

# Package ‘RFGeneRank’

May 8, 2026

**Title** RFGeneRank: Cross-validated Stable Predictive Gene Ranking for Transcriptomics

**Version** 1.0.0

**Description** Tools to harmonize bulk RNA-seq matrices, optionally apply batch correction, and train cross-validated classification models using ranger, glmnet, or xgboost. Supports leakage-safe feature selection, permutation importance, SHAP-based interpretability, and calibration methods (Platt or isotonic). Provides stability metrics across folds, embeddings (PCA/UMAP), ROC visualization, SHAP dependence plots, and tidy ranked-gene tables for downstream analysis.

**License** MIT + file LICENSE

**URL** <https://github.com/Abdulaziz-Albeshri/RFGeneRank>

**BugReports** <https://github.com/Abdulaziz-Albeshri/RFGeneRank/issues>

**Depends** R (>= 4.6.0)

**Imports** ggplot2, limma, methods, pROC, ranger, stats,  
SummarizedExperiment, sva, AnnotationDbi, umap, scales, utils,  
S4Vectors, digest, mgcv, Matrix, glmnet, xgboost, patchwork

**Suggests** GEOquery, Biobase, edgeR, uwot, BiocStyle, knitr,  
org.Hs.eg.db, rmarkdown, DESeq2, MASS, matrixStats,  
BiocGenerics, fastshap, caret, testthat (>= 3.0.0), covr

**Config/testthat/edition** 3

**VignetteBuilder** knitr

**biocViews** Transcriptomics, RNASeq, GeneExpression, FeatureExtraction,  
Classification, Visualization, Software, StatisticalMethod,  
Alignment

**Encoding** UTF-8

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.3

**Collate** 'globals.R' 'class\_def.R' 'accessors.R' 'accessors\_public.R'  
'check\_utils.R' 'batch\_correct.R' 'calibrate.R'  
'combat\_helpers.R' 'compute\_class\_weights.R' 'confounding.R'  
'crossval.R' 'id\_map.R' 'plots\_all.R' 'prepare\_data.R'

'align\_datasets.R' 'rank\_genes.R' 'top\_genes.R' 'Rzzz.R'  
 'package-doc.R' 'sign\_importance.R' 'factor\_dependence.R'  
 'validate\_genes.R'

**NeedsCompilation** no

**git\_url** <https://git.bioconductor.org/packages/RFGeneRank>

**git\_branch** RELEASE\_3\_23

**git\_last\_commit** 9a749d0

**git\_last\_commit\_date** 2026-04-28

**Repository** Bioconductor 3.23

**Date/Publication** 2026-05-07

**Author** Abdulaziz Albeshri [aut, cre] (ORCID:  
 <<https://orcid.org/0000-0002-0718-2439>>),  
 Thamer Ahmad Bouback [ctb],  
 Majid Al-Zahrani [ctb],  
 Tasneem Alsahafi [ctb]

**Maintainer** Abdulaziz Albeshri <a.z.a1410@hotmail.com>

## Contents

align_datasets . . . . .	3
apply_calibration . . . . .	5
calibrate_oof . . . . .	5
factor_dependence . . . . .	6
GeneRankFit-accessors . . . . .	8
id_map . . . . .	10
plot_confusion_heatmap . . . . .	11
plot_embed . . . . .	11
plot_embed_expr . . . . .	13
plot_importance . . . . .	14
plot_roc . . . . .	15
plot_roc_multi . . . . .	16
plot_shap_dependence . . . . .	17
plot_sign_importance . . . . .	19
prepare_data . . . . .	20
rank_genes . . . . .	22
RFGeneRank . . . . .	24
rfgr_crossval . . . . .	24
rfgr_plot_suite . . . . .	26
shap_train_ranger . . . . .	28
sign_importance . . . . .	29
top_genes . . . . .	31
validate_genes . . . . .	32

**Index**

**34**

---

align_datasets	<i>Align and merge expression + metadata (genes by intersection; strict sample match)</i>
----------------	---

---

### Description

align\_datasets() ingests parallel lists of expression tables (matrices/data.frames) and metadata data.frames, cleans them, enforces strict sample alignment (rownames(metadata) == colnames(expression)), drops metadata rows with any NA (with a warning + structured report), merges by the intersection of genes, and returns the merged expression, metadata, and a SummarizedExperiment with gene IDs locked into rowData(se)\$gene\_id.

No regex recoding is performed. If a dataset lacks a batch column and require\_batch=TRUE, a per-dataset batch factor ("Batch1", "Batch2", ...) is created.

### Usage

```
align_datasets(
  expr_list,
  meta_list,
  prefer = NULL,
  require_batch = TRUE,
  tag_dataset = TRUE,
  gene_merge = "intersection",
  verbose = TRUE
)
```

### Arguments

expr_list	list of matrices or data.frames; rows = genes, cols = samples. If a data.frame contains a gene-ID column, it will be moved into rownames. All expression values are coerced to numeric (double).
meta_list	list of data.frames; rows = samples (must be in rownames). Must contain a state column and (if require_batch=TRUE) a batch column. If batch is missing, it is auto-created per dataset as "Batch1", "Batch2", etc.
prefer	optional selector/renamer for metadata columns. Two forms are supported: <ul style="list-style-type: none"> <li>• index form: c("2:state", "5:batch") (select column 2 and 5, rename to state, batch)</li> <li>• name map: c("state=phenotype", "batch=plate") (select phenotype, rename to state, etc.) Accepts either a single character vector applied to all datasets, or a named list specifying datasets individually. with one vector per dataset (names must match names(expr_list)/names(meta_list)). No value recoding is performed.</li> </ul>
require_batch	logical (default TRUE). If TRUE and a dataset lacks batch, create a per-dataset batch factor ("Batch1", "Batch2", etc.).
tag_dataset	logical (default TRUE). If TRUE, add a dataset factor ("ds1", "ds2", ...) to the merged metadata.

**gene\_merge** character(1), default "intersection". Currently only intersection is supported (keeps only genes shared by all datasets).

