glm_superlearner_iter_", sl_iter, "_cv_", j))
    names <- meta@model$coefficients_table[, "names"]
    coeffs <- coeffs + (meta@model$coefficients_table[, "standardized_coefficients"] > 0)
  }
  base_models_ <- as.list(names[coeffs >= ceiling(nfolds / 2) & names != "Intercept"])
  if (length(base_models_) == 0) {
    cat("No base models passing the threshold\n\n")
    break
  }
  if (sum(base_models %in% base_models_) == length(base_models)) {
    cat("No further reduction of base models\n\n")
    break
  }
  sl_iter <- sl_iter + 1
  base_models <- base_models_
  cat(paste0("Super learner iteration ", sl_iter, " (", length(base_models), " models)\n"))
  sl <- h2o.stackedEnsemble(
    x = x, y = y, model_id = paste0("superlearner_iter_", sl_iter),
    training_frame = tmp, seed = 1,
    base_models = base_models,
    metalearner_algorithm = "glm",
    metalearner_nfolds = nfolds,
    keep_levelone_frame = TRUE,
    metalearner_params = list(
      standardize = TRUE, keep_cross_validation_predictions = TRUE,
      balance_classes = TRUE, class_sampling_factors = samp_factors,
      max_after_balance_size = 0.5
    )
  )
  tmp_path <- h2o.saveModel(h2o.getModel(paste0("superlearner_iter_", sl_iter)), path = exp_dir,
↪  force = TRUE)
  model_id <- paste0("metalearner_glm_superlearner_iter_", sl_iter)
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
  res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_", model_id))[3],
    col.names = paste0(model_id, "_", length(base_models), "_models")
```

```
  ))
}
# ----------------------------------------
# super learner for homogeneous base models
# ----------------------------------------
# DeepLearning
base_models <- deeplearning_1@model_ids
cat(paste0("Super learner deep learning (", length(base_models), " models)\n"))
sl <- h2o.stackedEnsemble(
  x = x, y = y, model_id = "superlearner_deeplearning",
  training_frame = tmp, seed = 1,
  base_models = base_models,
  metalearner_algorithm = "glm",
  metalearner_nfolds = nfolds,
  keep_levelone_frame = TRUE,
  metalearner_params = list(
    standardize = TRUE, keep_cross_validation_predictions = TRUE,
    balance_classes = TRUE, class_sampling_factors = samp_factors,
    max_after_balance_size = 0.5
  )
)
tmp_path <- h2o.saveModel(h2o.getModel("superlearner_deeplearning"), path = exp_dir, force = TRUE)
model_id <- "metalearner_glm_superlearner_deeplearning"
tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_", model_id))[3],
  col.names = paste0(model_id, "_", length(base_models), "_models")
))
# GBM
base_models <- as.list(c(unlist(gbm@model_ids), paste0("GBM_", 1:5)))
cat(paste0("Super learner GBM (", length(base_models), " models)\n"))
sl <- h2o.stackedEnsemble(
  x = x, y = y, model_id = "superlearner_gbm",
  training_frame = tmp, seed = 1,
  base_models = base_models,
  metalearner_algorithm = "glm",
  metalearner_nfolds = nfolds,
  keep_levelone_frame = TRUE,
  metalearner_params = list(
    standardize = TRUE, keep_cross_validation_predictions = TRUE,
    balance_classes = TRUE, class_sampling_factors = samp_factors,
    max_after_balance_size = 0.5
```

```
    )
  )
  tmp_path <- h2o.saveModel(h2o.getModel("superlearner_gbm"), path = exp_dir, force = TRUE)
  model_id <- "metalearner_glm_superlearner_gbm"
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
  res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_", model_id))[3],
    col.names = paste0(model_id, "_", length(base_models), "_models")
  ))
  # XGBoost
  base_models <- as.list(c(unlist(xgboost@model_ids), paste0("XGBoost_", 1:3)))
  cat(paste0("Super learner XGBoost (", length(base_models), " models)\n"))
  sl <- h2o.stackedEnsemble(
    x = x, y = y, model_id = "superlearner_xgboost",
    training_frame = tmp, seed = 1,
    base_models = base_models,
    metalearner_algorithm = "glm",
    metalearner_nfolds = nfolds,
    keep_levelone_frame = TRUE,
    metalearner_params = list(
      standardize = TRUE, keep_cross_validation_predictions = TRUE,
      balance_classes = TRUE, class_sampling_factors = samp_factors,
      max_after_balance_size = 0.5
    )
  )
  tmp_path <- h2o.saveModel(h2o.getModel("superlearner_xgboost"), path = exp_dir, force = TRUE)
  model_id <- "metalearner_glm_superlearner_xgboost"
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
  res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_", model_id))[3],
    col.names = paste0(model_id, "_", length(base_models), "_models")
  ))
  write.csv(res, file = paste0(exp_dir, "/cv_holdout_predictions.csv"), row.names = FALSE)
  cat("\n\n")
  h2o.removeAll()
}
```

5.4.2 Inner cross-validation

```
load(file = "./rdata/var_reduct_tx_train_test_splits_y_shuffled.RData")
for (i in 1:10) {
```

```
  cat(paste0("Outer training set ", i, "\n"))
  prefix <- paste0("outer_", i)
  exp_dir <- paste0("/Users/renee/Downloads/toxicity_adversarial/", prefix)
  dir.create(exp_dir)
  train_tx_models(train_set = outer_splits[[i]][["train"]], exp_dir = exp_dir, prefix = prefix)
}
```

```
# Keep track of grid models
dl_grid <- list()
gbm_grid <- list()
xgboost_grid <- list()
dl_grid_params <- list()
gbm_grid_params <- list()
xgboost_grid_params <- list()
for (i in 1:10) {
  cat(paste0("Outer training set ", i, "\n"))
  prefix <- paste0("outer_", i)
  dir <- paste0("/Users/renee/Downloads/toxicity_adversarial/", prefix)
  files <- list.files(dir)
  # Deep learning
  dl <- files[str_detect(files, "DeepLearning_model")]
  for (m in dl) {
    model <- h2o.loadModel(paste0(dir, "/", m))
    hs <- sha1(paste(c(
      model@allparameters$epsilon,
      model@allparameters$hidden,
      model@allparameters$hidden_dropout_ratios,
      model@allparameters$input_dropout_ratio,
      model@allparameters$rho
    ), collapse = " "))
    if (hs %in% names(dl_grid)) {
      dl_grid[[hs]] <- c(dl_grid[[hs]], m)
    } else {
      dl_grid[[hs]] <- c(m)
      dl_grid_params <- list.append(
        dl_grid_params,
        c(
          "epsilon" = model@allparameters$epsilon,
          "hidden" = paste0("[", paste(model@allparameters$hidden, collapse = ","), "]"),
```

```
        "hidden_dropout_ratios" = paste0(
          "[",
          paste(model@allparameters$hidden_dropout_ratios,
            collapse = ","
          ), "]"
        ),
        "input_dropout_ratio" = model@allparameters$input_dropout_ratio,
        "rho" = model@allparameters$rho
      )
    )
  }
}
h2o.removeAll()
# GBM
gbm <- files[str_detect(files, "GBM_model")]
for (m in gbm) {
  model <- h2o.loadModel(paste0(dir, "/", m))
  hs <- sha1(paste(c(
    model@allparameters$col_sample_rate,
    model@allparameters$col_sample_rate_per_tree,
    model@allparameters$max_depth,
    model@allparameters$min_rows,
    model@allparameters$min_split_improvement,
    model@allparameters$sample_rate
  ), collapse = " "))
  if (hs %in% names(gbm_grid)) {
    gbm_grid[[hs]] <- c(gbm_grid[[hs]], m)
  } else {
    gbm_grid[[hs]] <- c(m)
    gbm_grid_params <- list.append(
      gbm_grid_params,
      c(
        "col_sample_rate" = model@allparameters$col_sample_rate,
        "col_sample_rate_per_tree" = model@allparameters$col_sample_rate_per_tree,
        "max_depth" = model@allparameters$max_depth,
        "min_rows" = model@allparameters$min_rows,
        "min_split_improvement" = model@allparameters$min_split_improvement,
        "sample_rate" = model@allparameters$sample_rate
      )
    )
  }
```

```
  }
  h2o.removeAll()
  # XGBoost
  xgboost <- files[str_detect(files, "XGBoost_model")]
  for (m in xgboost) {
    model <- h2o.loadModel(paste0(dir, "/", m))
    hs <- sha1(paste(c(
      model@allparameters$booster,
      model@allparameters$col_sample_rate,
      model@allparameters$col_sample_rate_per_tree,
      model@allparameters$max_depth,
      model@allparameters$min_rows,
      model@allparameters$reg_alpha,
      model@allparameters$reg_lambda,
      model@allparameters$sample_rate
    ), collapse = " "))
    if (hs %in% names(xgboost_grid)) {
      xgboost_grid[[hs]] <- c(xgboost_grid[[hs]], m)
    } else {
      xgboost_grid[[hs]] <- c(m)
      xgboost_grid_params <- list.append(
        xgboost_grid_params,
        c(
          "booster" = model@allparameters$booster,
          "col_sample_rate" = model@allparameters$col_sample_rate,
          "col_sample_rate_per_tree" = model@allparameters$col_sample_rate_per_tree,
          "max_depth" = model@allparameters$max_depth,
          "min_rows" = model@allparameters$min_rows,
          "reg_alpha" = model@allparameters$reg_alpha,
          "reg_lambda" = model@allparameters$reg_lambda,
          "sample_rate" = model@allparameters$sample_rate
        )
      )
    }
  }
  h2o.removeAll()
}

dl_grid_params <- as.data.frame(t(data.frame(dl_grid_params)))
rownames(dl_grid_params) <- paste("Neural network grid model", 1:nrow(dl_grid_params))
```

