TENTACLES workflow for: Supervised Ensemble Learning Identifies Minimal Consensus Gene Signatures for Crohn's Disease Classification

Giorgio Montesi, Gabriel Dos Santos Mouta, Maria Novedrati, André Ferreira Cunha, Alessandro Fuschi, Simone Lucchesi, Chiara Sonnati, Annalisa Ciabattini, Francesco Santoro, Donata Medaglini, Helder Nakaya

Contents

1 Introduction

This vignette demonstrates the application of **TENTACLES** to bulk RNA-seq datasets from patients with Crohn's disease (CD) and healthy controls (HC), reproducing the analysis described in the manuscript *"Supervised Ensemble Learning Identifies Minimal Consensus Gene Signatures for Crohn's Disease Classification."*

1.1 Structure of the document

- **Setup**
  Preparation of libraries and datasets required to run the TENTACLES workflow.
- **Consensus gene-signature discovery on Dataset 1**
  - Preprocessing of Dataset 1 with `preProcess()` (normalization, filtering, optional batch correction).
  - Training of multiple supervised classifiers with `runClassifiers()` and embedded feature selection.
  - Derivation of a consensus gene-signature via `getConsensus()`.
  - DEG analysis via DESeq2.
  - Evaluation and comparison of DEG- vs consensus-based signatures using univariate and multivariate analyses (PCA, per-gene AUROC, MLP) with `testConsensus()`.
- **Cross-cohort validation on Dataset 2 (`testConsensus`)**
  - Preprocessing of Dataset 2.
  - Application of `testConsensus()` to assess consensus gene-signature reproducibility on a second independent dataset.
  - Benchmarking of consensus gene-signature performance against DEG-derived gene sets.
  - DEG Analysis on Dataset 2 and identification of common DEGs across Dataset 1 and 2.
- **Unsupervised external validation on Dataset 3**
  - Preprocessing of Dataset 3.
  - Use of `valConsensus()` to assess the discriminatory power of the signature via unsupervised clustering across six different methods.

2 Setup

This section imports the required libraries and the datasets used throughout the analysis.

```r
# Increase max size for parallel processing
options(future.globals.maxSize = 2 * 1024^3)
# Set seed for reproducibility
set.seed(1789)
# Load the package
library(cowplot)
library(scales)
library(tibble)
library(ggrepel)
library(DESeq2)
library(tidyverse)
library(TENTACLES)
```

The datasets used in this vignette correspond to publicly available intestinal bulk RNA-Seq samples from Crohn's disease (CD) patients and healthy controls. Raw sequencing data were originally retrieved from the NCBI Sequence Read Archive (SRA) under the following BioProjects:

- PRJNA248469 – **Dataset 1** (training cohort)
- PRJNA565216 and PRJNA985602 – **Dataset 2** (testing cohort)
- PRJNA702434 – **Dataset 3** (validation cohort)

All samples were selected based on the inclusion criteria described in the manuscript: intestinal biopsies from individuals clinically diagnosed with CD or from HC.

In the original study, the raw FASTQ files were processed through a uniform bioinformatics pipeline to ensure cross-study consistency. Briefly, quality control was performed with *fastp* (v1.0.1), reads were aligned to the GRCh38.p14 human reference genome using *STAR* (v2.7.10), strandedness was verified with *RSeQC* (v5.0.1), and gene-level quantification was obtained with *Rsubread* (v2.22.1).

For the present vignette, these processed count matrices and their sample metadata are re-used as a single ready-to-load object (`ibd_dataset.RData`, containing `ibd.count` (samples on rows, genes on columns) and `ibd.clin` (sample-level annotations)), deposited on Zenodo as cited in the paper.

```r
# Load all datasets
load("ibd_dataset.RData")
```

2.1 Dataset cleaning

For all downstream analyses we focus on CD versus HC (labeled `Not IBD` in the metadata), excluding *ulcerative colitis* (UC). The helper function below subsets a given study accession and returns harmonized objects (`desc`, `count`) aligned by sample ID, keeping only CD and Not IBD:

```r
# Helper Function
filter_study_dataset <- function(clin, count, study, class_exclude = "UC") {
  desc <- clin[clin$Study %in% study & !clin$class %in% class_exclude, ]
  count_sub <- count[rownames(count) %in% rownames(desc), ]
  rownames(desc) <- desc$ID
  list(desc = desc, count = count_sub)
}
```

3 Consensus gene-signature discovery on Dataset 1

In this section, we focus on **Dataset 1 (PRJNA248469)**, which serves as the **training dataset** for the construction of a robust gene signature distinguishing CD from HC.

The analysis proceeds through the following steps:

1. **Preprocessing**
   The dataset is processed with `preProcess()`, which performs library-size normalization, low-expression filtering, and metadata preparation for downstream classification.
2. **Classifier training and feature selection**
   A panel of supervised machine-learning models is trained using the `runClassifiers()` function. Each model is coupled with a dedicated **feature selection strategy**, executed within resampling folds to avoid information leakage. During training, **variable importance scores** are computed for all classifiers.
3. **Consensus signature extraction**
   Features that are recurrently selected across multiple high-performing models are aggregated into a unified **consensus gene-signature** using the `getConsensus()` function.
4. **Differential gene expression analysis (DEG)**
   DEGs are computed using **DESeq2** on the same dataset, serving as a classical baseline for comparison.
5. **Benchmarking**
   The discriminative performance of the consensus gene-signature is evaluated using `testConsensus()` and compared to the DEG-based gene set through univariate and multivariate analyses.

3.1 Process the data (`preProcess`)

The `preProcess()` function from **TENTACLES** performs **library-size normalization**, **low-expression filtering**, and —if specified— **batch-effect correction**.
It requires as input:

- `df.count`: a data frame of raw gene counts, with **samples as rows** and **genes as columns**;
- `df.clin`: a data frame containing **sample-level metadata**, with samples as rows and variables (e.g., class labels, batch ID) as columns;
- `class`: the name of the column in `df.clin` indicating the **outcome variable** (e.g., `"class"`), which must be a binary factor (e.g., `"CD"` vs `"Not IBD"`).

Optional arguments allow for batch correction (via `batch` and `covar.mod`), plotting (PCA and PVCA before and after batch correction), and threshold customization for low gene expression filtering (through `mincpm` and `minfraction`).

The `is.normalized` parameter specifies whether the input data are already normalized: it should be set to `TRUE` for **microarray** data and `FALSE` (default) for **RNA-Seq** data.

In this case, since the input consists of raw RNA-Seq counts, we leave `is.normalized = FALSE`:

```r
# Helper Function to select Dataset 1
tmp <- filter_study_dataset(ibd.clin, ibd.count, "PRJNA248469")
# Define Desc and Count
desc_d1 <- tmp$desc
```

```r
count_d1 <- tmp$count
# Preprocess data with preProcess()
pp_d1 <- preProcess(df.count = count_d1, df.clin = desc_d1, class = "class",
                    case.label = "CD", is.normalized = FALSE,
                    mincpm = 1, minfraction = 0.1, plot = FALSE)
rm(tmp)
```

The output of the `preProcess()` function is a `preProcess.obj` object, a structured S4 object containing all intermediate and final elements from the preprocessing pipeline.

```r
str(pp_d1, list.len = 4)
#> Formal class 'preProcess.obj' [package "TENTACLES"] with 4 slots
#>   ..@ raw      :'data.frame':    260 obs. of  10062 variables:
#>   .. ..$ TNFRSF4       : int [1:260] 120 53 146 144 56 74 34 178 101 30 ...
#>   .. ..$ TNFRSF18      : int [1:260] 36 53 111 106 30 93 49 107 57 38 ...
#>   .. ..$ PRDM16        : int [1:260] 45 34 7 46 9 27 9 13 18 29 ...
#>   .. ..$ PRKCZ         : int [1:260] 432 551 232 390 176 263 142 178 154 642 ...
#>   .. .. [list output truncated]
#>   ..@ processed:List of 2
#>   .. ..$ normalized   :'data.frame': 260 obs. of  8426 variables:
#>   .. .. ..$ TNFRSF4      : num [1:260] 4.2 3.24 4.91 4.03 3.9 ...
#>   .. .. ..$ TNFRSF18     : num [1:260] 2.63 3.24 4.53 3.62 3.09 ...
#>   .. .. ..$ PRDM16       : num [1:260] 2.91 2.68 1.26 2.56 1.7 ...
#>   .. .. ..$ PRKCZ        : num [1:260] 5.99 6.47 5.56 5.41 5.49 ...
#>   .. .. .. [list output truncated]
#>   .. ..$ adjusted.data:'data.frame': 260 obs. of  8426 variables:
#>   .. .. ..$ TNFRSF4      : num [1:260] 4.2 3.24 4.91 4.03 3.9 ...
#>   .. .. ..$ TNFRSF18     : num [1:260] 2.63 3.24 4.53 3.62 3.09 ...
#>   .. .. ..$ PRDM16       : num [1:260] 2.91 2.68 1.26 2.56 1.7 ...
#>   .. .. ..$ PRKCZ        : num [1:260] 5.99 6.47 5.56 5.41 5.49 ...
#>   .. .. .. [list output truncated]
#>   ..@ metadata :'data.frame':    260 obs. of  12 variables:
#>   .. ..$ Age           : num [1:260] 12.5 8 9 5.08 10.42 ...
#>   .. ..$ Assay.Type    : chr [1:260] "RNA-Seq" "RNA-Seq" "RNA-Seq" "RNA-Seq" ...
#>   .. ..$ Study         : chr [1:260] "PRJNA248469" "PRJNA248469" "PRJNA248469" "PRJNA248469" ...
#>   .. ..$ ID            : chr [1:260] "SAMN03322967" "SAMN03322968" "SAMN03322969" "SAMN03322970" ..
#>   .. .. [list output truncated]
#>   ..@ data.info:List of 2
#>   .. ..$ type      : chr "rnaseq"
#>   .. ..$ normalized: logi FALSE
```