**verbose** logical. If TRUE (default), prints a brief summary and per-dataset stats.

## Value

A list with:

**expr** Merged numeric matrix (genes x samples) with syntactic, unique gene IDs.

**metadata** Merged metadata (samples as rownames) with state/batch as factors.

**se** A SummarizedExperiment with assay "expr", colData = metadata, and rowData(se)\$gene\_id set to current rownames.

**report** List with per-dataset input/kept counts, NA-drop details, observed levels, and final shared-gene count.

Align and merge expression + metadata ...

## Examples

```
# Single toy dataset: expression matrix (genes x samples)
expr <- matrix(
  rnorm(5 * 4),
  nrow = 5,
  dimnames = list(
    paste0("gene", 1:5),
    paste0("s1_", 1:4)
  )
)

# Matching metadata: one row per sample, with 'state' and 'batch' columns
meta <- data.frame(
  state = rep("Control", ncol(expr)),
  batch = rep("A", ncol(expr))
)
rownames(meta) <- colnames(expr)

# Lists of length 1 for expression and metadata
expr_list <- list(A = expr)
meta_list <- list(A = meta)

aligned <- align_datasets(
  expr_list = expr_list,
  meta_list = meta_list
)
str(aligned)
```

---

apply_calibration	<i>Apply stored calibration to a numeric vector of probabilities</i>
-------------------	--

---

**Description**

Apply stored calibration to a numeric vector of probabilities

**Usage**

```
apply_calibration(fit, p)
```

**Arguments**

fit	GeneRankFit with stored calibration information.
p	numeric vector of positive-class probabilities

**Value**

numeric vector of calibrated probabilities (or original if none stored)

**Examples**

```
# Example probabilities
p <- runif(5)

# Minimal GeneRankFit without calibration (returns original p)
fit0 <- methods::new("GeneRankFit")
apply_calibration(fit0, p)
```

---

calibrate_oof	<i>Calibrate out-of-fold probabilities (isotonic or Platt)</i>
---------------	--

---

**Description**

Fits a calibration model on the out-of-fold (OOF) positive-class probabilities and stores it inside the GeneRankFit object. No data leakage: uses OOF only.

**Usage**

```
calibrate_oof(fit, method = c("isotonic", "platt"))
```

**Arguments**

fit	GeneRankFit containing out-of-fold predictions and labels.
method	"isotonic" or "platt"

**Value**

GeneRankFit with stored calibration information.

**Examples**

```
# Example probabilities
p <- runif(5)

# Case 1: no calibration stored → returns original p
fit0 <- methods::new("GeneRankFit")
apply_calibration(fit0, p)

# Case 2: simple calibration function (illustration)
fit1 <- methods::new("GeneRankFit")
calibration(fit1) <- list(fun = function(x) x^0.8)
apply_calibration(fit1, p)
```

---

factor_dependence	<i>Covariate dependence of gene contributions (SHAP/proxy; "expr" assay)</i>
-------------------	--

---

**Description**

Covariate dependence of gene contributions (SHAP/proxy; "expr" assay)

**Usage**

```
factor_dependence(
  fit,
  se,
  covariates,
  method = c("shap", "proxy"),
  ngenes = 500L,
  gene_selection = c("importance", "variance"),
  nsim = 128L,
  bg_per_class = 64L,
  cache_dir = NULL,
  fdr_method = "BH",
  seed = 1L,
  pred_fun = NULL,
  pos_label = NULL,
  positive = NULL
)
```

**Arguments**

<code>fit</code>	Trained object containing out-of-fold predictions (n x k) and labels. For S4 wrappers (e.g., GeneRankFit), the finalized learner should be in A final fitted model and the training feature names may also be present. this function will train a temporary full-data model on-the-fly (not saved).
<code>se</code>	SummarizedExperiment with assay "expr" (genes x samples).
<code>covariates</code>	Character vector of covariate names in colData(se), e.g. c("sex","age").
<code>method</code>	"shap" (default; uses fastshap if available) or "proxy".
<code>ngenes</code>	Number of genes to test (default 500). Use "ALL" or NULL for all genes.
<code>gene_selection</code>	"importance" (default) or "variance".
<code>nsim</code>	SHAP Monte-Carlo permutations (default 128).
<code>bg_per_class</code>	Max background samples per class for SHAP (default 64).
<code>cache_dir</code>	Cache directory for SHAP matrices; default R user cache dir.
<code>fdr_method</code>	Multiple-testing correction (default "BH").
<code>seed</code>	RNG seed (default 1).
<code>pred_fun</code>	OPTIONAL user predictor: function(newdata) -> numeric p(positive). If provided, it takes precedence over the built-in predictor.
<code>pos_label</code>	OPTIONAL (legacy) positive class label; kept for back-compat.
<code>positive</code>	OPTIONAL override for positive class. Either a class label (character) or an index 1/2. If provided, it overrides pos_label.

**Value**

data.frame with columns: gene, covariate, test, stat, pval, fdr, effect, dependent. Attributes: "method","assay","genes\_used",

**Examples**

```
#' # For reproducibility, set a fixed seed such as set.seed(1) before running this example.

# Tiny expression matrix
expr <- matrix(stats::rnorm(30), nrow = 6)
rownames(expr) <- paste0("gene", 1:6)
colnames(expr) <- paste0("sample", 1:5)

# Covariates
cd <- data.frame(
  label = factor(c("A", "A", "B", "B", "B")),
  sex   = factor(c("M", "F", "F", "M", "M")),
  age   = c(30, 40, 35, 50, 60)
)

se <- SummarizedExperiment::SummarizedExperiment(
  assays = list(expr = expr),
  colData = cd
)
```

```

# Minimal mock GeneRankFit object with required slots
prob_mat <- matrix(
  stats::runif(5),
  ncol = 1,
  dimnames = list(colnames(se), "B")
)

fit <- methods::new(
  "GeneRankFit",
  oof = list(
    prob = prob_mat,
    y     = cd$label
  )
)

```

---

GeneRankFit-accessors *Accessors for GeneRankFit*

---

## Description

Getter/setter accessors for GeneRankFit slots.