```
gbm_grid_params <- as.data.frame(t(data.frame(gbm_grid_params)))
rownames(gbm_grid_params) <- paste("GBM grid model", 1:nrow(gbm_grid_params))

xgboost_grid_params <- as.data.frame(t(data.frame(xgboost_grid_params)))
rownames(xgboost_grid_params) <- paste("XGBoost grid model", 1:nrow(xgboost_grid_params))

write.csv(dl_grid_params, "./other_data/toxicity_adversarial/neural_network_grid_params.csv")
write.csv(gbm_grid_params, "./other_data/toxicity_adversarial/gbm_grid_params.csv")
write.csv(xgboost_grid_params, "./other_data/toxicity_adversarial/xgboost_grid_params.csv")

save(dl_grid, gbm_grid, xgboost_grid, file = "./rdata/model_training_grid_models_tx_ad.RData")
```

5.4.3 Model evaluation

```
model_evaluation <- function(holdout_pred, fold_asign, grid_name, grid_meta = NULL) {
  res_ <- list()
  model_num <- c()
  if (startsWith(grid_name, "Super learner")) {
    if (grid_name == "Super learner all models") {
      m <- colnames(holdout_pred)[startsWith(colnames(holdout_pred),
↪  "metalearner_glm_superlearner_iter_0")]
    } else if (grid_name == "Super learner final") {
      sl_models <- colnames(holdout_pred)[startsWith(colnames(holdout_pred),
↪  "metalearner_glm_superlearner_iter")]
      m <- sl_models[length(sl_models)]
    } else if (grid_name == "Super learner neural network") {
      m <- colnames(holdout_pred)[startsWith(colnames(holdout_pred),
↪  "metalearner_glm_superlearner_deeplearning")]
    } else if (grid_name == "Super learner GBM") {
      m <- colnames(holdout_pred)[startsWith(colnames(holdout_pred),
↪  "metalearner_glm_superlearner_gbm")]
    } else if (grid_name == "Super learner XGBoost") {
      m <- colnames(holdout_pred)[startsWith(colnames(holdout_pred),
↪  "metalearner_glm_superlearner_xgboost")]
    }
    logloss <- c()
    MCC <- c()
    F1 <- c()
    acc <- c()
```

```
    EF <- c()
    BEDROC <- c()
    threshold <- 0.5
    for (k in 1:10) {
      y_true <- holdout_pred[fold_assign == k, "category"]
      y_pred <- holdout_pred[fold_assign == k, m]
      y_pred_cls <- sapply(y_pred, function(x) if (x >= threshold) 1 else 0)
      logloss <- c(logloss, LogLoss(y_pred = y_pred, y_true = y_true))
      MCC <- c(MCC, mcc(preds = y_pred_cls, actuals = y_true))
      F1 <- c(F1, F1_Score(y_pred = y_pred_cls, y_true = y_true))
      acc <- c(acc, balanced_accuracy(y_pred = y_pred_cls, y_true = y_true))
      EF <- c(EF, enrichment_factor(x = y_pred, y = y_true, top = 0.05))
      BEDROC <- c(BEDROC, bedroc(x = y_pred, y = y_true, alpha = 20))
    }
    res_ <- list.append(res_, c(
      mean(logloss, na.rm = TRUE), sem(logloss),
      mean(MCC, na.rm = TRUE), sem(MCC),
      mean(F1, na.rm = TRUE), sem(F1),
      mean(acc, na.rm = TRUE), sem(acc),
      mean(EF, na.rm = TRUE), sem(EF),
      mean(BEDROC, na.rm = TRUE), sem(BEDROC),
      logloss, MCC, F1, acc, EF, BEDROC
    ))
  } else {
    for (j in 1:length(grid_meta)) {
      g <- grid_meta[[j]]
      m <- intersect(colnames(holdout_pred), g)
      if (length(m) == 1) {
        model_num <- c(model_num, j)
        logloss <- c()
        MCC <- c()
        F1 <- c()
        acc <- c()
        EF <- c()
        BEDROC <- c()
        threshold <- 0.5
        for (k in 1:10) {
          y_true <- holdout_pred[fold_assign == k, "category"]
          y_pred <- holdout_pred[fold_assign == k, m]
          y_pred_cls <- sapply(y_pred, function(x) if (x >= threshold) 1 else 0)
          logloss <- c(logloss, LogLoss(y_pred = y_pred, y_true = y_true))
```

```
          MCC <- c(MCC, mcc(preds = y_pred_cls, actuals = y_true))
          F1 <- c(F1, F1_Score(y_pred = y_pred_cls, y_true = y_true))
          acc <- c(acc, balanced_accuracy(y_pred = y_pred_cls, y_true = y_true))
          EF <- c(EF, enrichment_factor(x = y_pred, y = y_true, top = 0.05))
          BEDROC <- c(BEDROC, bedroc(x = y_pred, y = y_true, alpha = 20))
        }
        res_ <- list.append(res_, c(
          mean(logloss, na.rm = TRUE), sem(logloss),
          mean(MCC, na.rm = TRUE), sem(MCC),
          mean(F1, na.rm = TRUE), sem(F1),
          mean(acc, na.rm = TRUE), sem(acc),
          mean(EF, na.rm = TRUE), sem(EF),
          mean(BEDROC, na.rm = TRUE), sem(BEDROC),
          logloss, MCC, F1, acc, EF, BEDROC
        ))
      }
    }
  }
  res <- as.data.frame(t(data.frame(res_)))
  colnames(res) <- c(
    "Log loss mean", "Log loss s.e.m.", "MCC mean", "MCC s.e.m.", "F_1 mean", "F_1 s.e.m.",
    "Balanced accuracy mean", "Balanced accuracy s.e.m.", "EF mean", "EF s.e.m.", "BEDROC mean",
    ↪  "BEDROC s.e.m.",
    paste("Log loss CV", 1:10), paste("MCC CV", 1:10), paste("F_1 CV", 1:10),
    paste("Balanced accuracy CV", 1:10), paste("EF CV", 1:10), paste("BEDROC CV", 1:10)
  )
  if (nrow(res) == 1) {
    rownames(res) <- grid_name
  } else {
    rownames(res) <- paste(grid_name, model_num)
  }
  return(res)
}
```

5.4.4 Inner loop model selection

```
load(file = "./rdata/model_training_grid_models_tx_ad.RData")

for (i in 1:10) {
```

```
  holdout_pred <- read.csv(paste0("./other_data/toxicity_adversarial/outer_", i,
↪  "/cv_holdout_predictions.csv"))
  fold_assign <- rep(1:10, ceiling(nrow(holdout_pred) / 10))[1:nrow(holdout_pred)]
  data_ <- list()
  # Deep learning grid models
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
    grid_name = "Neural network grid model",
    grid_meta = dl_grid
  ))
  # GBM grid models
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
    grid_name = "GBM grid model",
    grid_meta = gbm_grid
  ))
  # GBM default models
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
    grid_name = "GBM model",
    grid_meta = as.list(paste0("GBM_", 1:5))
  ))
  # XGBoost grid models
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
    grid_name = "XGBoost grid model",
    grid_meta = xgboost_grid
  ))
  # XGBoost default models
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign,
    grid_name = "XGBoost model",
    grid_meta = as.list(paste0("XGBoost_", 1:3))
  ))
  # GLM
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "GLM model",
↪  grid_meta = list("GLM")))
  # DRF
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "DRF model",
↪  grid_meta = list("DRF")))
  # XRT
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "XRT model",
↪  grid_meta = list("XRT")))
  # Super learner all
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪  all models"))
```

```
  # Super learner final
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪  final"))
  # Super learner deep learning
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪  neural network"))
  # Super learner GBM
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪  GBM"))
  # Super learner XGBoost
  data_ <- list.append(data_, model_evaluation(holdout_pred, fold_assign, grid_name = "Super learner
↪  XGBoost"))

  data <- do.call(rbind, data_)
  write.csv(data, paste0("./other_data/toxicity_adversarial/outer_", i, "/cv_res.csv"))
}
```

5.4.4.1 CV 1

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
GBM grid model 83	0.9214	0.5238		0.6806	14	67	217	5	303

Showing 1 to 1 of 141 entries Previous 1 2 3 4 5 … 141 Next

5.4.4.2 CV 2

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
GBM grid model 95	0.8926	0.1085		1	46	17	201	4	268

Showing 1 to 1 of 126 entries Previous 1 2 3 4 5 … 126 Next

5.4.4.3 CV 3

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
GBM grid model 48	0.7521	0.1078		0.2377	46	55	200	7	308

Showing 1 to 1 of 118 entries Previous 1 2 3 4 5 … 118 Next

5.4.4.4 CV 4

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
GBM grid model 83	0.8041	0.111		0.6898	17	99.5	235	6.5	358

Showing 1 to 1 of 130 entries Previous 1 2 3 4 5 … 130 Next

5.4.4.5 CV 5

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
XGBoost grid model 84	0.0004		0.0155	0.0002	194	1	43	68	306

Showing 1 to 1 of 80 entries Previous 1 2 3 4 5 … 80 Next

5.4.4.6 CV 6

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
GBM grid model 21	0.2346	0.1862		0.0045	33	3	255	26	317

Showing 1 to 1 of 78 entries Previous 1 2 3 4 5 … 78 Next

5.4.4.7 CV 7

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
GBM grid model 23	0.7728	0.0993		1	53	32	238	5	328

Showing 1 to 1 of 131 entries Previous 1 2 3 4 5 … 131 Next

5.4.4.8 CV 8

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
GBM grid model 55	0.7623	0.0269		1	38	61	211	3	313

Showing 1 to 1 of 131 entries Previous 1 2 3 4 5 … 131 Next

5.4.4.9 CV 9

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
GBM grid model 71	0.384	0.2049		0.9698	102	24	221	7	354

Showing 1 to 1 of 136 entries Previous 1 2 3 4 5 … 136 Next

5.4.4.10 CV 10

Show 1 entries Search:

	Log loss adj. *p* value	MCC adj. *p* value	F_1 adj. *p* value	Balanced accuracy adj. *p* value	Log loss rank	MCC rank	F_1 rank	Balanced accuracy rank	Rank sum
GBM grid model 11	1	0.112		0.9728	4	97	233	3	337

Showing 1 to 1 of 127 entries Previous 1 2 3 4 5 … 127 Next

5.4.5 Final evaluation