Its main slots include:

- `raw`: the original count matrix provided as input;
- `processed`: a list of data frames containing the results of each preprocessing step, including:
  - `normalized`: the library-size normalized counts data frame;
  - `filtered.data`: genes passing expression thresholds;
  - `adjusted.data`: the final matrix used for classification (after optional batch correction);
- `metadata`: a data frame with sample-level annotations, **filtered and aligned** to the final count matrix;

- `data.info`: a list with metadata about the dataset, including data type (`\"RNA-Seq\"` or `\"microarray\"`), normalization status, number of samples and features, and whether batch correction was applied.

This object is compatible with all downstream TENTACLES functions (e.g., `runClassifiers()`, `testConsensus()`, `valConsensus()`), ensuring reproducibility and consistent sample tracking. The resulting `preProcess.obj` provides the input structure for the multi-model training step described next.

3.2 Training multiple classification algorithms (`runClassifiers`)

The function `runClassifiers()` trains, tunes, and evaluates several supervised models through a unified interface. Each classifier can be coupled with a specific feature-selection strategy, and model hyperparameters are optimized through cross-validation. Variable importance (VIP) is computed for each model, with a permutation-based fallback when direct computation is unavailable.

It takes as inputs:

- `preProcess.obj`: output of `preProcess()` containing the adjusted counts and aligned metadata. Alternatively, `df.count` and `df.clin` can be passed directly if preprocessing was performed outside TENTACLES.
- `models`: character vector of classifiers to fit. Supported options include: `"xgboost"`, `"bag_tree"`, `"lightGBM"`, `"pls"`, `"logistic"`, `"C5_rules"`, `"mars"`, `"bag_mars"`, `"mlp"`, `"bag_mlp"`, `"decision_tree"`, `"rand_forest"`, `"svm_linear"`, `"svm_poly"`, `"svm_rbf"`.
- `selector.recipes`: feature-selection strategy for each model. Supported: `"base"`, `"boruta"`, `"roc"`, `"infgain"`, `"mrmr"`, `"corr"`. If a single recipe is provided, it is applied to all models; otherwise, pairing follows position order.
- `tuning.method`: hyperparameter search. Options: `"tune_grid"` (default), `"tune_race_anova"`, `"tune_race_win_loss"`, `"tune_bayes"`, `"tune_sim_anneal"`.
- `n`: an integer specifying the number of iterations for the tuning method.
- `v`: an integer specifying the number of folds for the cross-validation during the hyperparameters tuning.
- **Thresholds and selectors**: `selector.threshold` (threshold parameter used for `"roc"`, `"infgain"`, `"mrmr"`, `"corr"`), `boruta.maxRuns` (for `"boruta"`).
- **Balancing and filtering**: `downsample = TRUE` to downsample the majority class; `filter = TRUE` to remove genes not annotated in GO/KEGG databases before modeling.
- **Other controls**: `metric` (default `"accuracy"`), `nsim` (# simulations for permutation VIP), `seed`, `plot`, and `parallel` (to speed-up computation).

In the manuscript, ten models were evaluated with model-specific selectors and 5-fold CV grid search; the optimal configuration per model was chosen by the highest F1-score across folds:

```r
# runClassifiers()
rc <- runClassifiers(
  pp_d1,
  models = c("C5_rules", "pls", "rand_forest", "bag_mlp", "decision_tree", "mlp",
             "svm_linear", "xgboost", "bag_tree", "lightGBM", "mars"),
  selector.recipes = c("boruta", "boruta", "corr", "roc", "boruta", "roc",
                       "boruta", "base", "mrmr", "roc", "boruta"),
  tuning.method = "tune_grid",
  filter = TRUE, downsample = TRUE,
  selector.threshold = 0.95, boruta.maxRuns = 100,
  parallel = TRUE,
  metric = "f_meas",
  n = 5, v = 5, seed = 1789,
  plot = FALSE
)
```

The function returns a S4 object of class `runClassifiers.obj` with:

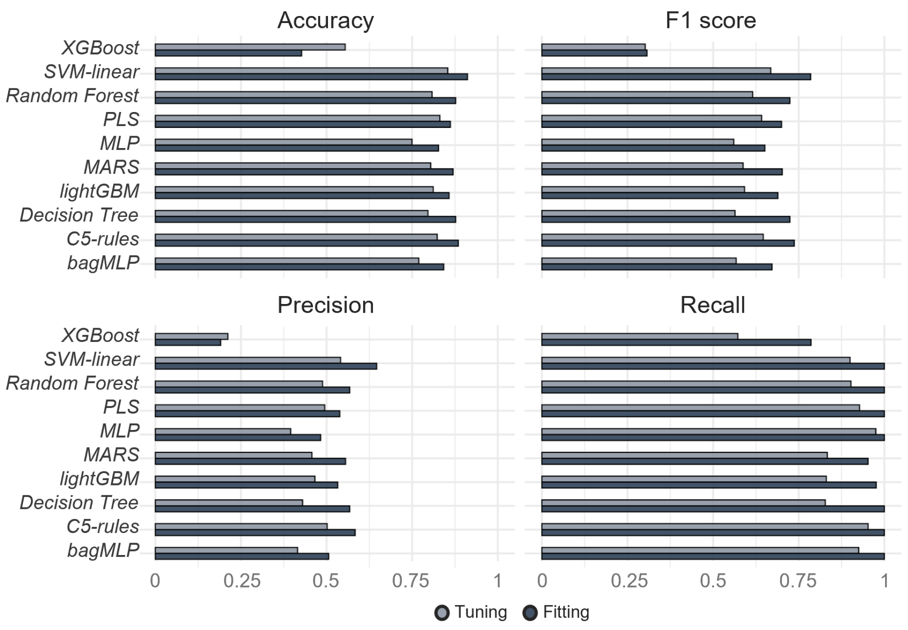
- `data`: adjusted data and metadata used for modeling.
- `models.info`: finalized workflows per model (recipes + model + tuned parameters).
- `model.features`: variable importance tables for each model.
- `performances`: tuning curves and final metrics for each model.
- `predictions`: out-of-fold predictions and associated probabilities, when available.

3.2.1 Visualizing model performance and feature overlap

After training, **TENTACLES** provides dedicated visualization utilities to interpret model performance and feature-selection overlap. Three main plotting utilities are available for objects returned by `runClassifiers()`.

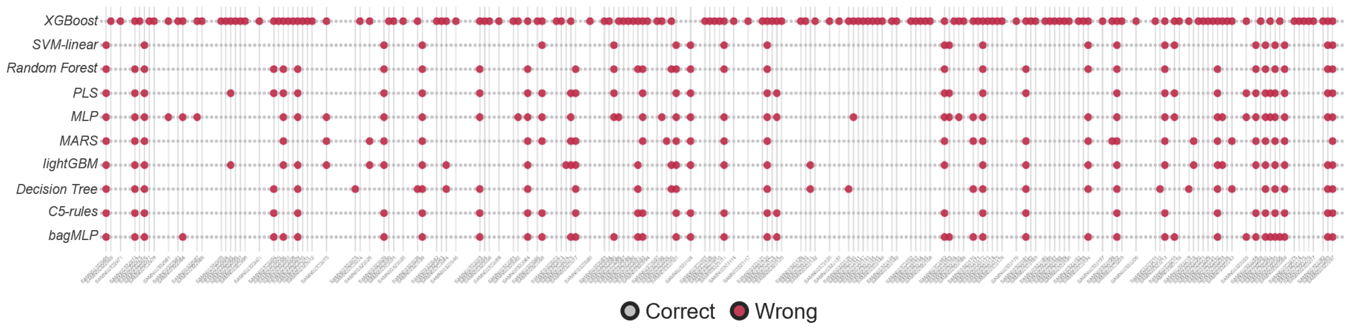
The `performances.plot()` function visualizes the cross-validation and final metrics for each trained model. It takes as input the `runClassifiers.obj` object and returns a grouped bar plot summarizing **Accuracy**, **F1-score**, **Precision**, and **Recall**.
Light-colored bars represent performance during resampling (cross-validation), whereas dark bars correspond to the final model fitted on the entire training set:

```
performances.plot(rc)
```