## Usage

```

params(x)

## S4 method for signature 'GeneRankFit'
params(x)

oof(x)

## S4 method for signature 'GeneRankFit'
oof(x)

imp(x)

## S4 method for signature 'GeneRankFit'
imp(x)

imp(x) <- value

## S4 replacement method for signature 'GeneRankFit'
imp(x) <- value

calibration(x)

## S4 method for signature 'GeneRankFit'

```

```

calibration(x)

calibration(x) <- value

## S4 replacement method for signature 'GeneRankFit'
calibration(x) <- value

```

### Arguments

x	A GeneRankFit object.
value	Replacement value.

### Value

For getters:

params(x) A list of training parameters and metadata.  
 oof(x) A list of out-of-fold results including predictions and labels.  
 imp(x) A data.frame containing aggregated variable importance scores.  
 calibration(x) A list describing calibration settings.

For setters:

imp(x) <- value Returns the updated GeneRankFit object with modified importance data.  
 calibration(x) <- value Returns the updated GeneRankFit object with modified calibration settings.

### Examples

```

fit <- methods::new("GeneRankFit")

# getters
params(fit)
oof(fit)
imp(fit)
calibration(fit)

# setters
imp(fit) <- data.frame(
  gene = c("GeneA", "GeneB"),
  importance = c(0.8, 0.3)
)
calibration(fit) <- list(
  method = "platt",
  fun = function(p) p
)

imp(fit)
calibration(fit)

```

---

`id_map`*Map gene identifiers using AnnotationDbi*

---

### Description

A thin wrapper around `AnnotationDbi::select()` that preserves input order and enables straightforward conversion between identifier types (e.g., ENTREZID → SYMBOL).

### Usage

```
id_map(  
  keys,  
  from,  
  to,  
  OrgDb = org.Hs.eg.db::org.Hs.eg.db,  
  drop_na = TRUE,  
  unique = TRUE  
)
```

### Arguments

<code>keys</code>	character vector of gene IDs to map (e.g., ENTREZ IDs).
<code>from</code>	source keytype (e.g., "ENTREZID", "ENSEMBL", "SYMBOL").
<code>to</code>	destination keytype (e.g., "SYMBOL").
<code>OrgDb</code>	an <code>OrgDb</code> object. Defaults to <code>org.Hs.eg.db</code> .
<code>drop_na</code>	logical; if TRUE, drop rows with missing mapped values.
<code>unique</code>	logical; if TRUE, keep at most one mapping per input key, preferring the first match returned by <code>AnnotationDbi::select()</code> .

### Value

data.frame with columns `from`, `to` in that order.

### Examples

```
if (requireNamespace("org.Hs.eg.db", quietly = TRUE)) {  
  x <- c("7157", "7158") # ENTREZ IDs  
  id_map(x, from = "ENTREZID", to = "SYMBOL")  
}
```

---

`plot_confusion_heatmap`*Confusion-matrix heatmap*

---

**Description**

Confusion-matrix heatmap

**Usage**`plot_confusion_heatmap(cm, mode = c("counts", "rowpct"))`**Arguments**

<code>cm</code>	A 2x2 table as returned by <code>val\$&lt;method&gt;\$conf_mat</code> .
<code>mode</code>	Either "counts" or "rowpct".

**Value**

A ggplot object.

**Examples**

```
# For reproducibility, specify a fixed seed (e.g., set.seed(1)) before running this example.

true <- factor(rep(c("A", "B"), each = 5))
pred <- factor(sample(c("A", "B"), 10, replace = TRUE))

tab <- table(True = true, Predicted = pred)

plot_confusion_heatmap(tab)
```

---

`plot_embed`*Decision-space embedding (OOF probability space): PCA or UMAP*

---

**Description**Embeds samples using the out-of-fold (OOF) probability space (2D). Good for inspecting decision geometry; for biology structure, prefer `plot_embed_expr()`.

## Usage

```
plot_embed(  
  fit,  
  type = c("pca", "umap"),  
  engine = c("umap", "uwot"),  
  neighbors = NULL,  
  min_dist = NULL,  
  metric = NULL,  
  seed = NULL,  
  point_size = 2,  
  show_legend = TRUE,  
  palette = NULL  
)
```

## Arguments

fit	GeneRankFit (must have OOF probabilities).
type	"pca" or "umap".
engine	UMAP engine if type="umap" ("umap" or "uwot").
neighbors, min_dist, metric, seed	UMAP params.
point_size	numeric
show_legend	logical
palette	optional manual color palette

## Value

A ggplot object.

## Examples

```
# For reproducibility, specify a fixed seed (e.g., set.seed(1)) before running this example.  
  
# Toy 2D embedding for 10 samples  
embed_df <- data.frame(  
  sample = paste0("sample", 1:10),  
  dim1 = rnorm(10),  
  dim2 = rnorm(10),  
  label = factor(rep(c("A", "B"), each = 5))  
)  
  
head(embed_df)
```

---

plot\_embed\_expr      *Expression-space embedding (PCA or UMAP) of top RF genes*

---

### Description

Plots samples in the original expression space using the top RF-ranked genes. For PCA, axis labels include the percent variance explained. For UMAP, the title shows the engine and key parameters used.

### Usage

```
plot_embed_expr(
  fit,
  se,
  n_top = 100,
  type = c("umap", "pca"),
  engine = c("umap", "uwot"),
  neighbors = 15,
  min_dist = 0.1,
  metric = "euclidean",
  zscore = TRUE,
  seed = NULL,
  point_size = 2,
  show_legend = TRUE,
  palette = NULL
)
```

### Arguments

fit	GeneRankFit
se	SummarizedExperiment with assay "expr"
n_top	integer; number of top RF genes to use (default 100)
type	"umap" or "pca"
engine	UMAP engine, "umap" or "uwot" (used when type="umap")
neighbors, min_dist, metric	UMAP parameters
zscore	Logical; z-score samples x genes matrix before embedding (default TRUE)
seed	RNG seed for UMAP reproducibility (default uses stored pipeline seed if available).
point_size	numeric
show_legend	logical
palette	optional named vector of colors (names must match levels)

**Value**

A ggplot object.