```
load(file = "./rdata/var_reduct_tx_train_test_splits_y_shuffled.RData")
load(file = "./rdata/model_training_grid_models_tx_ad.RData")
dir <- "/Users/renee/Downloads/toxicity_adversarial"
selected_models <- c(
  "GBM grid model 52", "GBM grid model 95", "GBM grid model 48",
  "GBM grid model 98", "GBM grid model 55", "GBM grid model 21",
  "GBM grid model 23", "GBM grid model 54", "GBM grid model 3",
  "GBM grid model 68"
)
logloss <- c()
MCC <- c()
F1 <- c()
acc <- c()
EF <- c()
BEDROC <- c()
threshold <- 0.5
for (i in 1:10) {
  holdout_pred <- read.csv(paste0("./other_data/toxicity_adversarial/outer_", i,
↪  "/cv_holdout_predictions.csv"))
  if (startsWith(selected_models[i], "Neural network grid model")) {
    grid_num <- as.integer(str_extract(selected_models[i], "[0-9]+"))
    m <- intersect(colnames(holdout_pred), dl_grid[[grid_num]])
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/", m))
  } else if (startsWith(selected_models[i], "GBM grid model")) {
    grid_num <- as.integer(str_extract(selected_models[i], "[0-9]+"))
    m <- intersect(colnames(holdout_pred), gbm_grid[[grid_num]])
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/", m))
  } else if (startsWith(selected_models[i], "GBM model")) {
    grid_num <- as.integer(str_extract(selected_models[i], "[0-9]+"))
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/GBM_", grid_num))
  } else if (startsWith(selected_models[i], "XGBoost grid model")) {
    grid_num <- as.integer(str_extract(selected_models[i], "[0-9]+"))
    m <- intersect(colnames(holdout_pred), xgboost_grid[[grid_num]])
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/", m))
  } else if (startsWith(selected_models[i], "XGBoost model")) {
    grid_num <- as.integer(str_extract(selected_models[i], "[0-9]+"))
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/XGBoost_", grid_num))
  } else if (selected_models[i] == "Super learner all models") {
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/superlearner_iter_0"))
```

```
  } else if (selected_models[i] == "Super learner final") {
    files <- list.files(paste0(dir, "/outer_", i))
    files <- files[startsWith(files, "superlearner_iter")]
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/", files[length(files)]))
  } else if (selected_models[i] == "Super learner neural network") {
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/superlearner_deeplearning"))
  } else if (selected_models[i] == "Super learner GBM") {
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/superlearner_gbm"))
  } else if (selected_models[i] == "Super learner XGBoost") {
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/superlearner_xgboost"))
  } else {
    model <- h2o.loadModel(paste0(dir, "/outer_", i, "/", unlist(strsplit(selected_models[i], "
↪  "))[1]))
  }
  tmp <- as.h2o(subset(outer_splits[[i]][["test"]], select = -category))
  y_true <- as.numeric(outer_splits[[i]][["test"]]$category) - 1
  y_pred <- as.data.frame(as.data.frame(h2o.predict(model, tmp)))[, 3]
  y_pred_cls <- sapply(y_pred, function(x) if (x >= threshold) 1 else 0)
  logloss <- c(logloss, LogLoss(y_pred = y_pred, y_true = y_true))
  MCC <- c(MCC, mcc(preds = y_pred_cls, actuals = y_true))
  F1 <- c(F1, F1_Score(y_pred = y_pred_cls, y_true = y_true))
  acc <- c(acc, balanced_accuracy(y_pred = y_pred_cls, y_true = y_true))
  EF <- c(EF, enrichment_factor(x = y_pred, y = y_true, top = 0.05))
  BEDROC <- c(BEDROC, bedroc(x = y_pred, y = y_true, alpha = 20))
}
save(logloss, MCC, F1, acc, EF, BEDROC, file = "./rdata/final_evaluation_tx_ad.RData")
```

```
load(file = "./rdata/final_evaluation_tx_ad.RData")
data.frame(
  Metric = c("Log loss", "MCC", "F<sub>1</sub>", "Balanced accuracy", "EF", "BEDROC"),
  `Mean ± s.e.m.` = c(
    paste0(round(mean(logloss), 3), " ± ", round(sem(logloss), 3)),
    paste0(round(mean(MCC), 3), " ± ", round(sem(MCC), 3)),
    paste0(round(mean(F1, na.rm = TRUE), 3), " ± ", round(sem(F1), 3)),
    paste0(round(mean(acc), 3), " ± ", round(sem(acc), 3)),
    paste0(round(mean(EF), 3), " ± ", round(sem(EF), 3)),
    paste0(round(mean(BEDROC), 3), " ± ", round(sem(BEDROC), 3))
  ),
  check.names = FALSE
```

```
) %>%
  datatable(escape = FALSE, rownames = FALSE)
```

Show 10 entries Search:

Metric	Mean ± s.e.m.
Log loss	0.654 ± 0.005
MCC	0.001 ± 0.012
F_1	0.047 ± 0.023
Balanced accuracy	0.619 ± 0.035
EF	0.956 ± 0.057
BEDROC	0.62 ± 0.023

Showing 1 to 6 of 6 entries Previous 1 Next

5.5 Overall model interpretation

```
import h2o
import pandas as pd
import numpy as np
import shap
import matplotlib.pyplot as plt
import session_info
```

5.5.1 Melanin binding (regression)

```
# Save train and test sets in csv format
load(file = "./rdata/var_reduct_mb_train_test_splits_y_shuffled.RData")
for (i in 1:10) {
  prefix <- paste0("outer_", i)
  exp_dir <- paste0("/Users/renee/Downloads/melanin_binding_adversarial/", prefix)
  write.csv(outer_splits[[i]]$train, file = paste0(exp_dir, "/train_set.csv"))
  write.csv(outer_splits[[i]]$test, file = paste0(exp_dir, "/test_set.csv"))
}
```

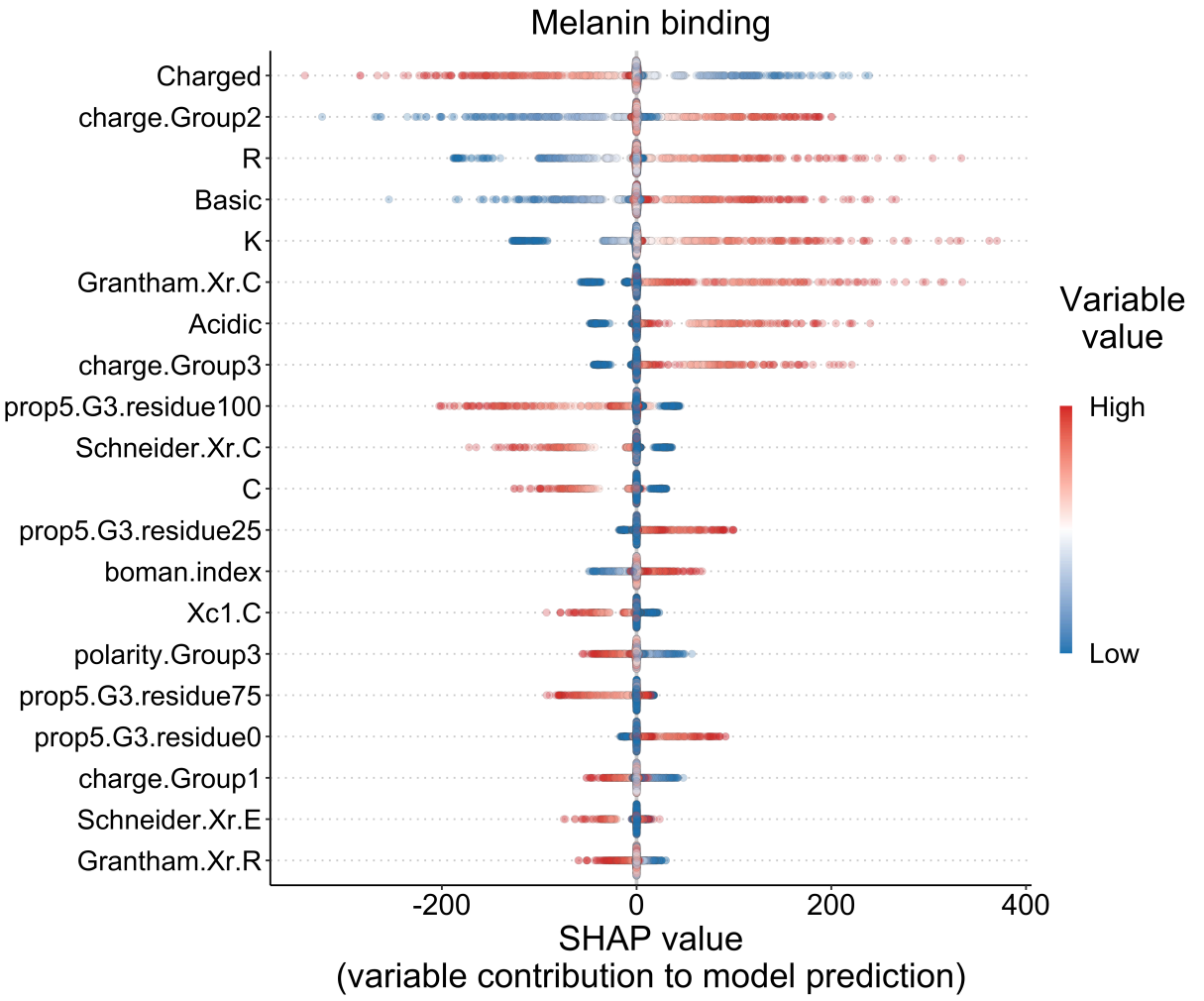
```
h2o.init(nthreads=-1)