To examine how individual samples are classified by each model, the `wrong.preds.plot()` function can be used. It takes the `@predictions` slot of the `runClassifiers.obj` object and produces a concordance plot where each column represents a sample and each row a classifier.
Red dots indicate misclassified samples, and grey dots denote correct predictions.
This visualization highlights samples that are consistently difficult to classify across multiple algorithms:

```
wrong.preds.plot(rc@predictions) +
  theme( axis.text.x = element_text( angle = 45, hjust = 1, vjust = 1,
                                     margin = margin(t = 2),
                                     size = 6,
                                     color = "#7c7b7b"),
  plot.margin = margin(t = 10, r = 10, b = 5, l = 10)
  )
```

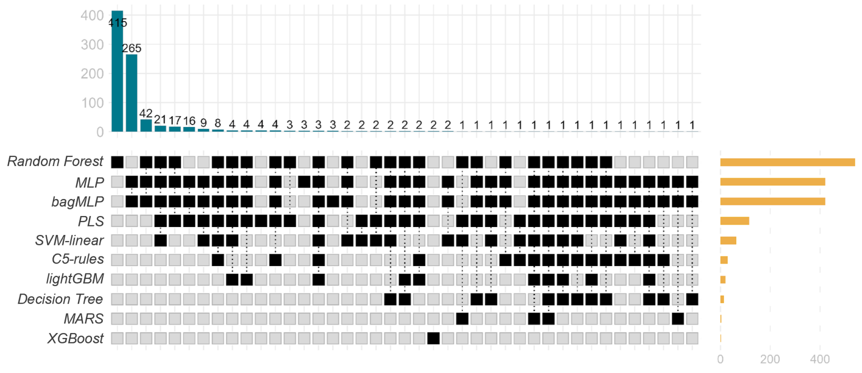


Note: Because these plots are built on **ggplot2**, they can be fully customized with standard `theme()` or aesthetic arguments, as shown above (e.g., adjusting text size or margins to better fit in documents).

The `upset.plot()` function visualizes the intersection of important genes across models.
It takes the `runClassifiers()` object and generates an UpSet plot showing how feature sets selected by different classifiers intersect.

Each horizontal row represents a model, and each vertical bar shows the size of an intersection (number of genes selected by that specific combination of models).
This provides a concise overview of shared vs. model-specific gene selections, helping identify robust features contributing to the consensus signature:

```
upset.plot(rc)
```



3.3 Identification of consensus gene-signature through ensemble learning (`getConsensus`)

After training individual models, it is possible to derive a consensus set of features repeatedly appearing across high performing classifiers derived and computed in the training part through the `runClassifiers()` function. To do this we use the `getConsensus()` function which takes as input a `runClassifiers.obj` or a binary data frame and supports 3 possible approaches:

- **Threshold-based selection:** assign values to `n_min` and `exclude` parameters. In this way we can exclude algorithms with poor performances and pick as consensus genes, those that appeared in at least a number of `n_min` algorithms.
- **Group-wise logic (single group):** leave as `NULL` these 2 parameters and specify instead `group1` and `meth1` parameters. In this way we can take the `'union'` or the `'intersection'` of the algorithms listed in `group1`.

- **Group-wise logic (two groups):** leave as `NULL` `n_min` and `exclude` while specifying `group1`, `group2`, `meth1`, `meth2` and `meth_comb` parameters all together. In this way we can take genes based on the `meth1` of `group1`, `meth_comb` with the `meth2` of `group2`. e.g. take the union (`meth1 = "union"`) of genes in `group1`, the union (`meth2 = "union"`) of genes in `group2` and pick the intersection (`meth_comb = "intersect"`) of these two groups.

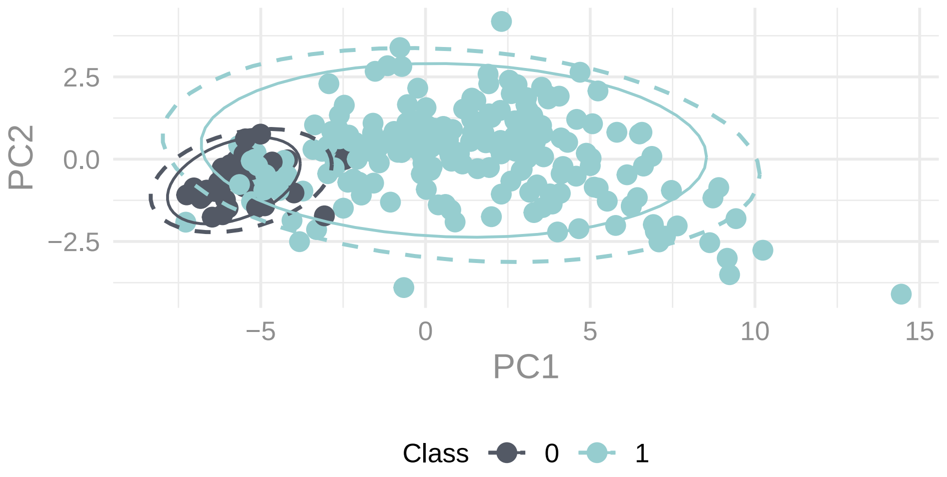
In this analysis, we used the **threshold-based approach** and retrieved genes that were appearing in at least six algorithms (`n.min = 6`) excluding XGBoost (`exclude = c(‘’xgboost_base’’)`):

```r
# getConsensus()
gc <- getConsensus(
  rc,
  n.min = 6,
  exclude = c("xgboost_base"),
  plot = FALSE
)
```

To analyze and visualize the discriminative performance of the genes selected by `getConsensus()`, the output object provides access to three different plots, each representing a complementary analysis of the consensus gene set.
These visualizations assess both multivariate and univariate behavior of the selected genes on the dataset used as input to `getConsensus()`.

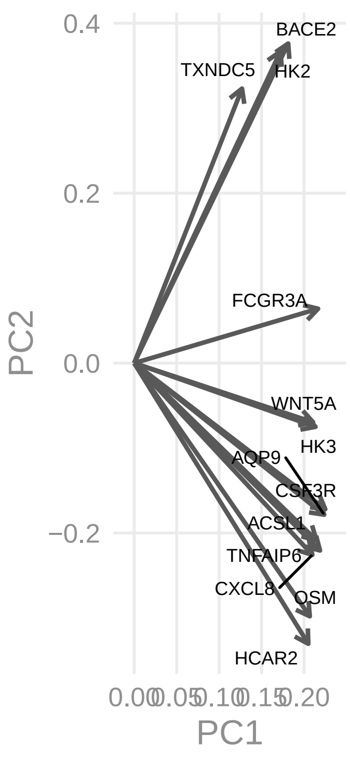
- `getConsensus.obj$test_consensus@pca$plots$pca_plot` displays a **Principal Component Analysis (PCA)** of the samples based on the consensus genes, showing overall class separation.

```r
gc$test_consensus@pca$plots$pca_plot
```