**Examples**

```
# For reproducibility, specify a fixed seed (e.g., set.seed(1)) before running this example.

# Toy expression: 15 genes × 8 samples
expr <- matrix(rnorm(15 * 8), nrow = 15)
rownames(expr) <- paste0("gene", 1:15)
colnames(expr) <- paste0("sample", 1:8)

# Binary labels for samples
label <- factor(rep(c("A", "B"), each = 4))

se <- SummarizedExperiment::SummarizedExperiment(
  assays = list(expr = expr),
  colData = data.frame(label = label)
)

se
```

---

plot\_importance

*Feature importance (top genes)*

---

**Description**

Uses fold-normalized mean importances aggregated across CV folds. To display gene SYMBOLs on the axes, set `map_to_symbol = TRUE`. (requires `org.Hs.eg.db` and `RFGeneRank::top_genes()`).

**Usage**

```
plot_importance(
  fit,
  top = 30,
  map_to_symbol = FALSE,
  from = "ENTREZID",
  to = "SYMBOL"
)
```

**Arguments**

<code>fit</code>	GeneRankFit from <code>rank_genes()</code> .
<code>top</code>	Integer; number of genes to display (default 30).
<code>map_to_symbol</code>	Logical; map gene IDs to symbols if available (default FALSE).
<code>from</code>	Keytype for input IDs (default "ENTREZID").
<code>to</code>	Keytype for output labels (default "SYMBOL").

**Value**

A ggplot object.

**Examples**

```
# Toy expression matrix: genes x samples
expr <- matrix(
  rnorm(10 * 20),
  nrow = 10,
  dimnames = list(
    paste0("gene", 1:10),
    paste0("sample", 1:20)
  )
)

# Binary phenotype stored in 'label' column
y <- factor(rep(c("Control", "Case"), each = 10))

se <- SummarizedExperiment::SummarizedExperiment(
  assays = list(expr = expr),
  colData = data.frame(
    label = y,
    row.names = colnames(expr)
  )
)

# Fit a small GeneRank model (note: no 'genes' argument)
fit <- rank_genes(
  se = se,
  label_col = "label",
  n_top = 10,
  trees = 200
)

# Plot feature importance for the top-ranked genes
plot_importance(fit)
```

---

plot\_roc

*ROC curve from OOF probabilities (single model)*

---

**Description**

ROC curve from OOF probabilities (single model)

**Usage**

plot\_roc(fit)

**Arguments**

`fit` GeneRankFit with stored out-of-fold predictions and labels.

**Value**

A ggplot2 object containing the ROC curve derived from

**Examples**

```
expr <- matrix(
  rnorm(10 * 20),
  nrow = 10,
  dimnames = list(
    paste0("gene", 1:10),
    paste0("sample", 1:20)
  )
)

y <- factor(rep(c("Control", "Case"), each = 10))

se <- SummarizedExperiment::SummarizedExperiment(
  assays = list(expr = expr),
  colData = data.frame(state = y)
)

fit <- rank_genes(se, label_col = "state", trees = 100)
plot_roc(fit)
```

---

plot\_roc\_multi

*Multi-model ROC (guardrailed OOF)*

---

**Description**

Overlays ROC curves for one or more fits that expose \$oof with p\_use and y. Works with validate\_genes() outputs.

**Usage**

```
plot_roc_multi(fits, title = "ROC (OOF, guardrailed)")
```

**Arguments**

`fits` Named list of model results (e.g., list(ranger=val\$ranger, ...)).

`title` Character plot title.

**Value**

A ggplot object.

**Examples**

```
# Toy binary outcome
y <- factor(rep(c("Control", "Case"), each = 10))

# Positive-class probabilities (Case = positive class)
p_use1 <- runif(20)
p_use2 <- runif(20)

# Create minimal fit objects expected by plot_roc_multi()
fits <- list(
  Model_1 = list(
    oof = list(
      y = y,
      p_use = p_use1
    )
  ),
  Model_2 = list(
    oof = list(
      y = y,
      p_use = p_use2
    )
  )
)

plot_roc_multi(fits)
```

---

plot\_shap\_dependence *SHAP dependence scatter for one gene (cached-first, robust, fast)*

---

**Description**

SHAP dependence scatter for one gene (cached-first, robust, fast)

**Usage**

```
plot_shap_dependence(
  fit,
  se,
  gene,
  x = "age",
  color_by = NULL,
  palette = NULL,
  shap_mat = NULL,
  nsim = 64,
  pos_label = NULL,
  age_band_width = 10,
  band_alpha = 0.12,
  band_mode = c("significant", "top_k"),
  top_k = 1
)
```

**Arguments**

<code>fit</code>	GeneRankFit containing out-of-fold predictions and labels.
<code>se</code>	SummarizedExperiment with assay "expr".
<code>gene</code>	Character gene ID present in assay(se,"expr").
<code>x</code>	Covariate name in colData(se) for x-axis (default "age").
<code>color_by</code>	Optional grouping column name in colData(se); if NULL, no colour grouping.
<code>palette</code>	Optional character vector of colour names or hex codes. If <code>color_by</code> is: <ul style="list-style-type: none"> <li>• numeric: <code>palette</code> (length <math>\geq 2</math>) is used as a continuous gradient via <code>scale_color_gradientn()</code>.</li> <li>• categorical: <code>palette</code> is recycled/trimmed to the number of levels and used via <code>scale_color_manual()</code>.</li> </ul> <p>If <code>palette</code> is NULL, ggplot2 defaults are used, except for <code>color_by = "sex" / "sex_clean"</code> where a fixed pink/cyan palette is used.</p>
<code>shap_mat</code>	Optional SHAP matrix (rows = samples, cols = genes).
<code>nsim</code>	Fastshap nsim if auto-computing (default 64).
<code>pos_label</code>	Positive class label (defaults to level 2 of outcome).
<code>age_band_width</code>	Width of shaded age bands (years) for sex plots.
<code>band_alpha</code>	Alpha of bands.
<code>band_mode</code>	"significant" (FDR $\leq$ 0.05) or "top_k".
<code>top_k</code>	If <code>band_mode="top_k"</code> , how many bins per group to shade.