dir_path = '/Users/renee/Downloads/melanin_binding_adversarial'
model_names = ['superlearner_iter_6', 'superlearner_iter_5', 'superlearner_iter_6',
↪  'superlearner_iter_6',
               'superlearner_iter_7', 'superlearner_iter_7', 'superlearner_iter_6',
                ↪  'superlearner_iter_7',
               'superlearner_iter_7', 'superlearner_iter_6']
shap_res = []
for i in range(10):
    print('Iter ' + str(i + 1) + ':')
    train = pd.read_csv(dir_path + '/outer_' + str(i + 1) + '/train_set.csv', index_col=0)
    X_train = train.iloc[:, :-1]
    test = pd.read_csv(dir_path + '/outer_' + str(i + 1) + '/test_set.csv', index_col=0)
    X_test = test.iloc[:, :-1]
    model = h2o.load_model(dir_path + '/outer_' + str(i + 1) + '/' + model_names[i])
    def model_predict_(data):
        data = pd.DataFrame(data, columns=X_train.columns)
        h2o_data = h2o.H2OFrame(data)
        res = model.predict(h2o_data)
        res = res.as_data_frame()
        return(res)
    np.random.seed(1)
    X_train_ = X_train.iloc[np.random.choice(X_train.shape[0], 100, replace=False), :]
    explainer = shap.KernelExplainer(model_predict_, X_train_, link='identity')
    shap_values = explainer.shap_values(X_test, nsamples=1000)
    shap_res.append(shap_values)
    h2o.remove_all()

shap_res = pd.DataFrame(np.vstack(shap_res), columns=X_train.columns)
shap_res.to_csv('./other_data/mb_ad_shap_values_cv_data_sets.csv', index=False)
```

```
load(file = "./rdata/var_reduct_mb_train_test_splits_y_shuffled.RData")
data <- do.call(rbind, lapply(outer_splits, function(x) x$test[-ncol(mb_data)]))
data <- apply(data, 2, function(x) rank(x))
data <- apply(data, 2, function(x) rescale(x, to = c(0, 100)))
shap_values <- as.matrix(read.csv("./other_data/mb_ad_shap_values_cv_data_sets.csv", check.names =
↪  FALSE))
shap_values <- as.data.frame(rescale(shap_values, to = c(0, 100), from = range(mb_data$log_intensity)))
```

```r
var_diff <- apply(shap_values, 2, function(x) max(x) - min(x))
var_diff <- var_diff[order(-var_diff)]
df <- data.frame(
  variable = melt(shap_values)$variable,
  shap = melt(shap_values)$value,
  variable_value = melt(data)$value
)
```

```r
top_vars <- names(var_diff)[1:20]
df <- df[df$variable %in% top_vars, ]
df$variable <- factor(df$variable, levels = rev(top_vars), labels = rev(top_vars))

p1 <- ggplot(df, aes(x = shap, y = variable)) +
  geom_hline(yintercept = top_vars, linetype = "dotted", color = "grey80") +
  geom_vline(xintercept = 0, color = "grey80", size = 1) +
  geom_point(aes(fill = variable_value), color = "grey30", shape = 21, alpha = 0.3, size = 2, position
  ↪  = "auto", stroke = 0.1) +
  scale_fill_gradient2(low = "#1f77b4", mid = "white", high = "#d62728", midpoint = 50, breaks = c(0,
  ↪  100), labels = c("Low", "High")) +
  theme_bw() +
  theme(
    plot.margin = ggplot2::margin(1.5, 8, 1.5, -3),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    strip.background = element_blank(),
    panel.border = element_blank(),
    axis.line = element_line(color = "black"),
    plot.title = element_text(hjust = 0.5, size = 20),
    axis.title.x = element_text(hjust = 0.5, colour = "black", size = 20),
    axis.title.y = element_text(colour = "black", size = 20),
    axis.text.x = element_text(colour = "black", size = 17),
    axis.text.y = element_text(colour = "black", size = 16),
    legend.title = element_text(size = 20),
    legend.text = element_text(colour = "black", size = 16),
    legend.title.align = 0.5
  ) +
  guides(
    fill = guide_colourbar("Variable\nvalue\n", ticks = FALSE, barheight = 10, barwidth = 0.5),
```

```
  color = "none"
) +
xlab("SHAP value\n(variable contribution to model prediction)") +
ylab("") +
ggtitle("Melanin binding")
```



5.5.2 Cell-penetration (classification)

```
# Save train and test sets in csv format
load(file = "./rdata/var_reduct_cpp_train_test_splits_y_shuffled.RData")
for (i in 1:10) {
```

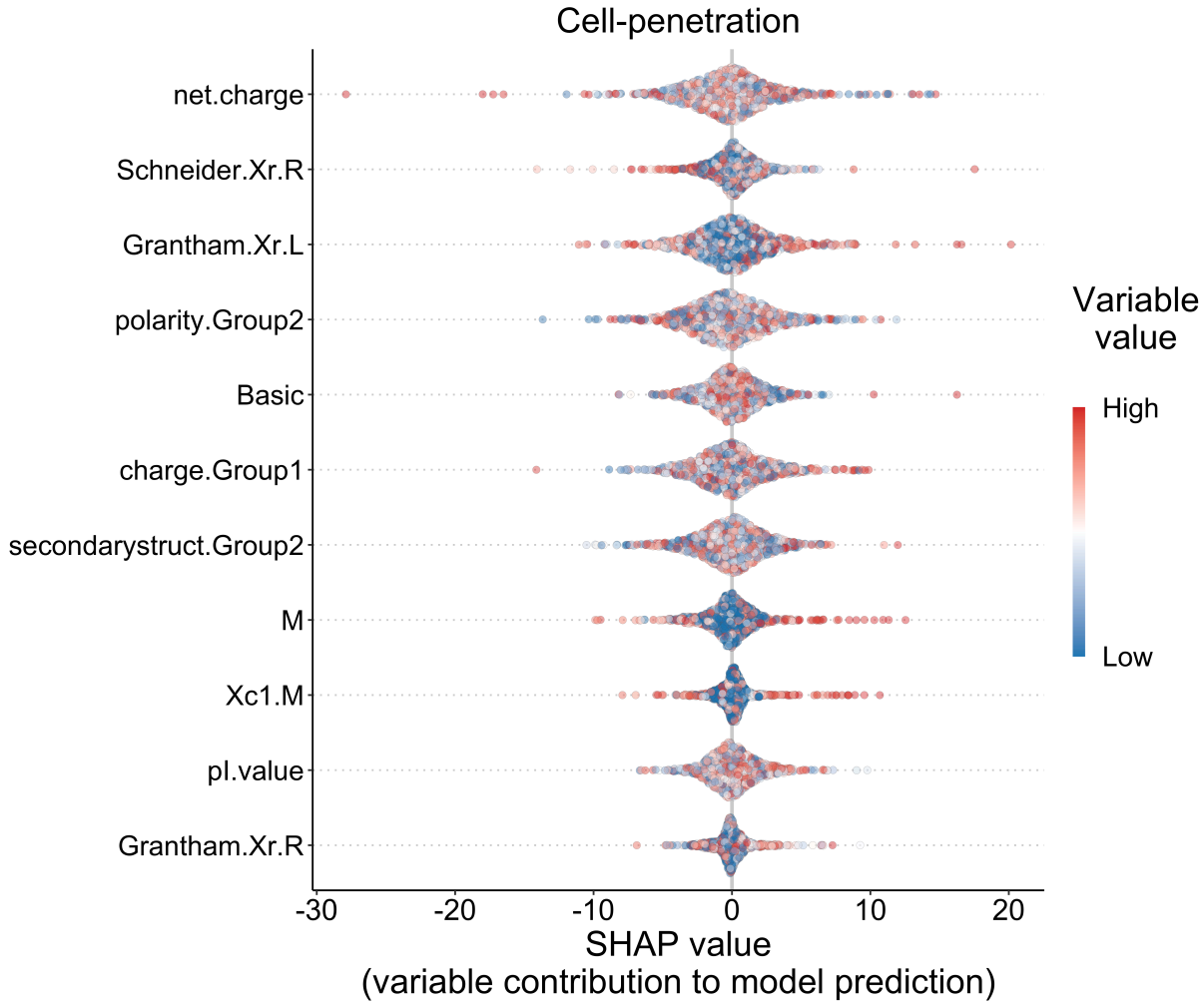
```
  prefix <- paste0("outer_", i)
  exp_dir <- paste0("/Users/renee/Downloads/cell_penetration_adversarial/", prefix)
  write.csv(outer_splits[[i]]$train, file = paste0(exp_dir, "/train_set.csv"))
  write.csv(outer_splits[[i]]$test, file = paste0(exp_dir, "/test_set.csv"))
}
```

```
h2o.init(nthreads=-1)

dir_path = '/Users/renee/Downloads/cell_penetration_adversarial'
model_names = ['GBM_model_1671047404901_26693', 'GBM_model_1671047404901_85146',
↪  'GBM_model_1671047404901_143415',
               'GBM_model_1671047404901_201211', 'GBM_model_1671047404901_258107',
                ↪  'GBM_model_1671047404901_315866',
               'GBM_model_1671047404901_373347', 'GBM_model_1671047404901_432044',
                ↪  'GBM_model_1671122685293_29338',
               'GBM_model_1671122685293_86356']
shap_res = []
for i in range(10):
    print('Iter ' + str(i + 1) + ':')
    train = pd.read_csv(dir_path + '/outer_' + str(i + 1) + '/train_set.csv', index_col=0)
    X_train = train.iloc[:, :-1]
    test = pd.read_csv(dir_path + '/outer_' + str(i + 1) + '/test_set.csv', index_col=0)
    X_test = test.iloc[:, :-1]
    model = h2o.load_model(dir_path + '/outer_' + str(i + 1) + '/' + model_names[i])
    def model_predict_(data):
        data = pd.DataFrame(data, columns=X_train.columns)
        h2o_data = h2o.H2OFrame(data)
        res = model.predict(h2o_data)
        res = res.as_data_frame().iloc[:,2]
        return(res)
    np.random.seed(1)
    X_train_ = X_train.iloc[np.random.choice(X_train.shape[0], 100, replace=False), :]
    explainer = shap.KernelExplainer(model_predict_, X_train_, link='identity')
    shap_values = explainer.shap_values(X_test, nsamples=1000)
    shap_res.append(shap_values)
    h2o.remove_all()

shap_res = pd.DataFrame(np.vstack(shap_res), columns=X_train.columns)
shap_res.to_csv('./other_data/cpp_ad_shap_values_cv_data_sets.csv', index=False)
```