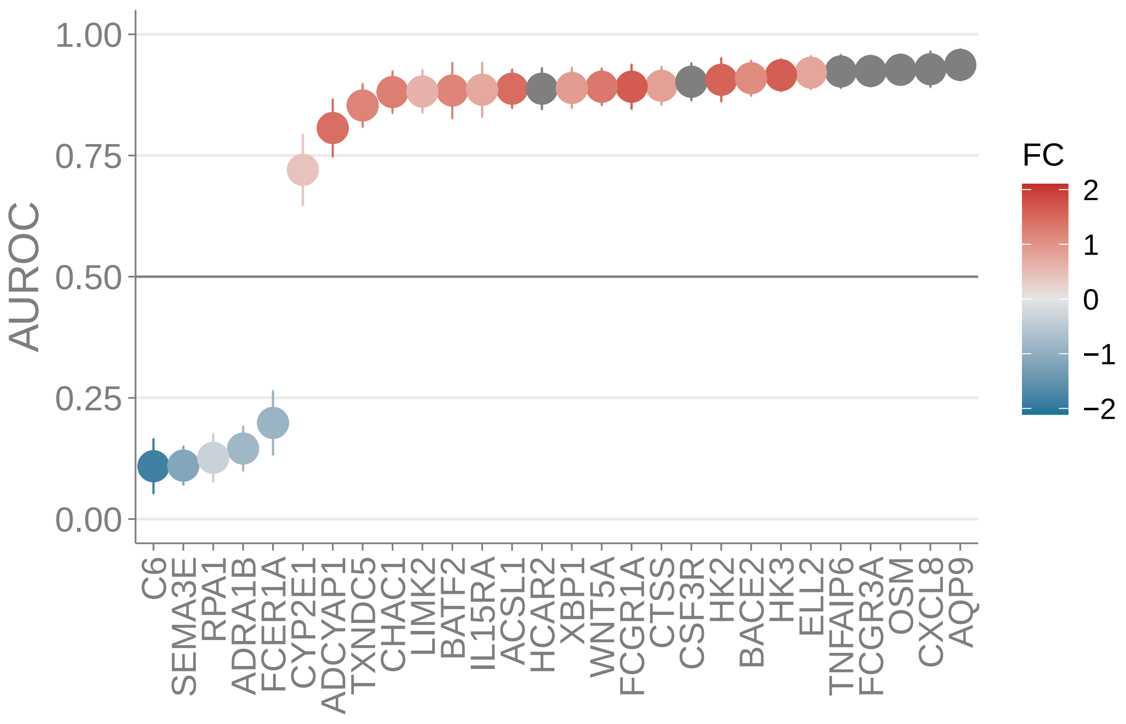
- `getConsensus.obj$test_consensus@pca$plots$loadings_plot` shows the **PCA loadings**, highlighting the contribution of each gene to the principal components and its recurrence across models.

```r
gc$test_consensus@pca$plots$loadings_plot
```



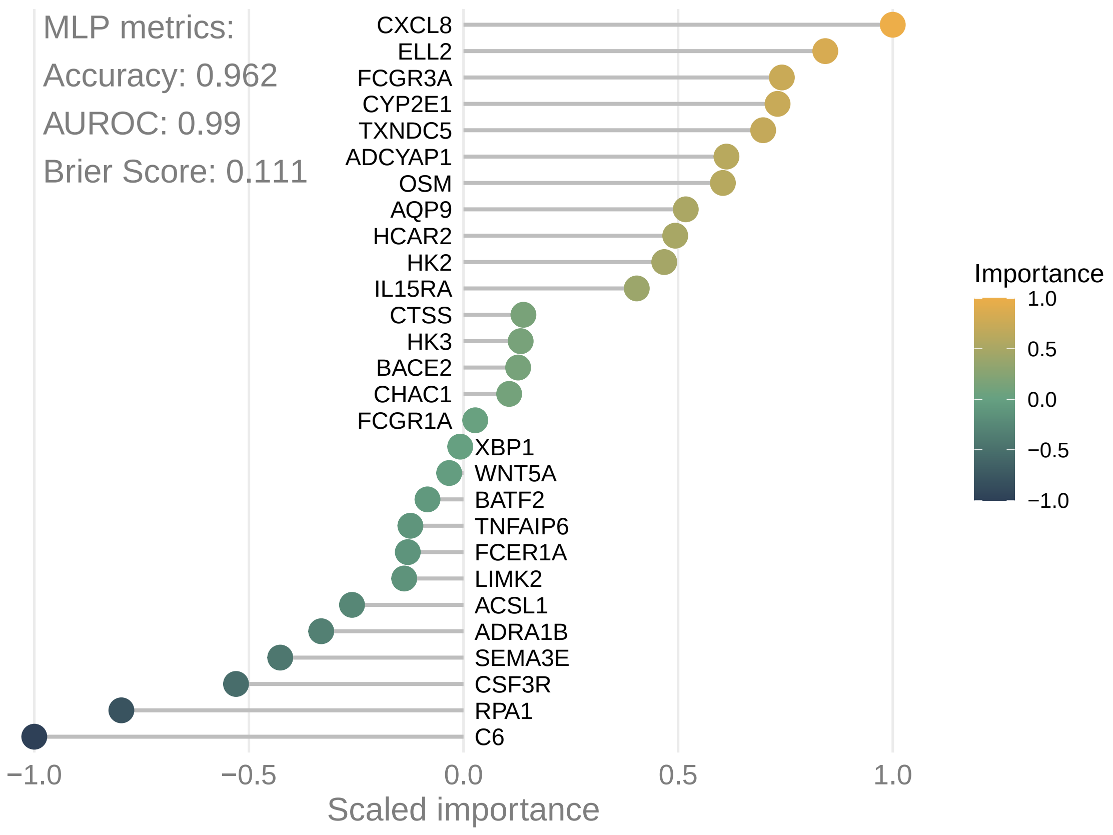
- `getConsensus.obj$test_consensus@auroc_fc$plot` summarizes **univariate performance** by plotting the AUROC of each gene, with colors indicating log2 fold change between classes.

```
gc$test_consensus@auroc_fc$plot
```



- `getConsensus.obj$test_consensus@mlp$plot` reports the performance of a **multivariate MLP classifier** trained only on the consensus genes, including classification metrics and variable importance.

```
gc$test_consensus@mlp$plot
```



3.4 Differential gene expression analysis (DEG)

To complement the workflow, we conducted a classical **differential expression (DEG)** analysis on the training cohort (**Dataset 1 – PRJNA248469**). The goal was to identify genes significantly modulated between CD and HC, to later compare their discriminative performance against the **TENTACLES consensus signature**.

Raw counts were filtered for low-coverage samples (total reads < 500,000), normalized with **DESeq2**, and modeled with the formula `~ class`. Genes were considered differentially expressed if **padj < 0.05** and **|log2FC| >= 1**.
The resulting DEG list (hereafter `genes_dds_d1`) served as a baseline reference for benchmarking:

```
# Transpose the gene count matrix and align sample names with clinical data
x <- t(count_d1)
desc <- desc_d1
m <- match(desc$ID, colnames(x))

# Filter the count table to include only samples present in clinical data
x <- x[, m]
```

```r
# Remove samples with low total counts (<500,000) and update clinical data
keepsamples <- colSums(x) > 500000
x <- x[, keepsamples]
desc <- desc[keepsamples, ]
# Clean and prepare class and batch columns in clinical data
desc$class <- as.factor(desc$class)
desc$class <- relevel(desc$class, ref = "Not IBD")  # Set "Not IBD" as reference
desc$batch2 <- as.factor(paste(desc$batch, desc$internal.batch))  # Combine batch info
desc$Group <- as.factor(paste(desc$batch, desc$class))  # Create group column

# ---------------------------- DESeq2 Analysis ----------------------------
# Create DESeq2 dataset object using count data and clinical metadata
dds_d1 <- DESeqDataSetFromMatrix(
  countData = x,
  colData = desc,
  design = ~ class  # Design formula: batch and class as factors
)

keep <- rownames(dds_d1) %in% colnames(pp_d1@processed$adjusted.data)

# Subset the DESeq2 dataset to retain only highly expressed genes
dds_d1 <- dds_d1[keep, ]

# Perform DESeq2 normalization and differential expression analysis
dds_d1 <- DESeq(dds_d1)

# Clean up unnecessary objects from the environment
rm(x, m, keep, keepsamples, desc)
```

3.5 Benchmarking via `testConsensus`

To assess the discriminative performance of the **differentially expressed genes** identified and to compare it with the discriminative performance of the consensus gene-signature, we use the `testConsensus()` function. This function performs a comprehensive analysis of the input genes on the dataset passed as input, combining univariate and multivariate evaluations.

Specifically, `testConsensus()` includes the following components:

- **Principal Component Analysis (PCA):**
  Dimensionality reduction based on the consensus genes.
  The top `top_loadings` genes (by absolute loading) are identified, and the PCA scores and loadings are returned showing class separation.
- **AUROC and Fold Change analysis:**
  For each gene, the function computes the **Area Under the ROC Curve (AUROC)** and the **log2 fold change** between classes. These metrics are summarized in a data frame and visualized in a joint plot.
- **Heatmap with hierarchical clustering (HC):**
  A normalized expression heatmap of the consensus genes, grouped by hierarchical clustering, to visualize co-expression and class-specific patterns.
- **Multi-Layer Perceptron (MLP):**
  A multivariate model is trained on the consensus gene set.
  The resulting output includes the **model's performance metrics** and **feature importance** scores.

The output is an S4 object of class `testConsensus.obj`, containing all intermediate data structures and ggplot2 objects for full customization and reuse.

```r
# Retrieve DEG results
res_dds_d1 <- results(dds_d1, alpha = 0.05)
# Subset DEGs
res_dds_d1 <- subset(res_dds_d1,
                     padj < 0.05 &
                       abs(log2FoldChange) >= 1)
# Define the final DEGs list
genes_dds_d1 <- rownames(res_dds_d1)
```

```r
# Apply testConsensus() to DEGs
tc_dds_d1_d1 <- testConsensus(df.count = pp_d1@processed$adjusted.data,
                              gene.list = genes_dds_d1,
                              class = pp_d1@metadata$class)
```

4 Cross-cohort validation on Dataset 2 (`testConsensus`)

To evaluate the **cross-cohort generalizability** of the consensus gene-signature identified in the training dataset (Dataset 1), we use an independent test set (**Dataset 2**) composed of two additional studies: **PRJNA565216** and **PRJNA985602**.