**Value**

ggplot object.

**Examples**

```

expr <- matrix(
  rnorm(10 * 20),
  nrow = 10,
  dimnames = list(
    paste0("gene", 1:10),
    paste0("sample", 1:20)
  )
)

y <- factor(rep(c("Control", "Case"), each = 10))

se <- SummarizedExperiment::SummarizedExperiment(
  assays = list(expr = expr),
  colData = data.frame(
    state = y,
    age = seq_len(20) + 40
  )
)

```

```

fit <- rank_genes(se, label_col = "state", trees = 50)

# Minimal SHAP-like matrix: one column named by the gene
shap_mat <- matrix(
  rnorm(20),
  nrow = 20,
  dimnames = list(NULL, "gene1")
)

plot_shap_dependence(
  fit      = fit,
  se      = se,
  gene    = "gene1",
  x      = "age",
  shap_mat = shap_mat
)

```

---

plot\_sign\_importance    *Signed feature importance (directional effect)*

---

### Description

Visualizes direction-aware importances produced by `sign_importance()`. If `tab` is `NULL`, the function will try to read the stored importance table and require a `signed_importance` column to be present there.

### Usage

```

plot_sign_importance(
  fit = NULL,
  tab = NULL,
  top = 30,
  map_to_symbol = FALSE,
  from = "ENTREZID",
  to = "SYMBOL",
  show_legend = TRUE,
  palette = NULL
)

```

### Arguments

<code>fit</code>	GeneRankFit (optional if <code>tab</code> is supplied)
<code>tab</code>	data.frame from <code>sign_importance()</code> with columns: <code>gene</code> , <code>importance</code> , <code>direction</code> (-1/0/1), <code>signed_importance</code>
<code>top</code>	Integer; number of genes to show (default 30)
<code>map_to_symbol</code>	Logical; map x-axis labels to SYMBOLs (default FALSE)
<code>from</code>	Keytype for input IDs (default "ENTREZID")

to                    Keytype for output labels (default "SYMBOL")

show\_legend        Logical; show legend (default TRUE)

palette            Optional named vector for fill colors, e.g. `c(`-1`="#3182bd", `1`="#de2d26", `0`="#9e9e9e")`

### Value

A ggplot object.

### Examples

```
# Toy signed importance table for 5 genes
signed_imp <- data.frame(
  gene      = paste0("gene", 1:5),
  importance = c(0.5, 0.4, 0.3, 0.2, 0.1),
  signed_effect = c(0.5, -0.4, 0.3, -0.2, 0.1)
)

head(signed_imp)
```

---

prepare_data	<i>Prepare matrices + metadata: align, (log) transform, batch correct, prefilter</i>
--------------	--

---

### Description

Prepare matrices + metadata: align, (log) transform, batch correct, prefilter

### Usage

```
prepare_data(
  mats,
  metas,
  label_col,
  batch_col = NULL,
  n_var = 5000,
  log1p = TRUE,
  batch_method = c("none", "combat", "limma", "combat_seq"),
  counts = FALSE,
  filter_in_cv = FALSE,
  batch_correction_scope = c("global", "fold"),
  batch_covariates = NULL
)
```

**Arguments**

<code>mats</code>	list of numeric matrices (genes-by-samples)
<code>metas</code>	list of data.frames (rownames = sample IDs)
<code>label_col</code>	outcome column in metadata
<code>batch_col</code>	batch column in metadata (or NULL)
<code>n_var</code>	keep top-N variable genes globally (unsupervised)
<code>log1p</code>	logical; log1p transform before variance filter
<code>batch_method</code>	"none", "combat", "limma", "combat_seq"
<code>counts</code>	logical; TRUE if raw counts (for combat_seq)
<code>filter_in_cv</code>	logical; if TRUE, skip global variance filter and let CV do it inside folds
<code>batch_correction_scope</code>	"global" or "fold" (fold correction happens inside CV)
<code>batch_covariates</code>	optional character vector of metadata column names used as covariates in batch correction

**Value**

SummarizedExperiment

**Examples**

```
set.seed(1)

expr <- matrix(stats::rnorm(20 * 10), nrow = 20)
rownames(expr) <- paste0("gene", 1:20)
colnames(expr) <- paste0("sample", 1:10)

label <- factor(rep(c("A", "B"), each = 5))
batch <- factor(rep(c("batch1", "batch2"), times = 5))

# Build lists of matrices + metadata as expected by prepare_data()
mats <- list(expr)
metas <- list(data.frame(
  label = label,
  batch = batch,
  row.names = colnames(expr)
))

prep <- prepare_data(
  mats      = mats,
  metas     = metas,
  label_col = "label",
  batch_col = "batch"
)
prep
```

---

rank\_genes                      *Rank genes with batch-aware cross-validation (UMAP-free)*

---

### Description

Performs CV on a SummarizedExperiment, aggregates fold-normalized feature importances, and stores out-of-fold predictions. Supports K-fold, LOBO (Leave-One-Batch-Out), and group-K by batch. Batch correction, filtering, transforms, and standardization are applied *inside folds* using TRAIN-only statistics. If auto\_confounds=TRUE, the function inspects batch~label association and will switch to LOBO and enable fold-safe batch correction if confounding is moderate/severe.