```r
load(file = "./rdata/var_reduct_cpp_train_test_splits_y_shuffled.RData")
data <- do.call(rbind, lapply(outer_splits, function(x) x$test[-ncol(cpp_data)]))
data <- apply(data, 2, function(x) rank(x))
data <- apply(data, 2, function(x) rescale(x, to = c(0, 100)))
shap_values <- read.csv("./other_data/cpp_ad_shap_values_cv_data_sets.csv", check.names = FALSE) * 100

var_diff <- apply(shap_values, 2, function(x) max(x) - min(x))
var_diff <- var_diff[order(-var_diff)]
df <- data.frame(
  variable = melt(shap_values)$variable,
  shap = melt(shap_values)$value,
  variable_value = melt(data)$value
)
```

```r
top_vars <- names(var_diff)[1:11]
df <- df[df$variable %in% top_vars, ]
df$variable <- factor(df$variable, levels = rev(top_vars), labels = rev(top_vars))

p2 <- ggplot(df, aes(x = shap, y = variable)) +
  geom_hline(yintercept = top_vars, linetype = "dotted", color = "grey80") +
  geom_vline(xintercept = 0, color = "grey80", size = 1) +
  geom_point(aes(fill = variable_value), color = "grey30", shape = 21, alpha = 0.5, size = 2, position
  ↪  = "auto", stroke = 0.1) +
  scale_fill_gradient2(low = "#1f77b4", mid = "white", high = "#d62728", midpoint = 50, breaks = c(0,
  ↪  100), labels = c("Low", "High")) +
  theme_bw() +
  theme(
    plot.margin = ggplot2::margin(1.5, 8, 1.5, -3),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    strip.background = element_blank(),
    panel.border = element_blank(),
    axis.line = element_line(color = "black"),
    plot.title = element_text(hjust = 0.5, size = 20),
    axis.title.x = element_text(hjust = 0.5, colour = "black", size = 20),
    axis.title.y = element_text(colour = "black", size = 20),
    axis.text.x = element_text(colour = "black", size = 17),
    axis.text.y = element_text(colour = "black", size = 16),
    legend.title = element_text(size = 20),
```

```
  legend.text = element_text(colour = "black", size = 16),
  legend.title.align = 0.5
) +
guides(
  fill = guide_colourbar("Variable\nvalue\n", ticks = FALSE, barheight = 10, barwidth = 0.5),
  color = "none"
) +
xlab("SHAP value\n(variable contribution to model prediction)") +
ylab("") +
ggtitle("Cell-penetration")
```



5.5.3 Toxicity (classification)

```
# Save train and test sets in csv format
load(file = "./rdata/var_reduct_tx_train_test_splits_y_shuffled.RData")
for (i in 1:10) {
  prefix <- paste0("outer_", i)
  exp_dir <- paste0("/Users/renee/Downloads/toxicity_adversarial/", prefix)
  write.csv(outer_splits[[i]]$train, file = paste0(exp_dir, "/train_set.csv"))
  write.csv(outer_splits[[i]]$test, file = paste0(exp_dir, "/test_set.csv"))
}
```

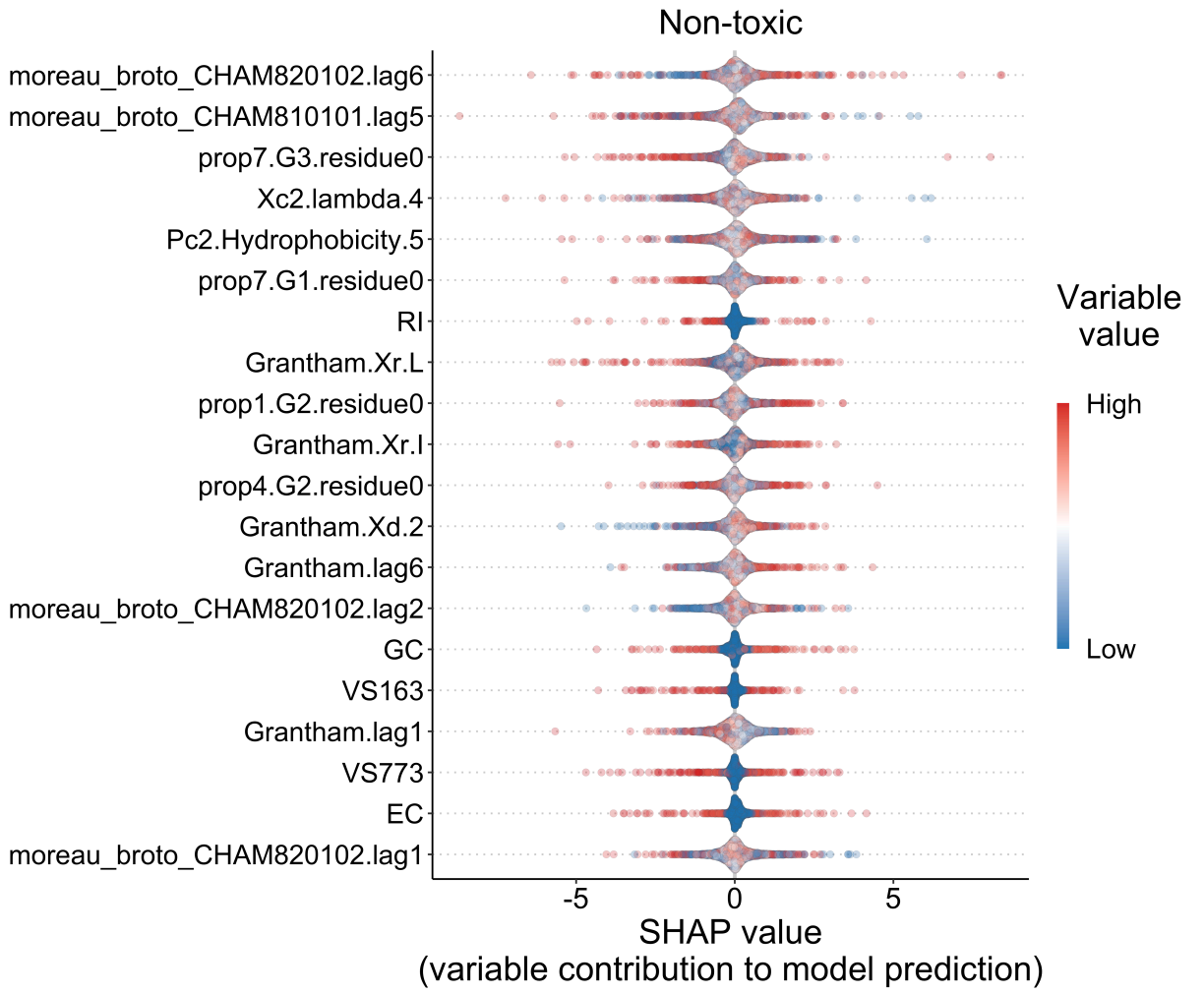
```
h2o.init(nthreads=-1)

dir_path = '/Users/renee/Downloads/toxicity_adversarial'
model_names = ['superlearner_iter_5', 'superlearner_iter_6', 'superlearner_iter_5',
↪  'superlearner_iter_7',
               'superlearner_iter_4', 'superlearner_iter_6', 'superlearner_iter_6',
                ↪  'superlearner_iter_5',
               'superlearner_iter_4', 'superlearner_iter_5']
shap_res = []
for i in range(10):
    print('Iter ' + str(i + 1) + ':')
    train = pd.read_csv(dir_path + '/outer_' + str(i + 1) + '/train_set.csv', index_col=0)
    X_train = train.iloc[:, :-1]
    test = pd.read_csv(dir_path + '/outer_' + str(i + 1) + '/test_set.csv', index_col=0)
    X_test = test.iloc[:, :-1]
    model = h2o.load_model(dir_path + '/outer_' + str(i + 1) + '/' + model_names[i])
    def model_predict_(data):
        data = pd.DataFrame(data, columns=X_train.columns)
        h2o_data = h2o.H2OFrame(data)
        res = model.predict(h2o_data)
        res = res.as_data_frame().iloc[:,2] # 0:toxic; 1:non-toxic
        return(res)
    np.random.seed(1)
    X_train_ = X_train.iloc[np.random.choice(X_train.shape[0], 100, replace=False), :]
    explainer = shap.KernelExplainer(model_predict_, X_train_, link='identity')
    shap_values = explainer.shap_values(X_test, nsamples=1000)
    shap_res.append(shap_values)
    h2o.remove_all()
```

```
shap_res = pd.DataFrame(np.vstack(shap_res), columns=X_train.columns)
shap_res.to_csv('./other_data/tx_ad_shap_values_cv_data_sets.csv', index=False)
```

```
load(file = "./rdata/var_reduct_tx_train_test_splits_y_shuffled.RData")
data <- do.call(rbind, lapply(outer_splits, function(x) x$test[-ncol(tx_data)]))
data <- apply(data, 2, function(x) rank(x))
data <- apply(data, 2, function(x) rescale(x, to = c(0, 100)))
shap_values <- read.csv("./other_data/tx_ad_shap_values_cv_data_sets.csv", check.names = FALSE) * 100

set.seed(12)
ind <- sample(1:nrow(data), 2500)
data <- data[ind, ]
shap_values <- shap_values[ind, ]

var_diff <- apply(shap_values, 2, function(x) max(x) - min(x))
var_diff <- var_diff[order(-var_diff)]
df <- data.frame(
  variable = melt(shap_values)$variable,
  shap = melt(shap_values)$value,
  variable_value = melt(data)$value
)
```