These datasets are first **merged** to obtain a more heterogeneous cohort and then processed using the `preProcess()` function to perform normalization and batch-effect correction.

We then apply `testConsensus()` to this combined dataset to evaluate the performance of the consensus gene-signature—both at the univariate and multivariate level.
Finally, we compare these results to those obtained using differentially expressed genes (DEGs) from **Dataset 1**, providing a benchmark for assessing the added value of the ensemble-derived signature.

4.1 Preprocess of Dataset 2 composed by PRJNA565216 and PRJNA985602

Since the two studies originate from different experiments, they introduce **batch effects** that must be addressed before downstream analysis.
To handle this, we create a new variable—`batch_new`—and we pass this variable to the `batch` argument of `preProcess()` function:

```r
# Create Dataset 2 by combining PRJNA565216 and PRJNA985602
tmp <- filter_study_dataset(ibd.clin, ibd.count, c("PRJNA565216", "PRJNA985602"))
desc_g23 <- tmp$desc
count_g23 <- tmp$count
# Create batch_new to account for different studies
desc_g23$batch_new <- as.factor(paste0(desc_g23$batch, desc_g23$internal.batch))
# preprocess Dataset 2
pp_g23 <- preProcess(df.count = count_g23, df.clin = desc_g23, batch = "batch_new",
                     class = "class", case.label = "CD", plot = FALSE)
rm(tmp)
```

4.2 External Validation of the consensus gene-signature

The first external validation of the consensus gene-signature is performed using the `testConsensus()` function, applied to the merged Dataset 2 (Studies 2 and 3):

```r
# External validation via testConsensus()
tc <- testConsensus(df.count = pp_g23@processed$adjusted.data,
                    gene.list = gc$consensusGenes,
                    class = pp_g23@metadata$class,
                    plot = FALSE)
```

The output of `testConsensus()` is a structured S4 object of class `testConsensus.obj`, which includes:

- **`data`**: a list with the input expression matrix (`df.count`), gene list (`gene.list`), and class labels (`class`).
- **`pca`**: PCA results, including sample scores, gene loadings, variance explained, and the corresponding PCA plot.
- **`auroc_fc`**: per-gene **AUROC** and **log2 fold change** values, alongside a summary plot.

- **heatmap**: a heatmap with **hierarchical clustering** of the consensus genes.
- **mlp**: a multivariate **MLP classifier**, with model performance metrics, variable importances, and an associated plot.

The resulting visualizations can be accessed directly from the object using:

```
tc@pca$plots$pca_plot
tc@pca$plots$loadings_plot
tc@auroc_fc$plot
tc@mlp$plot
```

4.3 Comparison with classical DEG Analysis

To provide a direct benchmark, we applied the same evaluation procedure to the set of **DEGs** identified in **Dataset 1** via DESeq2. Specifically, we use the `testConsensus()` function to assess the **discriminative performance** of the DEG panel when applied to **Dataset 2**, following the same approach used for the TENTACLES-derived signature:

```
tc_dds_d1_g23 <- testConsensus(df.count = pp_g23@processed$adjusted.data,
                               gene.list = genes_dds_d1,
                               class = pp_g23@metadata$class)
```

The resulting testConsensus.obj is again an S4 object of five elements containing all results, informations and plots accessible as previously shown.

```
str(tc_dds_d1_g23@data, max.level = 4, vec.len = 2, list.len = 5)
#> List of 3
#>  $ df.count :'data.frame':   179 obs. of  8367 variables:
#>   ..$ TNFRSF4      : num [1:179] 1 2.54 ...
#>   ..$ TNFRSF18     : num [1:179] 2.32 3.1 ...
#>   ..$ PRDM16       : num [1:179] 2.44 2.17 ...
#>   ..$ PRKCZ        : num [1:179] 0.775 0.775 ...
#>   ..$ ACAP3        : num [1:179] 3.81 2.35 ...
#>   .. [list output truncated]
#>  $ gene.list: chr [1:582] "MYOM3" "AJAP1" ...
#>  $ class    : Factor w/ 2 levels "0","1": 2 2 2 2 2 ...
```

```
str(tc_dds_d1_g23@pca, max.level = 2, vec.len = 2, list.len = 5)
#> List of 5
#>  $ scores             :'data.frame':  179 obs. of  180 variables:
#>   ..$ sample: chr [1:179] "SAMN12736816" "SAMN12736818" ...
#>   ..$ PC1   : num [1:179] -9.53 6.38 ...
#>   ..$ PC2   : num [1:179] -4.664 -0.208 ...
#>   ..$ PC3   : num [1:179] 3.48 2.32 ...
#>   ..$ PC4   : num [1:179] -6.4 -1.81 ...
#>   .. [list output truncated]
#>  $ loadings           :'data.frame':  541 obs. of  180 variables:
#>   ..$ gene : chr [1:541] "MYOM3" "AJAP1" ...
#>   ..$ PC1  : num [1:541] -0.0309 0.0125 ...
#>   ..$ PC2  : num [1:541] 0.1053 0.0229 ...
#>   ..$ PC3  : num [1:541] -0.0689 0.0333 ...
#>   ..$ PC4  : num [1:541] 0.01335 0.00527 ...
#>   .. [list output truncated]
#>  $ explained_variance:'data.frame':  179 obs. of  4 variables:
#>   ..$ PC      : int [1:179] 1 2 3 4 5 ...
#>   ..$ variance: num [1:179] 160.2 31.1 ...
#>   ..$ pct     : num [1:179] 29.61 5.75 ...
#>   ..$ pct_cum : num [1:179] 29.6 35.4 ...
#>  $ top_loadings      :'data.frame':  15 obs. of  3 variables:
#>   ..$ gene: chr [1:15] "MYOM3" "AADAC" ...
#>   ..$ PC1 : num [1:15] -0.0309 -0.0491 ...
#>   ..$ PC2 : num [1:15] 0.105 0.106 ...
#>  $ plots             :List of 2
#>   ..$ pca_plot     :List of 11
#>   .. ..- attr(*, "class")= chr [1:2] "gg" "ggplot"
#>   ..$ loadings_plot:List of 11
#>   .. ..- attr(*, "class")= chr [1:2] "gg" "ggplot"
```

```
str(tc_dds_d1_g23@auroc_fc, max.level = 1, vec.len = 2, list.len = 5)
#> List of 2
#>  $ data:'data.frame':    541 obs. of  5 variables:
#>  $ plot:List of 11
#>   ..- attr(*, "class")= chr [1:2] "gg" "ggplot"
```

```
str(tc_dds_d1_g23@heatmap, max.level = 1, vec.len = 2, list.len = 5)
#> List of 5
#>  $ matrix     :'data.frame': 541 obs. of  179 variables:
#>  $ hc_rows    :List of 7
#>   ..- attr(*, "class")= chr "hclust"
#>  $ hc_cols    :List of 7
#>   ..- attr(*, "class")= chr "hclust"
#>  $ side_colors:'data.frame': 179 obs. of  1 variable:
#>  $ plot       :List of 8
#>   ..- attr(*, "class")= chr [1:2] "plotly" "htmlwidget"
#>   ..- attr(*, "package")= chr "plotly"
```

```
str(tc_dds_d1_g23@mlp, max.level = 1, vec.len = 2, list.len = 5)
#> List of 4
#>  $ models.info     :List of 4
#>   ..- attr(*, "class")= chr "workflow"
#>  $ importances     :'data.frame':    534 obs. of  4 variables:
#>   ..- attr(*, "type")= chr "olden"
#>  $ test_performance: tibble [3 x 4] (S3: tbl_df/tbl/data.frame)
#>  $ plot            :List of 11
#>   ..- attr(*, "class")= chr [1:2] "gg" "ggplot"
```

4.4 Independent DEG Analysis on Dataset 2

Finally, we also performed an **independent DEG analysis** on the merged **Dataset 2**.
The following code reproduces this step, filtering low-count samples and including both `class` and `batch2` as factors in the model design:

```r
x <- t(count_g23)
desc <- desc_g23
m <- match(desc$ID, colnames(x))

x <- x[, m]
desc$ID == colnames(x)

keepsamples <- colSums(x) > 500000
x <- x[, keepsamples]
desc <- desc[keepsamples, ]

desc$class <- as.factor(desc$class)
desc$class <- relevel(desc$class, ref = "Not IBD")
desc$batch2 <- as.factor(paste(desc$batch, desc$internal.batch))
desc$Group <- as.factor(paste(desc$batch, desc$class))

dds_g23 <- DESeqDataSetFromMatrix(
  countData = x,
  colData = desc,
  design = ~ class + batch2
)

keep <- rownames(dds_g23) %in% colnames(pp_g23@processed$adjusted.data)
dds_g23 <- dds_g23[keep, ]
dds_g23 <- DESeq(dds_g23)

res_dds_g23 <- results(dds_g23, alpha = 0.05)

res_dds_g23 <- subset(res_dds_g23,
                      padj < 0.05 &
                        abs(log2FoldChange) >= 1)

genes_dds_g23 <- rownames(res_dds_g23)

rm(x, m, keep, keepsamples, desc)
```