### Usage

```
rank_genes(
  se,
  label_col = "state",
  n_top = 500,
  k = 5,
  trees = 1000,
  importance = c("permutation", "impurity"),
  class_weights = NULL,
  threads = max(1, parallel::detectCores() - 1),
  seed = 1,
  fold_batch_correction = FALSE,
  batch_col = NULL,
  batch_covariates = NULL,
  filter_low_expr = FALSE,
  min_prop = 0.2,
  transform = c("none", "log1p"),
  standardize = FALSE,
  cv = c("kfold", "lobo", "groupk"),
  auto_confounds = TRUE
)
```

### Arguments

se	SummarizedExperiment
label_col	character; class-label column in colData(se)
n_top	integer; per-fold top-variance feature count (0 = off)
k	integer; number of folds (ignored by LOBO)
trees	integer; number of trees for ranger
importance	"permutation" or "impurity"
class_weights	named numeric vector or NULL (auto-computed if NULL and imbalance >= 1.5x)
threads	integer; CPU threads for ranger

seed integer; RNG seed

fold\_batch\_correction logical; if TRUE, do train-only batch removal per fold

batch\_col optional; name of batch column in colData(se)

batch\_covariates optional character vector of covariates for batch model

filter\_low\_expr logical; drop genes expressed (>0) in < min\_prop of TRAIN

min\_prop numeric in (0,1]; minimum TRAIN proportion to keep a gene

transform "none" or "log1p"

standardize logical; z-score by TRAIN mean/SD (applied to train+test)

cv "kfold", "lobo", or "groupk"

auto\_confounds logical; if TRUE, auto-switch to LOBO and enable fold batch-correction when confounded

**Value**

GeneRankFit S4 object

**Examples**

```
# For reproducibility, specify a fixed seed (e.g., set.seed(1)) before running this example.

# Toy expression: 20 genes × 12 samples
expr <- matrix(stats::rnorm(20 * 12), nrow = 20)
rownames(expr) <- paste0("gene", 1:20)
colnames(expr) <- paste0("sample", 1:12)

# Binary labels
label <- factor(rep(c("A", "B"), each = 6))

se <- SummarizedExperiment::SummarizedExperiment(
  assays = list(expr = expr),
  colData = data.frame(label = label)
)

# Rank genes with default settings
rg <- rank_genes(
  se = se,
  label_col = "label"
)

str(rg)
```

---

RFGeneRank

*RFGeneRank: CV-stable predictive ranking for transcriptomics*

---

### Description

Package-level documentation and imports.

### Author(s)

**Maintainer:** Abdulaziz Albeshri <a.z.a1410@hotmail.com> ([ORCID](#))

Other contributors:

- Thamer Ahmad Bouback <Tbouback@kau.edu.sa> [contributor]
- Majid Al-Zahrani <maalzahrani4@kau.edu.sa> [contributor]
- Tasneem Alsahafi <Talsahafi0015@stu.kau.edu.sa> [contributor]

### See Also

Useful links:

- <https://github.com/Abdulaziz-Albeshri/RFGeneRank>
- Report bugs at <https://github.com/Abdulaziz-Albeshri/RFGeneRank/issues>

---

rfgr\_crossval

*Batch-aware cross-validation with optional ComBat modes (incl. frozen ComBat)*

---

### Description

Orchestrates leakage-safe CV for RFGeneRank with two ComBat modes:

- "none": no batch correction
- "train": train-only batch correction with leakage-safe application to TEST

Supports LOBO (leave-one-batch-out), Group K-Fold by batch, and standard K-Fold.

**Usage**

```
rfgr_crossval(
  expr,
  metadata,
  label_col = "state",
  batch_col = "batch",
  covariates = NULL,
  cv = c("lobo", "groupk", "kfold"),
  k = 5,
  combat_mode = c("none", "train"),
  rf_trees = 1000,
  seed = 1,
  verbose = TRUE
)
```

**Arguments**

<code>expr</code>	numeric matrix genes x samples (continuous scale: log2CPM or log2(TPM+1))
<code>metadata</code>	data.frame with SampleID, label_col, batch_col
<code>label_col</code>	character; target column in metadata (e.g., "state")
<code>batch_col</code>	character; batch/dataset column in metadata (e.g., "batch")
<code>covariates</code>	character vector of column names to <i>preserve</i> in ComBat (added to design)
<code>cv</code>	one of c("lobo","groupk","kfold")
<code>k</code>	integer; number of folds for "groupk" or "kfold"
<code>combat_mode</code>	one of c("none","train")
<code>rf_trees</code>	integer; number of trees for ranger
<code>seed</code>	integer; RNG seed
<code>verbose</code>	logical; emit progress messages

**Value**

list(auc\_by\_fold, mean\_auc, settings, folds\_info)

**Examples**

```
# Minimal runnable example: tiny dataset, fast evaluation
expr <- matrix(rnorm(10 * 6), nrow = 10)
rownames(expr) <- paste0("gene", 1:10)
colnames(expr) <- paste0("sample", 1:6)

label <- factor(rep(c("A", "B"), each = 3))
batch <- factor(rep(c("batch1", "batch2"), times = 3))

metadata <- data.frame(
  label = label,
  batch = batch,
```

```

    row.names = colnames(expr)
  )

  # Lightweight cross-validation: no ComBat, few trees
  cv_res <- rfgr_crossval(
    expr      = expr,
    metadata  = metadata,
    label_col = "label",
    batch_col = "batch",
    cv        = "kfold",
    k         = 2,
    combat_mode = "none",
    rf_trees  = 10,
    verbose   = FALSE
  )

  cv_res

```

---

rfgr\_plot\_suite      *One-call plot suite (optional file export)*

---

## Description

Produces: importance, signed-importance, expression PCA, decision PCA, ROC overlay (if val provided), confusion heatmaps, and optional SHAP.

## Usage

```
rfgr_plot_suite(fit, se, val = NULL, outdir = NULL, top = 30, shap_gene = NULL)
```

## Arguments

fit	GeneRankFit from rank_genes().
se	SummarizedExperiment with assay "expr".
val	Optional result from validate_genes().
outdir	Optional directory to save PNGs (if not NULL).
top	Integer; number of genes in importance plots.
shap_gene	Optional gene ID for SHAP dependence.

## Value

(Invisibly) list of ggplot objects.