```
top_vars <- names(var_diff)[1:20]
df <- df[df$variable %in% top_vars, ]
df$variable <- factor(df$variable, levels = rev(top_vars), labels = rev(top_vars))

p3 <- ggplot(df, aes(x = shap, y = variable)) +
  geom_hline(yintercept = top_vars, linetype = "dotted", color = "grey80") +
  geom_vline(xintercept = 0, color = "grey80", size = 1) +
  geom_point(aes(fill = variable_value), color = "grey30", shape = 21, alpha = 0.3, size = 2, position
  ↪  = "auto", stroke = 0.1) +
  scale_fill_gradient2(low = "#1f77b4", mid = "white", high = "#d62728", midpoint = 50, breaks = c(0,
  ↪  100), labels = c("Low", "High")) +
  theme_bw() +
  theme(
    plot.margin = ggplot2::margin(1.5, 8, 1.5, -3),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
```

```
  strip.background = element_blank(),
  panel.border = element_blank(),
  axis.line = element_line(color = "black"),
  plot.title = element_text(hjust = 0.5, size = 20),
  axis.title.x = element_text(hjust = 0.5, colour = "black", size = 20),
  axis.title.y = element_text(colour = "black", size = 20),
  axis.text.x = element_text(colour = "black", size = 17),
  axis.text.y = element_text(colour = "black", size = 16),
  legend.title = element_text(size = 20),
  legend.text = element_text(colour = "black", size = 16),
  legend.title.align = 0.5
) +
guides(
  fill = guide_colourbar("Variable\nvalue\n", ticks = FALSE, barheight = 10, barwidth = 0.5),
  color = "none"
) +
xlab("SHAP value\n(variable contribution to model prediction)") +
ylab("") +
ggtitle("Non-toxic")
```




```
sessionInfo()
```

```
## R version 4.2.2 (2022-10-31)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Big Sur ... 10.16
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
```

```
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] reticulate_1.25 ggforce_0.3.4   ggplot2_3.3.6   reshape2_1.4.4
##  [5] DT_0.24         digest_0.6.29   stringr_1.4.1   rlist_0.4.6.2
##  [9] enrichvs_0.0.5  scales_1.2.1    mltools_0.3.5   MLmetrics_1.1.1
## [13] h2o_3.38.0.2
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.9        here_1.0.1        lattice_0.20-45   ps_1.7.1
##  [5] png_0.1-7         rprojroot_2.0.3   assertthat_0.2.1  utf8_1.2.2
##  [9] R6_2.5.1          plyr_1.8.7        evaluate_0.16     highr_0.9
## [13] pillar_1.8.1      rlang_1.0.4       rstudioapi_0.14   data.table_1.14.2
## [17] callr_3.7.2       jquerylib_0.1.4   R.utils_2.12.0    R.oo_1.25.0
## [21] Matrix_1.5-1      rmarkdown_2.16    styler_1.8.0      webshot_0.5.4
## [25] htmlwidgets_1.5.4 RCurl_1.98-1.8    polyclip_1.10-0   munsell_0.5.0
## [29] compiler_4.2.2    xfun_0.32         pkgconfig_2.0.3   htmltools_0.5.3
## [33] tidyselect_1.1.2  tibble_3.1.8      bookdown_0.28     codetools_0.2-18
## [37] fansi_1.0.3       dplyr_1.0.9       withr_2.5.0       MASS_7.3-58.1
## [41] bitops_1.0-7      R.methodsS3_1.8.2 grid_4.2.2        jsonlite_1.8.0
## [45] gtable_0.3.0      lifecycle_1.0.1   DBI_1.1.3         magrittr_2.0.3
## [49] cachem_1.0.6      cli_3.3.0         stringi_1.7.8     farver_2.1.1
## [53] bslib_0.4.0       generics_0.1.3    vctrs_0.4.1       tools_4.2.2
## [57] R.cache_0.16.0    glue_1.6.2        tweenr_2.0.1      purrr_0.3.4
## [61] crosstalk_1.2.0   processx_3.7.0    fastmap_1.1.0     yaml_2.3.5
## [65] colorspace_2.0-3  knitr_1.40        sass_0.4.2
```

```
session_info.show()
```

```
## -----
## h2o                 3.38.0.2
## matplotlib          3.6.2
## numpy               1.23.5
## pandas              1.5.2
## session_info        1.0.0
## shap                0.41.0
## -----
## Python 3.8.15 | packaged by conda-forge | (default, Nov 22 2022, 09:04:40) [Clang 14.0.6 ]
## macOS-10.16-x86_64-i386-64bit
## -----
## Session information updated at 2023-03-12 23:20
```

Section 6

Small data set demo

6.1 Setup

6.1.1 Load libraries

```
library(ranger)
library(rlist)
library(h2o)
h2o.init(nthreads = -1)
```

6.1.2 AIC functions

```
#' Mean squared error function
#'
#' @param y_true a factor vector representing true values.
#' @param y_pred a numeric vector or a matrix of predicted values.
#'
#' @return MSE
#'
mse <- function(y_true, y_pred) {
  return(mean((y_true - y_pred)^2))
}

#' Regression AIC
#'
#' @param y_true a factor vector representing true values.
#' @param y_pred a numeric vector or a matrix of predicted values.
#' @param k number of parameters/variables.
#' @param eps a very small value to avoid negative infinitives generated by log(p=0).
#'
#' @return AIC
#'
```

```r
aic_reg <- function(y_true, y_pred, k, eps = 1e-15) {
  mserr <- mse(y_true, y_pred)
  if (mserr == 0) mserr <- eps
  AIC <- length(y_true) * log(mserr) + 2 * k
  return(AIC)
}

#' Cross-entropy function
#'
#' @param y_true a factor vector representing true labels.
#' @param y_pred a numeric vector (probabilities of the second class/factor level)
#' or a matrix of predicted probabilities.
#' @param eps a very small value to avoid negative infinitives generated by log(p=0).
#' If the function returns NaN, then increasing eps may solve the issue. Alternatively,
#' As NaN is usually generated in the case of p = 1 - eps = 1, resulting in log(1 - p)
#' = log(0) from binary cross-entropy, the user can pass prediction values as a matrix
#' to calculate categorical cross-entropy instead.
#'
#' @return H, cross-entropy (mean loss per sample)
#'
crossEntropy <- function(y_true, y_pred, eps = 1e-15) {
  stopifnot(is.factor(y_true))
  y_pred <- pmax(pmin(y_pred, 1 - eps), eps)
  n_levels <- nlevels(y_true)
  H <- NULL
  # Binary classification
  if (n_levels == 2 && is.vector(y_pred)) {
    y_true <- as.numeric(y_true == levels(y_true)[-1])
    H <- -mean(y_true * log2(y_pred) + (1 - y_true) * log2(1 - y_pred))
  }
  # Multi-class classification
  else if (n_levels == ncol(y_pred)) {
    y_true <- as.numeric(y_true)
    y_true_encoded <- t(sapply(y_true, function(x) {
      tmp <- rep(0, n_levels)
      tmp[x] <- 1
      return(tmp)
    }))
    H <- -mean(sapply(1:nrow(y_true_encoded), function(x) sum(y_true_encoded[x, ] * log2(y_pred[x,
↪  ])))))
  }
```

```
  return(H)
}

#' Classification AIC
#'
#' @param y_true a factor vector representing true labels.
#' @param y_pred a numeric vector or a matrix of predicted probabilities.
#' @param k number of parameters/variables.
#' @param ... other parameters to be passed to `crossEntropy()`
#'
#' @return AIC
#'
aic_clf <- function(y_true, y_pred, k, ...) {
  H <- crossEntropy(y_true, y_pred, ...)
  AIC <- 2 * log(2) * length(y_true) * H + 2 * k
  return(AIC)
}
```

6.1.3 Model parameters

```
# DeepLearning Grid 1
deeplearning_params_1 <- list(
  activation = "RectifierWithDropout",
  epochs = 10000, # early stopping
  epsilon = c(1e-6, 1e-7, 1e-8, 1e-9),
  hidden = list(c(50), c(200), c(500)),
  hidden_dropout_ratios = list(c(0.1), c(0.2), c(0.3), c(0.4), c(0.5)),
  input_dropout_ratio = c(0, 0.05, 0.1, 0.15, 0.2),
  rho = c(0.9, 0.95, 0.99)
)
# DeepLearning Grid 2
deeplearning_params_2 <- list(
  activation = "RectifierWithDropout",
  epochs = 10000, # early stopping
  epsilon = c(1e-6, 1e-7, 1e-8, 1e-9),
  hidden = list(c(50, 50), c(200, 200), c(500, 500)),
  hidden_dropout_ratios = list(
    c(0.1, 0.1), c(0.2, 0.2), c(0.3, 0.3),
    c(0.4, 0.4), c(0.5, 0.5)
```

```
  ),
  input_dropout_ratio = c(0, 0.05, 0.1, 0.15, 0.2),
  rho = c(0.9, 0.95, 0.99)
)
# DeepLearning Grid 3
deeplearning_params_3 <- list(
  activation = "RectifierWithDropout",
  epochs = 10000, # early stopping
  epsilon = c(1e-6, 1e-7, 1e-8, 1e-9),
  hidden = list(c(50, 50, 50), c(200, 200, 200), c(500, 500, 500)),
  hidden_dropout_ratios = list(
    c(0.1, 0.1, 0.1), c(0.2, 0.2, 0.2),
    c(0.3, 0.3, 0.3), c(0.4, 0.4, 0.4),
    c(0.5, 0.5, 0.5)
  ),
  input_dropout_ratio = c(0, 0.05, 0.1, 0.15, 0.2),
  rho = c(0.9, 0.95, 0.99)
)
# GBM
gbm_params <- list(
  col_sample_rate = c(0.4, 0.7, 1.0),
  col_sample_rate_per_tree = c(0.4, 0.7, 1.0),
  learn_rate = 0.1,
  max_depth = c(3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17),
  min_rows = c(1, 5, 10, 15, 30, 100),
  min_split_improvement = c(1e-4, 1e-5),
  ntrees = 10000, # early stopping
  sample_rate = c(0.5, 0.6, 0.7, 0.8, 0.9, 1.0)
)
# XGBoost
xgboost_params <- list(
  booster = c("gbtree", "dart"),
  col_sample_rate = c(0.6, 0.8, 1.0),
  col_sample_rate_per_tree = c(0.7, 0.8, 0.9, 1.0),
  max_depth = c(5, 10, 15, 20),
  min_rows = c(0.01, 0.1, 1.0, 3.0, 5.0, 10.0, 15.0, 20.0),
  ntrees = 10000, # early stopping
  reg_alpha = c(0.001, 0.01, 0.1, 1, 10, 100),
  reg_lambda = c(0.001, 0.01, 0.1, 0.5, 1),
  sample_rate = c(0.6, 0.8, 1.0)
)
```

6.2 Variable reduction