And to assess intersection among consensus gene-signature, DEGs derived genes from Dataset 1 and DEGs derived genes from Dataset 2 we used the following:

```r
# Define dds_d1 and genes_dds_g23 intersection
intersection_deseqs <- intersect(genes_dds_d1, genes_dds_g23)
# Find overlap with TENTACLES consensus gene-signature
intersect(intersection_deseqs, gc$consensusGenes)
#>  [1] "FCGR3A"  "CSF3R"   "HK2"     "TNFAIP6" "WNT5A"   "CXCL8"   "ACSL1"
#>  [8] "HK3"     "ADRA1B"  "C6"      "TXNDC5"  "SEMA3E"  "BATF2"   "HCAR2"
#> [15] "AQP9"    "OSM"     "BACE2"
```

5 Unsupervised external validation of the consensus-gene signature on Dataset 3 (`valConsensus`)

To further evaluate the **cross-cohort generalizability** of the consensus gene-signature identified in the training dataset (Dataset 1) and previously tested on an independent test set (Dataset 2), we used a third dataset for **external unsupervised validation**: **PRJNA702434** (referred to here as **Dataset 3**).

For this purpose, we first preprocess the dataset anf then apply the `valConsensus()` function, which assesses the **discriminative capacity** of a gene set using multiple unsupervised clustering methods and ranks combinations based on internal clustering metrics.

5.1 Preprocess of Dataset 3: PRJNA702434

```r
# Helper Function to get Dataset 3
tmp <- filter_study_dataset(ibd.clin, ibd.count, "PRJNA702434")
desc_d4 <- tmp$desc
count_d4 <- tmp$count
# Preprocess Dataset 3
pp_d4 <- preProcess(df.count = count_d4, df.clin = desc_d4,
                    class = "class", case.label = "CD")
rm(tmp)
```

5.2 Validation of the consensus gene-signature

The `valConsensus()` function provides an **unsupervised validation framework** for assessing the **discriminative power** of a gene signature on a new dataset, without fitting supervised models.

Given an expression matrix (`df.count`) and a set of input genes (`gene.list`), the function evaluates the **clustering performance** of multiple combinations of genes across a panel of unsupervised algorithms.

Specifically, the following clustering methods are implemented:

- **K-Means**
- **Gaussian Mixture Models (GMM)**
- **Hierarchical Clustering**
- **K-Means on PCA-reduced dimensions**
- **K-Means on t-SNE-reduced dimensions**
- **K-Means on UMAP-reduced dimensions**

For each method and gene combination, the function computes standard classification metrics by comparing cluster assignments to the known class labels (`labels` vector parameter), including Accuracy, Precision, Recall and F1 Score (default metric for ranking).

The user can also specify:

- `N`: the number of top-performing gene combinations to retain (default = 10),
- `metric`: the evaluation criterion to prioritize combinations (`"Accuracy"`, `"Precision"`, `"Recall"`, or `"FScore"`).

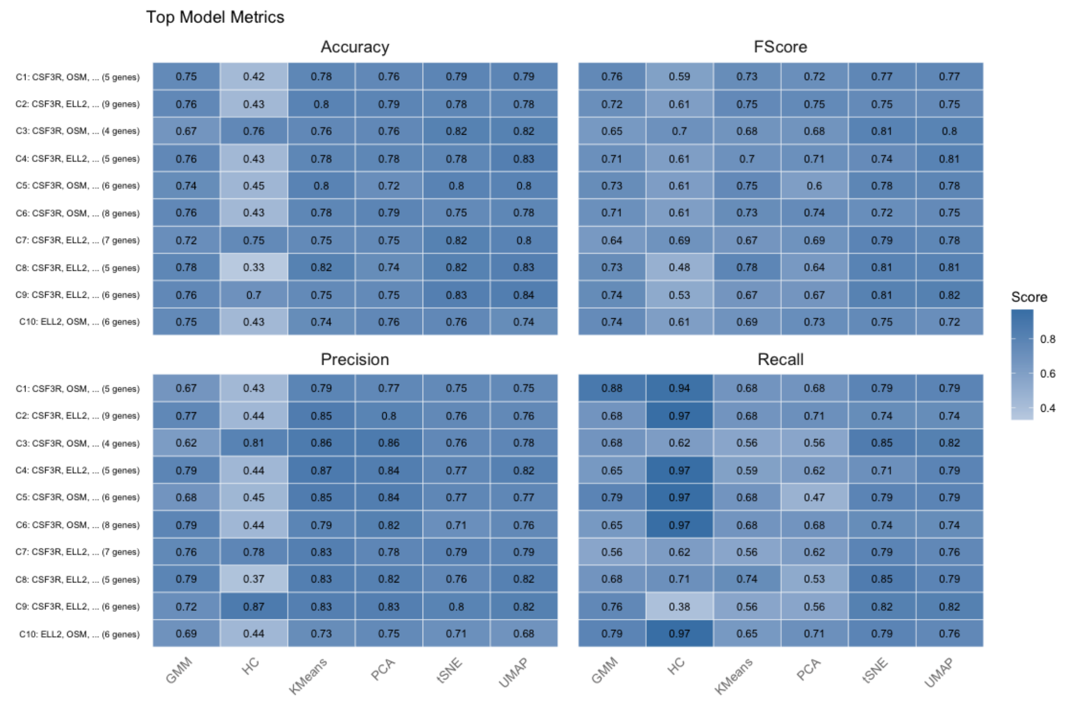
In the manuscript, we applied `valConsensus()` using a refined subset of genes that **preserved the direction of importance**(i.e., same sign of contribution) in both the training phase on **Dataset 1** and the cross-cohort testing on **Dataset 2**:

```
#Analysis----
# Joining the importance from d1 and g23-----
importances_d1g23 <- full_join(

  gc$test_consensus@mlp$importances %>%
    mutate(d1_signal = ifelse(Importance > 0, "Positive", "Negative")) %>%
    select(Variable, Importance_D1 = Importance, d1_signal)
  ,

  tc@mlp$importances %>%
    mutate(g23_signal = ifelse(Importance > 0, "Positive", "Negative")) %>%
    select(Variable, Importance_g23 = Importance, g23_signal)

) %>%
  mutate(same_signal = d1_signal == g23_signal,
         sum_importance = abs(Importance_D1 + Importance_g23)) %>%
  arrange(desc(same_signal), desc(sum_importance))

# Pull genes that have the same importance direction
genes_for_valcons <- importances_d1g23 %>%
  filter(same_signal) %>%
  pull(Variable)
```

```
# valConsensus
vc <- valConsensus(
  df.count = pp_d4@processed$adjusted.data,
  gene.list = genes_for_valcons,
  class = pp_d4@metadata$class
)
```

The output is a list containing clustering results, evaluation metrics, and ranked gene combinations, which can be visualized using the `plotTopMetrics()` function that creates a heatmap to visualize the performance of different clustering and dimensionality reduction models across multiple evaluation metrics, for each gene combination. Each combination is labeled with a concise and readable gene summary on the y-axis, prefixed by a combination number (C1, C2, . . .):

```r
plotTopMetrics(vc$topCombinations)
```