## Examples

```
# Toy expression matrix: genes x samples
expr <- matrix(
  rnorm(10 * 12),
  nrow = 10
)

# Use known human Entrez IDs as gene names so annotation works
rownames(expr) <- c(
  "1", # A1BG
  "2", # A2M
  "9", # NAT1
  "1956", # EGFR
  "2064", # ERBB2
  "5290", # PIK3CA
  "5728", # PTEN
  "7422", # VEGFA
  "1950", # EDN1
  "7157" # TP53
)

colnames(expr) <- paste0("sample", 1:12)

# Binary phenotype stored in 'label' column
label <- factor(rep(c("Control", "Case"), each = 6))

se <- SummarizedExperiment::SummarizedExperiment(
  assays = list(expr = expr),
  colData = data.frame(
    label = label,
    row.names = colnames(expr)
  )
)

# Fit a small GeneRank model
fit <- rank_genes(
  se = se,
  label_col = "label",
  n_top = 10,
  trees = 100
)

# For this example, ensure the stored importance table has signed_importance information
# required by plot_sign_importance().
imp0 <- imp(fit)
if (is.null(imp0)) {
  imp0 <- data.frame(
    gene = character(),
    importance = numeric(),
    direction = integer(),
    signed_importance = numeric(),
    stringsAsFactors = FALSE
  )
}
```

```

    )
  }
  if (!"direction" %in% colnames(imp0)) {
    imp0$direction <- 1L
  }
  if (!"signed_importance" %in% colnames(imp0)) {
    imp0$signed_importance <- imp0$importance * imp0$direction
  }
  imp(fit) <- imp0

  # Generate a suite of diagnostic plots (returned as a list of ggplot objects)
  plots <- rfgr_plot_suite(fit, se)

  # Inspect available plots
  names(plots)

```

---

shap\_train\_ranger      *Compute a SHAP matrix using a single ranger model*

---

### Description

Trains a single probability random forest on the selected genes and computes per-sample SHAP values with fastshap. Rows = samples (colData rows), cols = genes.

### Usage

```

shap_train_ranger(
  se,
  label_col = "state",
  genes,
  class_weights = NULL,
  num.trees = 500,
  seed = 1L,
  nsim = 64,
  pos_label = NULL
)

```

### Arguments

se	SummarizedExperiment with assay "expr" (genes x samples).
label_col	Outcome column in colData(se) (factor).
genes	Character vector of gene IDs (must match rownames of assay(se,"expr")).
class_weights	Optional named numeric vector of class weights; if NULL, inverse-frequency weights are used.
num.trees	Number of trees (default 500).
seed	RNG seed.
nsim	fastshap Monte Carlo samples (default 64).
pos_label	Positive class label (defaults to level 2 of y).

**Value**

Numeric matrix of SHAP values (n\_samples x length(genes)).

**Examples**

```
# For reproducibility, specify a fixed seed (e.g., set.seed(1)) before running this example.

# Toy feature matrix and binary outcome
X <- data.frame(
  x1 = stats::rnorm(20),
  x2 = stats::rnorm(20)
)
y <- factor(rep(c("A", "B"), each = 10))

# Inspect inputs
head(X)
```

---

sign_importance	<i>Signed feature importance for RFGeneRank models</i>
-----------------	--

---

**Description**

Add direction (+/-) to RF importance using group means ("mean"), external DE log2FC ("de"), or SHAP ("shap").

**Usage**

```
sign_importance(
  fit,
  X,
  y = NULL,
  method = c("mean", "de", "shap"),
  de_table = NULL,
  case_level = NULL,
  orientation = c("samples_by_genes", "genes_by_samples"),
  assay_name = NULL,
  shap_n = 50,
  seed = 1
)
```

**Arguments**

<code>fit</code>	Trained model object from <code>rank_genes()</code> or similar.
<code>X</code>	Expression (matrix/data.frame/SummarizedExperiment).
<code>y</code>	Factor labels (optional if retrievable from <code>fit</code> ).
<code>method</code>	<code>c("mean", "de", "shap")</code> . Default "mean".

de_table	data.frame with columns gene, log2FC (for method="de").
case_level	Positive class label (default = last level of y).
orientation	"samples_by_genes" or "genes_by_samples".
assay_name	SE assay name/index.
shap_n	Integer; number of Monte Carlo samples for SHAP.
seed	RNG seed.

**Value**

data.frame with gene, importance, direction, signed\_importance, mean\_case, mean\_ctrl, mean\_diff, log2FC, shap\_dir.

**Examples**

```
# Toy expression matrix: samples x genes
X <- matrix(
  rnorm(4 * 5),
  nrow = 4,
  dimnames = list(
    paste0("sample", 1:4),
    paste0("gene", 1:5)
  )
)

# Binary labels
y <- factor(c("Control", "Control", "Case", "Case"))

# Minimal "fit" object: a list with a numeric importance vector
fit <- list(
  importance = setNames(
    c(0.8, 0.6, 0.4, 0.2, 0.1),
    paste0("gene", 1:5)
  )
)

res <- sign_importance(
  fit = fit,
  X = X,
  y = y,
  method = "mean"
)

head(res)
# In practice, sign_importance() is called on a GeneRankFit object
# produced by the RFGeneRank workflow, for example:
#
# fit <- gene_rank(se, genes = rownames(se), ...)
# sig_imp <- sign_importance(fit, X = expr_matrix, y = outcome)
# head(sig_imp)
```

---

top_genes	<i>Extract top predictive genes from a GeneRankFit</i>
-----------	--

---

### Description

Returns a ranked list from the aggregated, fold-normalized importance stored in the fitted object. Optionally adds an ID mapping column (e.g., ENTREZID -> SYMBOL) using `id_map`.

### Usage

```
top_genes(
  fit,
  n = 100,
  map = FALSE,
  OrgDb = org.Hs.eg.db::org.Hs.eg.db,
  from = "SYMBOL",
  to = "SYMBOL"
)
```

### Arguments

<code>fit</code>	a GeneRankFit object (from <code>rank_genes()</code> ).
<code>n</code>	integer; number of top genes to return (default 100).
<code>map</code>	logical; if TRUE, map from -> to and add a mapped column.
<code>OrgDb</code>	an OrgDb object for mapping (default <code>org.Hs.eg.db</code> ).
<code>from</code>	source keytype used for the gene identifiers (e.g., "ENTREZID", "SYMBOL", "ENSEMBL").
<code>to</code>	destination keytype (e.g., "SYMBOL").