```
variable_reduction <- function(predictor_variables, response_variable, seed = 12) {
  # Calculate initial variable importance
  set.seed(seed)
  # Train random forest using ranger
  ntree <- 100000
  rf <- ranger(x = predictor_variables, y = response_variable, num.trees = ntree, importance =
↪  "permutation", verbose = TRUE, scale.permutation.importance = TRUE, seed = 3, keep.inbag = TRUE)
  var_imp <- ranger::importance(rf)
  var_imp <- var_imp[order(-var_imp)]
  var_imp <- var_imp[var_imp >= 0]
  # Variable reduction iterations
  ntree <- 1000
  var_subset_res <- list()
  for (i in 1:length(var_imp)) {
    cat(paste0("Evaluating ", i, " variable(s) out of ", length(var_imp), " variables\n"))
    predictor_variables_ <- predictor_variables[, colnames(predictor_variables) %in%
↪  names(var_imp)[1:i], drop = FALSE]
    rf_ <- ranger(x = predictor_variables_, y = response_variable, num.trees = ntree, importance =
↪  "none", verbose = TRUE, seed = seed, probability = is.factor(response_variable))
    if (!is.factor(response_variable)) { # Regression
      var_subset_res <- list.append(var_subset_res, list(
        rsq = rf$r.squared,
        aic = aic_reg(
          y_true = response_variable,
          y_pred = rf_$predictions,
          k = i
        )
      ))
    } else { # Classification
      var_subset_res <- list.append(var_subset_res, list(
        acc = 1 - rf$prediction.error,
        aic = aic_clf(
          y_true = response_variable,
          y_pred = rf_$predictions,
          k = i
```

```
        )
      ))
    }
  }
  aic_vec <- unlist(lapply(var_subset_res, function(x) x$aic))
  aic_cutoff <- which.min(aic_vec)
  var_imp <- ranger::importance(rf)
  var_imp <- var_imp[order(-var_imp)][1:aic_cutoff]
  cat(paste("Number of variables after variable reduction: ", aic_cutoff, "\n"))
  data <- predictor_variables[, colnames(predictor_variables) %in% names(var_imp), drop = FALSE]
  data$response <- response_variable
  return(data)
}
```

6.3 Model training

```
model_training <- function(train_set, prefix = "demo_models", exp_dir = "./demo_models", nfolds = 10,
↪  grid_seed = 1) {
  h2o.init(nthreads = -1)
  h2o.removeAll()
  dir.create(exp_dir)
  tmp <- as.h2o(train_set, destination_frame = prefix)
  classification <- FALSE
  if (is.factor(train_set$response)) classification <- TRUE
  samp_factors <- NULL
  if (classification) {
    tmp["response"] <- as.factor(tmp["response"])
    samp_factors <- as.vector(mean(table(train_set$response)) / table(train_set$response))
  }
  y <- "response"
  x <- setdiff(names(tmp), y)
  res <- as.data.frame(tmp$response)
  # --------------------
  # base model training
  # --------------------
  cat("Deep learning grid 1\n")
  deeplearning_1 <- h2o.grid(
    algorithm = "deeplearning", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
```

```
  keep_cross_validation_predictions = TRUE, hyper_params = deeplearning_params_1,
  stopping_rounds = 3, balance_classes = classification,
  class_sampling_factors = samp_factors,
  search_criteria = list(
    strategy = "RandomDiscrete",
    max_models = 5, seed = grid_seed
  ),
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo", parallelism = 0
)
for (model_id in deeplearning_1@model_ids) {
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
}
cat("Deep learning grid 2\n")
deeplearning_2 <- h2o.grid(
  algorithm = "deeplearning", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, hyper_params = deeplearning_params_2,
  stopping_rounds = 3, balance_classes = classification,
  class_sampling_factors = samp_factors,
  search_criteria = list(
    strategy = "RandomDiscrete",
    max_models = 5, seed = grid_seed
  ),
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo", parallelism = 0
)
for (model_id in deeplearning_2@model_ids) {
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
}
cat("Deep learning grid 3\n")
deeplearning_3 <- h2o.grid(
  algorithm = "deeplearning", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, hyper_params = deeplearning_params_3,
  stopping_rounds = 3, balance_classes = classification,
  class_sampling_factors = samp_factors,
  search_criteria = list(
    strategy = "RandomDiscrete",
    max_models = 5, seed = grid_seed
  ),
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo", parallelism = 0
)
for (model_id in deeplearning_3@model_ids) {
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
```

```
}
cat("GBM grid\n")
gbm <- h2o.grid(
  algorithm = "gbm", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, hyper_params = gbm_params, stopping_rounds = 3,
  balance_classes = classification, class_sampling_factors = samp_factors,
  search_criteria = list(strategy = "RandomDiscrete", max_models = 15, seed = grid_seed),
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo", parallelism = 0
)
for (model_id in gbm@model_ids) {
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
}
cat("GBM 5 default models\n")
gbm_1 <- h2o.gbm(
  model_id = "GBM_1", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  balance_classes = classification, class_sampling_factors = samp_factors,
  col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8, learn_rate = 0.1,
  max_depth = 6, min_rows = 1, min_split_improvement = 1e-5, ntrees = 10000, sample_rate = 0.8,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
gbm_2 <- h2o.gbm(
  model_id = "GBM_2", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  balance_classes = classification, class_sampling_factors = samp_factors,
  col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8, learn_rate = 0.1,
  max_depth = 7, min_rows = 10, min_split_improvement = 1e-5, ntrees = 10000, sample_rate = 0.8,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
gbm_3 <- h2o.gbm(
  model_id = "GBM_3", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  balance_classes = classification, class_sampling_factors = samp_factors,
  col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8, learn_rate = 0.1,
  max_depth = 8, min_rows = 10, min_split_improvement = 1e-5, ntrees = 10000, sample_rate = 0.8,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
gbm_4 <- h2o.gbm(
  model_id = "GBM_4", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  balance_classes = classification, class_sampling_factors = samp_factors,
```

```
  col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8, learn_rate = 0.1,
  max_depth = 10, min_rows = 10, min_split_improvement = 1e-5, ntrees = 10000, sample_rate = 0.8,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
# gbm_5 = h2o.gbm(model_id='GBM_5', x=x, y=y, training_frame=tmp, seed=1, nfolds=nfolds,
#                 keep_cross_validation_predictions=TRUE, stopping_rounds=3, score_tree_interval=5,
#                 balance_classes=classification, class_sampling_factors=samp_factors,
#                 col_sample_rate=0.8, col_sample_rate_per_tree=0.8, learn_rate=0.1,
#                 max_depth=15, min_rows=100, min_split_improvement=1e-5, ntrees=10000,
↪  sample_rate=0.8,
#                 keep_cross_validation_models=FALSE, fold_assignment='Modulo')
for (model_id in paste0("GBM_", 1:4)) {
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
}
cat("XGBoost grid\n")
xgboost <- h2o.grid(
  algorithm = "xgboost", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, hyper_params = xgboost_params, stopping_rounds = 3,
  search_criteria = list(strategy = "RandomDiscrete", max_models = 15, seed = grid_seed),
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo", parallelism = 0
)
for (model_id in xgboost@model_ids) {
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
}
cat("XGBoost 3 default models\n")
xgboost_1 <- h2o.xgboost(
  model_id = "XGBoost_1", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  booster = "gbtree", col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8,
  max_depth = 10, min_rows = 5, ntrees = 10000, reg_alpha = 0, reg_lambda = 1,
  sample_rate = 0.6, keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
xgboost_2 <- h2o.xgboost(
  model_id = "XGBoost_2", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  booster = "gbtree", col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8,
  max_depth = 20, min_rows = 10, ntrees = 10000, reg_alpha = 0, reg_lambda = 1,
  sample_rate = 0.6, keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
xgboost_3 <- h2o.xgboost(
  model_id = "XGBoost_3", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
```

```
  keep_cross_validation_predictions = TRUE, stopping_rounds = 3, score_tree_interval = 5,
  booster = "gbtree", col_sample_rate = 0.8, col_sample_rate_per_tree = 0.8,
  max_depth = 5, min_rows = 3, ntrees = 10000, reg_alpha = 0, reg_lambda = 1,
  sample_rate = 0.8, keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
for (model_id in paste0("XGBoost_", 1:3)) {
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
}
cat("GLM\n")
glm <- h2o.glm(
  model_id = "GLM", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, alpha = c(0.0, 0.2, 0.4, 0.6, 0.8, 1.0),
  balance_classes = classification, class_sampling_factors = samp_factors,
  max_after_balance_size = 0.5, keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
tmp_path <- h2o.saveModel(h2o.getModel("GLM"), path = exp_dir, force = TRUE)
cat("DRF\n")
drf <- h2o.randomForest(
  model_id = "DRF", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, ntrees = 10000,
  score_tree_interval = 5, stopping_rounds = 3,
  balance_classes = classification,
  class_sampling_factors = samp_factors,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
tmp_path <- h2o.saveModel(h2o.getModel("DRF"), path = exp_dir, force = TRUE)
cat("XRT\n")
xrt <- h2o.randomForest(
  model_id = "XRT", x = x, y = y, training_frame = tmp, seed = 1, nfolds = nfolds,
  keep_cross_validation_predictions = TRUE, ntrees = 10000, histogram_type = "Random",
  score_tree_interval = 5, stopping_rounds = 3,
  balance_classes = classification,
  class_sampling_factors = samp_factors,
  keep_cross_validation_models = FALSE, fold_assignment = "Modulo"
)
tmp_path <- h2o.saveModel(h2o.getModel("XRT"), path = exp_dir, force = TRUE)
# -----------------------
# get holdout predictions
# -----------------------
base_models <- as.list(c(
  unlist(deeplearning_1@model_ids),
```