6 Session Info

```
devtools::session_info()
#> Warning in system2("quarto", "-V", stdout = TRUE, env = paste0("TMPDIR=", :
#> running command '"quarto"
#> TMPDIR=C:/Users/gabri/AppData/Local/Temp/RtmpSQvQup/filea1bc50b55154 -V' had
#> status 1
#> - Session info ---------------------------------------------------------------
#>  setting  value
#>  version  R version 4.4.3 (2025-02-28 ucrt)
#>  os       Windows 11 x64 (build 26200)
#>  system   x86_64, mingw32
#>  ui       RTerm
#>  language en
#>  collate  Portuguese_Brazil.utf8
#>  ctype    Portuguese_Brazil.utf8
#>  tz       America/Sao_Paulo
#>  date     2025-11-02
#>  pandoc   3.2 @ C:/Program Files/RStudio/resources/app/bin/quarto/bin/tools/ (via rmarkdown)
#>  quarto   NA @ C:\\PROGRA~1\\RStudio\\RESOUR~1\\app\\bin\\quarto\\bin\\quarto.exe
#>
#> - Packages -------------------------------------------------------------------
#>  package              * version    date (UTC) lib source
#>  abind                  1.4-8      2024-09-12 [1] CRAN (R 4.4.1)
#>  annotate               1.84.0     2024-10-29 [1] Bioconductor 3.20 (R 4.4.1)
#>  AnnotationDbi          1.68.0     2024-10-29 [1] Bioconductor 3.20 (R 4.4.1)
#>  assertthat             0.2.1      2019-03-21 [1] CRAN (R 4.4.3)
#>  baguette               1.1.0      2025-01-28 [1] CRAN (R 4.4.3)
#>  Biobase              * 2.66.0     2024-10-29 [1] Bioconductor 3.20 (R 4.4.1)
#>  BiocGenerics         * 0.52.0     2024-10-29 [1] Bioconductor 3.20 (R 4.4.1)
#>  BiocParallel           1.40.2     2025-06-04 [1] Bioconductor
#>  Biostrings             2.74.1     2024-12-16 [1] Bioconductor 3.20 (R 4.4.2)
#>  bit                    4.6.0      2025-03-06 [1] CRAN (R 4.4.3)
#>  bit64                  4.6.0-1    2025-01-16 [1] CRAN (R 4.4.2)
#>  blob                   1.2.4      2023-03-17 [1] CRAN (R 4.4.1)
#>  bonsai                 0.4.0      2025-06-25 [1] CRAN (R 4.4.3)
#>  Boruta                 9.0.0      2025-07-22 [1] CRAN (R 4.4.3)
#>  C50                    0.2.0      2025-04-03 [1] CRAN (R 4.4.3)
#>  ca                     0.71.1     2020-01-24 [1] CRAN (R 4.4.3)
#>  cachem                 1.1.0      2024-05-16 [1] CRAN (R 4.4.1)
#>  class                  7.3-23     2025-01-01 [2] CRAN (R 4.4.3)
#>  cli                    3.6.5      2025-04-23 [1] CRAN (R 4.4.3)
#>  codetools              0.2-20     2024-03-31 [2] CRAN (R 4.4.3)
#>  colino                 0.0.1      2025-04-28 [1] Github (stevenpawley/colino@9a786ce)
#>  colorspace             2.1-1      2024-07-26 [1] CRAN (R 4.4.1)
#>  corpcor                1.6.10     2021-09-16 [1] CRAN (R 4.4.0)
#>  cowplot              * 1.2.0      2025-07-07 [1] CRAN (R 4.4.3)
#>  crayon                 1.5.3      2024-06-20 [1] CRAN (R 4.4.1)
#>  Cubist                 0.5.0      2025-04-03 [1] CRAN (R 4.4.3)
#>  data.table             1.17.8     2025-07-10 [1] CRAN (R 4.4.3)
#>  DBI                    1.2.3      2024-06-02 [1] CRAN (R 4.4.1)
#>  DelayedArray           0.32.0     2024-10-29 [1] Bioconductor 3.20 (R 4.4.1)
#>  dendextend             1.19.1     2025-07-15 [1] CRAN (R 4.4.3)
#>  DESeq2              * 1.46.0    2024-10-29 [1] Bioconductor 3.20 (R 4.4.1)
#>  devtools              2.4.5     2022-10-11 [1] CRAN (R 4.4.1)
#>  dials                 1.4.2     2025-09-04 [1] CRAN (R 4.4.3)
#>  DiceDesign            1.10      2023-12-07 [1] CRAN (R 4.4.3)
#>  dichromat             2.0-0.1   2022-05-02 [1] CRAN (R 4.4.0)
#>  digest                0.6.37    2024-08-19 [1] CRAN (R 4.4.1)
#>  dplyr               * 1.1.4     2023-11-17 [1] CRAN (R 4.4.3)
#>  edgeR                 4.4.2     2025-01-27 [1] Bioconductor 3.20 (R 4.4.2)
#>  ellipse               0.5.0     2023-07-20 [1] CRAN (R 4.4.2)
#>  ellipsis              0.3.2     2021-04-29 [1] CRAN (R 4.4.1)
#>  evaluate              1.0.5     2025-08-27 [1] CRAN (R 4.4.3)
#>  farver                2.1.2     2024-05-13 [1] CRAN (R 4.4.1)
#>  fastmap               1.2.0     2024-05-15 [1] CRAN (R 4.4.1)
#>  forcats             * 1.0.0     2023-01-29 [1] CRAN (R 4.4.1)
#>  foreach               1.5.2     2022-02-02 [1] CRAN (R 4.4.1)
#>  Formula               1.2-5     2023-02-24 [1] CRAN (R 4.4.0)
#>  fs                    1.6.6     2025-04-12 [1] CRAN (R 4.4.3)
#>  FSelectorRcpp         0.3.13    2024-10-02 [1] CRAN (R 4.4.3)
#>  furrr                 0.3.1     2022-08-15 [1] CRAN (R 4.4.2)
#>  future                1.67.0    2025-07-29 [1] CRAN (R 4.4.3)
#>  future.apply          1.20.0    2025-06-06 [1] CRAN (R 4.4.3)
#>  genefilter            1.88.0    2024-10-29 [1] Bioconductor 3.20 (R 4.4.1)
#>  generics              0.1.4     2025-05-09 [1] CRAN (R 4.4.3)
#>  GenomeInfoDb        * 1.42.3    2025-01-27 [1] Bioconductor 3.20 (R 4.4.2)
#>  GenomeInfoDbData      1.2.13    2024-12-03 [1] Bioconductor
#>  GenomicRanges       * 1.58.0    2024-10-29 [1] Bioconductor 3.20 (R 4.4.1)
#>  ggplot2             * 3.5.2     2025-04-09 [1] CRAN (R 4.4.3)
#>  ggrepel             * 0.9.6     2024-09-07 [1] CRAN (R 4.4.1)
#>  globals               0.18.0    2025-05-08 [1] CRAN (R 4.4.3)
#>  glue                  1.8.0     2024-09-30 [1] CRAN (R 4.4.1)
#>  gower                 1.0.2     2024-12-17 [1] CRAN (R 4.4.2)
#>  GPfit                 1.0-9     2025-04-12 [1] CRAN (R 4.4.3)
#>  gridExtra             2.3       2017-09-09 [1] CRAN (R 4.4.1)
#>  gtable                0.3.6     2024-10-25 [1] CRAN (R 4.4.2)
#>  hardhat               1.4.2     2025-08-20 [1] CRAN (R 4.4.3)
#>  heatmaply             1.6.0     2025-07-12 [1] CRAN (R 4.4.3)
#>  hms                   1.1.3     2023-03-21 [1] CRAN (R 4.4.1)
#>  htmltools             0.5.8.1   2024-04-04 [1] CRAN (R 4.4.1)
#>  htmlwidgets           1.6.4     2023-12-06 [1] CRAN (R 4.4.1)
#>  httpuv                1.6.16    2025-04-16 [1] CRAN (R 4.4.3)
#>  httr                  1.4.7     2023-08-15 [1] CRAN (R 4.4.1)
#>  igraph                2.1.4     2025-01-23 [1] CRAN (R 4.4.2)
#>  inum                  1.0-5     2023-03-09 [1] CRAN (R 4.4.3)
#>  ipred                 0.9-15    2024-07-18 [1] CRAN (R 4.4.2)
#>  IRanges             * 2.40.1    2024-12-05 [1] Bioconductor 3.20 (R 4.4.2)
#>  iterators             1.0.14    2022-02-05 [1] CRAN (R 4.4.1)
#>  jsonlite              2.0.0     2025-03-27 [1] CRAN (R 4.4.3)
#>  KEGGREST              1.46.0    2024-10-29 [1] Bioconductor 3.20 (R 4.4.1)
#>  knitr                 1.50      2025-03-16 [1] CRAN (R 4.4.3)
#>  labeling              0.4.3     2023-08-29 [1] CRAN (R 4.4.0)
#>  later                 1.4.4     2025-08-27 [1] CRAN (R 4.4.3)
#>  lattice               0.22-7    2025-04-02 [2] CRAN (R 4.4.3)
#>  lava                  1.8.1     2025-01-12 [1] CRAN (R 4.4.2)
#>  lazyeval             0.2.2      2019-03-15 [1] CRAN (R 4.4.1)