### Value

A list with:

**gene** character vector of top gene IDs in from keytype

**table** data.frame with columns gene, importance, SelectedInFolds, and optional mapped

### Examples

```
gene_scores <- data.frame(
  gene      = paste0("gene", 1:6),
  importance = c(5, 4, 3, 2, 1, 0)
)

fit <- methods::new("GeneRankFit", imp = gene_scores)

tg <- top_genes(fit, n = 3)
tg$gene
```

```

tg$table
# In practice, top_genes() is used on a GeneRankFit object.
# For example, after running a full RFGeneRank pipeline:
#
# fit <- gene_rank(se, genes = rownames(se), ...)
# head(top_genes(fit, n = 20))
#
# where 'fit' is a GeneRankFit containing gene importance scores.

```

---

validate\_genes

*Validate a ranked gene set with alternate learners via k-fold CV*


---

## Description

Validate a ranked gene set with alternate learners via k-fold CV

## Usage

```

validate_genes(
  se,
  genes,
  methods = c("glmnet", "xgboost", "ranger"),
  k = 5,
  seed = 1L,
  model_params = list(),
  label_col = "state",
  positive = NULL,
  calibrate = c("platt", "isotonic", "none"),
  thr_metric = c("youden", "f1", "cost"),
  cost = c(fp = 1, fn = 1)
)

```

## Arguments

se	A SummarizedExperiment whose assay is genes-by-samples.
genes	Character vector of gene IDs (must match rownames(assay(se))).
methods	Character vector; any of c("ranger", "glmnet", "xgboost").
k	Integer number of folds (default 5).
seed	Integer RNG seed for reproducibility.
model_params	Named list of per-method parameter lists, e.g. list( ranger = list(num.trees=1000, importance="none", nthread=1), glmnet = list(alpha=0.5, standardize="zscore", clip=8, eps_sd=1e-8, use_class_weights=TRUE, lambda="lambda.min"), xgboost = list(nrounds=400, eta=0.05, max_depth=4, nthread=1) ).
label_col	Column name in colData(se) with the class labels. Must be a binary factor (exactly 2 levels).
positive	Optional; the positive class label. If NULL, uses levels(y)[2].

calibrate	One of c("platt", "isotonic", "none"); applied per fold.
thr_metric	One of c("youden", "f1", "cost") for threshold selection.
cost	Named numeric vector c(fp=1, fn=1) if thr_metric == "cost".

### Value

A list with \$summary and one entry per method. Each entry contains: \$oof (data.frame with p\_raw, p\_cal, p\_use, y, fold), \$calibrator\_requested, \$calibrator\_used, \$threshold, \$conf\_mat, and \$metrics with both diagnostics (raw/cal) and final guardrailed metrics (auc\_final, ece\_final, brier\_final).

### Examples

```
# Toy expression matrix: 10 genes x 12 samples
expr <- matrix(
  stats::rnorm(10 * 12),
  nrow = 10,
  dimnames = list(
    paste0("gene", 1:10),
    paste0("sample", 1:12)
  )
)

# Binary labels as a factor (balanced)
label <- factor(rep(c("A", "B"), each = 6))

se <- SummarizedExperiment::SummarizedExperiment(
  assays = list(expr = expr),
  colData = S4Vectors::DataFrame(label = label)
)

# Use the first 5 genes as a toy signature
genes <- rownames(expr)[1:5]

# Validate using ranger only (fast and robust for examples)
res <- validate_genes(
  se = se,
  genes = genes,
  methods = "ranger",
  k = 3,
  seed = 1,
  label_col = "label"
)

# Inspect out-of-fold AUROC
res$summary$auc_final
```

# Index

## \* **internal**

- RFGeneRank, [24](#)
- [align\\_datasets, 3](#)
- [apply\\_calibration, 5](#)
- [calibrate\\_oof, 5](#)
- [calibration \(GeneRankFit-accessors\), 8](#)
- [calibration, GeneRankFit-method \(GeneRankFit-accessors\), 8](#)
- [calibration<- \(GeneRankFit-accessors\), 8](#)
- [calibration<- , GeneRankFit-method \(GeneRankFit-accessors\), 8](#)
- [factor\\_dependence, 6](#)
- [GeneRankFit-accessors, 8](#)
- [id\\_map, 10, 31](#)
- [imp \(GeneRankFit-accessors\), 8](#)
- [imp, GeneRankFit-method \(GeneRankFit-accessors\), 8](#)
- [imp<- \(GeneRankFit-accessors\), 8](#)
- [imp<- , GeneRankFit-method \(GeneRankFit-accessors\), 8](#)
- [oof \(GeneRankFit-accessors\), 8](#)
- [oof, GeneRankFit-method \(GeneRankFit-accessors\), 8](#)
- [params \(GeneRankFit-accessors\), 8](#)
- [params, GeneRankFit-method \(GeneRankFit-accessors\), 8](#)
- [plot\\_confusion\\_heatmap, 11](#)
- [plot\\_embed, 11](#)
- [plot\\_embed\\_expr, 13](#)
- [plot\\_importance, 14](#)
- [plot\\_roc, 15](#)
- [plot\\_roc\\_multi, 16](#)
- [plot\\_shap\\_dependence, 17](#)
- [plot\\_sign\\_importance, 19](#)
- [prepare\\_data, 20](#)
- [rank\\_genes, 22](#)
- [RFGeneRank, 24](#)
- [RFGeneRank-package \(RFGeneRank\), 24](#)
- [rfgr\\_crossval, 24](#)
- [rfgr\\_plot\\_suite, 26](#)
- [shap\\_train\\_ranger, 28](#)
- [sign\\_importance, 29](#)
- [top\\_genes, 31](#)
- [validate\\_genes, 32](#)