```
    unlist(deeplearning_2@model_ids),
    unlist(deeplearning_3@model_ids),
    unlist(gbm@model_ids), paste0("GBM_", 1:4),
    unlist(xgboost@model_ids), paste0("XGBoost_", 1:3),
    "GLM",
    "DRF",
    "XRT"
  ))
  for (model_id in base_models) {
    if (!classification) {
      res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_", model_id)),
        col.names = model_id
      ))
    } else {
      res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_", model_id))[3],
        col.names = model_id
      ))
    }
  }
  # ----------------------
  # super learner training
  # ----------------------
  sl_iter <- 0
  cat(paste0("Super learner iteration ", sl_iter, " (", length(base_models), " models)\n"))
  sl <- h2o.stackedEnsemble(
    x = x, y = y, model_id = paste0("superlearner_iter_", sl_iter),
    training_frame = tmp, seed = 1,
    base_models = base_models,
    metalearner_algorithm = "glm",
    metalearner_nfolds = nfolds,
    keep_levelone_frame = TRUE,
    metalearner_params = list(
      standardize = TRUE, keep_cross_validation_predictions = TRUE,
      balance_classes = classification,
      class_sampling_factors = samp_factors,
      max_after_balance_size = 0.5
    )
  )
  tmp_path <- h2o.saveModel(h2o.getModel(paste0("superlearner_iter_", sl_iter)), path = exp_dir, force
↪   = TRUE)
  model_id <- paste0("metalearner_glm_superlearner_iter_", sl_iter)
```

```
tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
if (!classification) {
  res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_", model_id)),
    col.names = paste0(model_id, "_", length(base_models), "_models")
  ))
} else {
  res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_", model_id))[3],
    col.names = paste0(model_id, "_", length(base_models), "_models")
  ))
}
# ---------------------------------
# super learner base model reduction
# ---------------------------------
while (TRUE) {
  meta <- h2o.getModel(paste0("metalearner_glm_superlearner_iter_", sl_iter, "_cv_1"))
  names <- meta@model$coefficients_table[, "names"]
  coeffs <- (meta@model$coefficients_table[, "standardized_coefficients"] > 0)
  for (j in 2:nfolds) {
    meta <- h2o.getModel(paste0("metalearner_glm_superlearner_iter_", sl_iter, "_cv_", j))
    names <- meta@model$coefficients_table[, "names"]
    coeffs <- coeffs + (meta@model$coefficients_table[, "standardized_coefficients"] > 0)
  }
  base_models_ <- as.list(names[coeffs >= ceiling(nfolds / 2) & names != "Intercept"])
  if (length(base_models_) == 0) {
    cat("No base models passing the threshold\n\n")
    break
  }
  if (sum(base_models %in% base_models_) == length(base_models)) {
    cat("No further reduction of base models\n\n")
    break
  }
  sl_iter <- sl_iter + 1
  base_models <- base_models_
  cat(paste0("Super learner iteration ", sl_iter, " (", length(base_models), " models)\n"))
  sl <- h2o.stackedEnsemble(
    x = x, y = y, model_id = paste0("superlearner_iter_", sl_iter),
    training_frame = tmp, seed = 1,
    base_models = base_models,
    metalearner_algorithm = "glm",
    metalearner_nfolds = nfolds,
    keep_levelone_frame = TRUE,
```

```
    metalearner_params = list(
      standardize = TRUE, keep_cross_validation_predictions = TRUE,
      balance_classes = classification,
      class_sampling_factors = samp_factors,
      max_after_balance_size = 0.5
    )
  )
  tmp_path <- h2o.saveModel(h2o.getModel(paste0("superlearner_iter_", sl_iter)), path = exp_dir,
↪ force = TRUE)
  model_id <- paste0("metalearner_glm_superlearner_iter_", sl_iter)
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
  if (!classification) {
    res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_", model_id)),
      col.names = paste0(model_id, "_", length(base_models), "_models")
    ))
  } else {
    res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_", model_id))[3],
      col.names = paste0(model_id, "_", length(base_models), "_models")
    ))
  }
}
# ----------------------------------------
# super learner for homogeneous base models
# ----------------------------------------
# DeepLearning
base_models <- as.list(c(
  unlist(deeplearning_1@model_ids),
  unlist(deeplearning_2@model_ids),
  unlist(deeplearning_3@model_ids)
))
cat(paste0("Super learner deep learning (", length(base_models), " models)\n"))
sl <- h2o.stackedEnsemble(
  x = x, y = y, model_id = "superlearner_deeplearning",
  training_frame = tmp, seed = 1,
  base_models = base_models,
  metalearner_algorithm = "glm",
  metalearner_nfolds = nfolds,
  keep_levelone_frame = TRUE,
  metalearner_params = list(
    standardize = TRUE, keep_cross_validation_predictions = TRUE,
    balance_classes = classification,
```

```
    class_sampling_factors = samp_factors,
    max_after_balance_size = 0.5
  )
)
tmp_path <- h2o.saveModel(h2o.getModel("superlearner_deeplearning"), path = exp_dir, force = TRUE)
model_id <- "metalearner_glm_superlearner_deeplearning"
tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
if (!classification) {
  res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_", model_id)),
    col.names = paste0(model_id, "_", length(base_models), "_models")
  ))
} else {
  res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_", model_id))[3],
    col.names = paste0(model_id, "_", length(base_models), "_models")
  ))
}
# GBM
base_models <- as.list(c(unlist(gbm@model_ids), paste0("GBM_", 1:4)))
cat(paste0("Super learner GBM (", length(base_models), " models)\n"))
sl <- h2o.stackedEnsemble(
  x = x, y = y, model_id = "superlearner_gbm",
  training_frame = tmp, seed = 1,
  base_models = base_models,
  metalearner_algorithm = "glm",
  metalearner_nfolds = nfolds,
  keep_levelone_frame = TRUE,
  metalearner_params = list(
    standardize = TRUE, keep_cross_validation_predictions = TRUE,
    balance_classes = classification,
    class_sampling_factors = samp_factors,
    max_after_balance_size = 0.5
  )
)
tmp_path <- h2o.saveModel(h2o.getModel("superlearner_gbm"), path = exp_dir, force = TRUE)
model_id <- "metalearner_glm_superlearner_gbm"
tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
if (!classification) {
  res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_", model_id)),
    col.names = paste0(model_id, "_", length(base_models), "_models")
  ))
} else {
```

```
    res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_", model_id))[3],
      col.names = paste0(model_id, "_", length(base_models), "_models")
    ))
  }
  # XGBoost
  base_models <- as.list(c(unlist(xgboost@model_ids), paste0("XGBoost_", 1:3)))
  cat(paste0("Super learner XGBoost (", length(base_models), " models)\n"))
  sl <- h2o.stackedEnsemble(
    x = x, y = y, model_id = "superlearner_xgboost",
    training_frame = tmp, seed = 1,
    base_models = base_models,
    metalearner_algorithm = "glm",
    metalearner_nfolds = nfolds,
    keep_levelone_frame = TRUE,
    metalearner_params = list(
      standardize = TRUE, keep_cross_validation_predictions = TRUE,
      balance_classes = classification,
      class_sampling_factors = samp_factors,
      max_after_balance_size = 0.5
    )
  )
  tmp_path <- h2o.saveModel(h2o.getModel("superlearner_xgboost"), path = exp_dir, force = TRUE)
  model_id <- "metalearner_glm_superlearner_xgboost"
  tmp_path <- h2o.saveModel(h2o.getModel(model_id), path = exp_dir, force = TRUE)
  if (!classification) {
    res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_", model_id)),
      col.names = paste0(model_id, "_", length(base_models), "_models")
    ))
  } else {
    res <- cbind(res, as.data.frame(h2o.getFrame(paste0("cv_holdout_prediction_", model_id))[3],
      col.names = paste0(model_id, "_", length(base_models), "_models")
    ))
  }
  write.csv(res, file = paste0(exp_dir, "/cv_holdout_predictions.csv"), row.names = FALSE)
  cat("\n\n")
  h2o.removeAll()
}
```

6.4 Run pipeline

```
data <-
↪  read.csv("https://archive.ics.uci.edu/ml/machine-learning-databases/parkinsons/parkinsons.data",
↪  check.names = FALSE, row.names = 1)

# Random subset
set.seed(10)
data <- data[sample(1:nrow(data), size = 50, replace = FALSE), ]
predictor_variables <- data[, colnames(data) != "status"]
response_variable <- as.factor(data$status)

system.time({
  reduced_data <- variable_reduction(predictor_variables = predictor_variables, response_variable =
↪  response_variable)
  model_training(train_set = reduced_data)
})
```

6.5 Session info

```
sessionInfo()
```

```
## R version 4.2.2 (2022-10-31)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Big Sur ... 10.16
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## loaded via a namespace (and not attached):
##  [1] bookdown_0.28     digest_0.6.29     R.methodsS3_1.8.2 magrittr_2.0.3
```

```
##  [5] evaluate_0.16     rlang_1.0.4       stringi_1.7.8     cli_3.3.0
##  [9] rstudioapi_0.14   R.utils_2.12.0    R.oo_1.25.0       vctrs_0.4.1
## [13] rmarkdown_2.16    styler_1.8.0      tools_4.2.2       stringr_1.4.1
## [17] R.cache_0.16.0    purrr_0.3.4       xfun_0.32         yaml_2.3.5
## [21] fastmap_1.1.0     compiler_4.2.2    htmltools_0.5.3   knitr_1.40
```