#>  lhs                  1.2.0      2024-06-30 [1] CRAN (R 4.4.3)
#>  libcoin              1.0-10     2023-09-27 [1] CRAN (R 4.4.3)
#>  lifecycle            1.0.4      2023-11-07 [1] CRAN (R 4.4.1)
#>  limma                3.62.2     2025-01-09 [1] Bioconductor 3.20 (R 4.4.2)
#>  listenv              0.9.1      2024-01-29 [1] CRAN (R 4.4.2)
#>  locfit               1.5-9.12   2025-03-05 [1] CRAN (R 4.4.3)
#>  lubridate          * 1.9.4      2024-12-08 [1] CRAN (R 4.4.2)
#>  magrittr             2.0.3      2022-03-30 [1] CRAN (R 4.4.1)
#>  MASS                 7.3-65     2025-02-28 [2] CRAN (R 4.4.3)
#>  Matrix               1.7-3      2025-03-11 [2] CRAN (R 4.4.3)
#>  MatrixGenerics     * 1.18.1     2025-01-09 [1] Bioconductor 3.20 (R 4.4.2)
#>  matrixStats        * 1.5.0      2025-01-07 [1] CRAN (R 4.4.2)
#>  mclust               6.1.1      2024-04-29 [1] CRAN (R 4.4.3)
#>  memoise              2.0.1      2021-11-26 [1] CRAN (R 4.4.1)
#>  mgcv                 1.9-3      2025-04-04 [1] CRAN (R 4.4.3)
#>  mime                 0.13       2025-03-17 [1] CRAN (R 4.4.3)
#>  miniUI               0.1.2      2025-04-17 [1] CRAN (R 4.4.3)
#>  mixOmics             6.30.0     2024-10-29 [1] Bioconductor 3.20 (R 4.4.1)
#>  mvtnorm              1.3-3      2025-01-10 [1] CRAN (R 4.4.2)
#>  NeuralNetTools       1.5.3      2022-01-06 [1] CRAN (R 4.4.3)
#>  nlme                 3.1-168    2025-03-31 [2] CRAN (R 4.4.3)
#>  nnet                 7.3-20     2025-01-01 [1] CRAN (R 4.4.2)
#>  parallelly           1.45.1     2025-07-24 [1] CRAN (R 4.4.3)
#>  parsnip              1.3.3      2025-08-31 [1] CRAN (R 4.4.3)
#>  partykit             1.2-24     2025-05-02 [1] CRAN (R 4.4.3)
#>  pillar               1.11.0     2025-07-04 [1] CRAN (R 4.4.3)
#>  pkgbuild             1.4.8      2025-05-26 [1] CRAN (R 4.4.3)
#>  pkgconfig            2.0.3      2019-09-22 [1] CRAN (R 4.4.1)
#>  pkgload              1.4.0      2024-06-28 [1] CRAN (R 4.4.1)
#>  plotly               4.11.0     2025-06-19 [1] CRAN (R 4.4.3)
#>  plsmod               1.0.0      2022-09-06 [1] CRAN (R 4.4.3)
#>  plyr                 1.8.9      2023-10-02 [1] CRAN (R 4.4.1)
#>  png                  0.1-8      2022-11-29 [1] CRAN (R 4.4.0)
#>  praznik              11.0.0     2022-05-20 [1] CRAN (R 4.4.3)
#>  prodlim              2025.04.28 2025-04-28 [1] CRAN (R 4.4.3)
#>  profvis              0.4.0      2024-09-20 [1] CRAN (R 4.4.1)
#>  promises             1.3.3      2025-05-29 [1] CRAN (R 4.4.3)
#>  purrr              * 1.1.0      2025-07-10 [1] CRAN (R 4.4.3)
#>  R6                   2.6.1      2025-02-15 [1] CRAN (R 4.4.3)
#>  rARPACK              0.11-0     2016-03-10 [1] CRAN (R 4.4.3)
#>  RColorBrewer         1.1-3      2022-04-03 [1] CRAN (R 4.4.0)
#>  Rcpp                 1.1.0      2025-07-02 [1] CRAN (R 4.4.3)
#>  readr              * 2.1.5      2024-01-10 [1] CRAN (R 4.4.1)
#>  recipes              1.3.1      2025-05-21 [1] CRAN (R 4.4.3)
#>  registry             0.5-1      2019-03-05 [1] CRAN (R 4.4.0)
#>  remotes              2.5.0      2024-03-17 [1] CRAN (R 4.4.1)
#>  reshape2             1.4.4      2020-04-09 [1] CRAN (R 4.4.1)
#>  rlang                1.1.6      2025-04-11 [1] CRAN (R 4.4.3)
#>  rmarkdown            2.29       2024-11-04 [1] CRAN (R 4.4.2)
#>  rpart                4.1.24     2025-01-07 [2] CRAN (R 4.4.3)
#>  rsample              1.3.1      2025-07-29 [1] CRAN (R 4.4.3)
#>  RSpectra             0.16-2     2024-07-18 [1] CRAN (R 4.4.2)
#>  RSQLite                 2.4.3      2025-08-20 [1] CRAN (R 4.4.3)
#>  rstudioapi              0.17.1     2024-10-22 [1] CRAN (R 4.4.2)
#>  rules                   1.0.2      2023-03-08 [1] CRAN (R 4.4.3)
#>  S4Arrays                1.6.0      2024-10-29 [1] Bioconductor 3.20 (R 4.4.1)
#>  S4Vectors             * 0.44.0     2024-10-29 [1] Bioconductor 3.20 (R 4.4.1)
#>  scales                * 1.4.0      2025-04-24 [1] CRAN (R 4.4.3)
#>  seriation               1.5.8      2025-08-20 [1] CRAN (R 4.4.3)
#>  sessioninfo             1.2.3      2025-02-05 [1] CRAN (R 4.4.3)
#>  shiny                   1.11.1     2025-07-03 [1] CRAN (R 4.4.3)
#>  SparseArray             1.6.2      2025-02-20 [1] Bioconductor 3.20 (R 4.4.2)
#>  statmod                 1.5.0      2023-01-06 [1] CRAN (R 4.4.2)
#>  stringi                 1.8.7      2025-03-27 [1] CRAN (R 4.4.3)
#>  stringr               * 1.5.1      2023-11-14 [1] CRAN (R 4.4.1)
#>  SummarizedExperiment * 1.36.0     2024-10-29 [1] Bioconductor 3.20 (R 4.4.1)
#>  survival                3.8-3      2024-12-17 [1] CRAN (R 4.4.2)
#>  sva                     3.54.0     2024-10-29 [1] Bioconductor 3.20 (R 4.4.1)
#>  TENTACLES             * 0.1.0      2025-10-30 [1] Github (Giomu/TENTACLES@199e3f7)
#>  tibble                * 3.3.0      2025-06-08 [1] CRAN (R 4.4.3)
#>  tidyr                 * 1.3.1      2024-01-24 [1] CRAN (R 4.4.1)
#>  tidyselect              1.2.1      2024-03-11 [1] CRAN (R 4.4.1)
#>  tidyverse             * 2.0.0      2023-02-22 [1] CRAN (R 4.4.1)
#>  timechange              0.3.0      2024-01-18 [1] CRAN (R 4.4.1)
#>  timeDate                4041.110   2024-09-22 [1] CRAN (R 4.4.1)
#>  TSP                     1.2-5      2025-05-27 [1] CRAN (R 4.4.3)
#>  tune                    2.0.0      2025-09-01 [1] CRAN (R 4.4.3)
#>  tzdb                    0.5.0      2025-03-15 [1] CRAN (R 4.4.3)
#>  UCSC.utils              1.2.0      2024-10-29 [1] Bioconductor 3.20 (R 4.4.1)
#>  urlchecker              1.0.1      2021-11-30 [1] CRAN (R 4.4.1)
#>  usethis                 3.2.0      2025-08-28 [1] CRAN (R 4.4.3)
#>  vctrs                   0.6.5      2023-12-01 [1] CRAN (R 4.4.1)
#>  viridis                 0.6.5      2024-01-29 [1] CRAN (R 4.4.1)
#>  viridisLite             0.4.2      2023-05-02 [1] CRAN (R 4.4.1)
#>  webshot                 0.5.5      2023-06-26 [1] CRAN (R 4.4.3)
#>  withr                   3.0.2      2024-10-28 [1] CRAN (R 4.4.2)
#>  workflows               1.3.0      2025-08-27 [1] CRAN (R 4.4.3)
#>  xfun                    0.53       2025-08-19 [1] CRAN (R 4.4.3)
#>  XML                     3.99-0.19  2025-08-22 [1] CRAN (R 4.4.3)
#>  xtable                  1.8-4      2019-04-21 [1] CRAN (R 4.4.1)
#>  XVector                 0.46.0     2024-10-29 [1] Bioconductor 3.20 (R 4.4.1)
#>  yaml                    2.3.10     2024-07-26 [1] CRAN (R 4.4.1)
#>  yardstick               1.3.2      2025-01-22 [1] CRAN (R 4.4.3)
#>  zlibbioc                1.52.0     2024-10-29 [1] Bioconductor 3.20 (R 4.4.1)
#>
#>  [1] C:/Users/gabri/AppData/Local/R/win-library/4.4
#>  [2] C:/Program Files/R/R-4.4.3/library
#>  * -- Packages attached to the search path.
#>
#> ------------------------------------------------------------------------------
```