# Package ‘GeneSelectR’

January 20, 2025

**Title** 'GeneSelectR' - Comprehensive Feature Selection Workflow for Bulk RNAseq Datasets

**Version** 1.0.1

**Description** The workflow is a versatile R package designed for comprehensive feature selection in bulk RNAseq datasets. Its key innovation lies in the seamless integration of the 'Python' 'scikit-learn' (<https://scikit-learn.org/stable/index.html>) machine learning framework with R-based bioinformatics tools. 'GeneSelectR' performs robust Machine Learning-driven (ML) feature selection while leveraging 'Gene Ontology' (GO) enrichment analysis as described by Thomas PD et al. (2022) <doi:10.1002/pro.4218>, using 'clusterProfiler' (Wu et al., 2021) <doi:10.1016/j.xinn.2021.100141> and semantic similarity analysis powered by 'simplifyEnrichment' (Gu, Huebschmann, 2021) <doi:10.1016/j.gpb.2022.04.008>. This combination of methodologies optimizes computational and biological insights for analyzing complex RNAseq datasets.

**License** MIT + file LICENSE

**URL** https://github.com/dzhakparov/GeneSelectR

**BugReports** https://github.com/dzhakparov/GeneSelectR/issues

**Imports** cowplot (>= 1.1.1), dplyr (>= 1.1.0), ggplot2 (>= 3.4.2), glue (>= 1.6.2), magrittr (>= 2.0.3), methods (>= 4.2.2), RColorBrewer (>= 1.1.3), reshape2 (>= 1.4.4), reticulate (>= 1.28), rlang (>= 1.1.1), testthat (>= 3.0.0), tibble (>= 3.2.1), tidyr (>= 1.3.0), tmod (>= 0.50.13),

**Suggests** clusterProfiler (>= 4.6.2), GO.db (>= 3.17.0), knitr, rmarkdown, BiocManager (>= 1.30.21), UpSetR (>= 1.4.0), AnnotationHub (>= 3.8.0), ensembldb (>= 2.24.0), org.Hs.eg.db (>= 3.17.0)

**Enhances** simplifyEnrichment (>= 1.8.0)

**Config/testthat/edition** 3

**Encoding** UTF-8

**RoxygenNote** 7.2.3

**VignetteBuilder** knitr

**Depends** R (>= 3.5.0)

**NeedsCompilation** no

**Author** Damir Zhakparov [aut, cre] (<https://orcid.org/0000-0001-7175-0843>)

**Maintainer** Damir Zhakparov <dzhakparov@gmail.com>

**Repository** CRAN

**Date/Publication** 2024-02-03 14:00:05 UTC

# Contents

aggregate_feature_importances . . . 3
AnnotatedGeneLists-class . . . 3
annotate_gene_lists . . . 4
calculate_mean_cv_scores . . . 5
calculate_overlap_coefficients . . . 6
calculate_permutation_feature_importance . . . 7
check_python_modules_available . . . 8
compute_GO_child_term_metrics . . . 8
configure_environment . . . 10
create_conda_env . . . 10
create_pipelines . . . 11
create_test_metrics_df . . . 12
define_sklearn_modules . . . 12
enable_multiprocess . . . 13
evaluate_test_metrics . . . 14
GeneList-class . . . 14
GeneSelectR . . . 15
get_feature_importances . . . 18
GO_enrichment_analysis . . . 19
import_python_packages . . . 21
install_python_packages . . . 21
perform_grid_search . . . 22
PipelineResults-class . . . 23
pipeline_to_list . . . 24
plot_feature_importance . . . 25
plot_metrics . . . 26
plot_overlap_heatmaps . . . 27
plot_upset . . . 28
run_simplify_enrichment . . . 29
set_default_fs_methods . . . 30
set_default_param_grids . . . 31
set_reticulate_python . . . 31
skip_if_no_modules . . . 32
split_data . . . 33
steps_to_tuples . . . 34

**Index** **35**

## `aggregate_feature_importances` *Aggregate Feature Importances*

### Description

This function aggregates the feature importances for each method across all splits.

### Usage

```
aggregate_feature_importances(selected_features)
```

### Arguments

`selected_features`
A list of selected features. Each element of the list represents a split and should be a named list where the names are the methods and the values are data frames containing the feature importances for that method in that split.

### Value

A list containing aggregated feature importances for each feature selection method. Each element in the list is a dataframe with mean and standard deviation of the feature importances for a particular method across all splits. The dataframe includes columns for feature names, mean importances, standard deviations, and ranks.

## `AnnotatedGeneLists-class` *AnnotatedGeneLists class*

### Description

A class to hold a list of GeneList objects, each representing a method.

### Slots

`inbuilt` A list of GeneList objects containing annotations for genes selected with inbuilt, model-specific feature importance.

`permutation` A list of GeneList objects containing annotations for genes selected with permutation importance.

## annotate_gene_lists *Convert and Annotate Gene Lists*

### Description

This function converts a list of gene lists to different formats and returns an object of the class AnnotatedGeneLists.

### Usage

```
annotate_gene_lists(
  pipeline_results,
  custom_lists = NULL,
  annotations_ahb,
  format = c("ENTREZ", "ENSEMBL", "SYMBOL")
)
```

### Arguments

`pipeline_results`
A PipelineResults object containing a named list of data frames.

`custom_lists` (optional) A named list of character vectors containing additional user-defined gene sets.

`annotations_ahb`
A data.frame object containing gene annotations with columns 'gene_id', 'gene_name', and 'entrezid'.

`format` The format of the gene list in 'pipeline_results' and 'custom_lists'. This should be one of "ENSEMBL", "ENTREZ", or "SYMBOL".

### Value

An object of the class AnnotatedGeneLists, which contains the following components: - @field inbuilt: A list of AnnotatedGeneList objects derived from the 'inbuilt_feature_importance' field of the 'pipeline_results' parameter. - @field permutation: A list of AnnotatedGeneList objects derived from the 'permutation_importance' field of the 'pipeline_results' parameter, if available. Each AnnotatedGeneList object in these lists includes the gene identifiers in different formats (SYMBOL, ENSEMBL, ENTREZID) and is structured to facilitate further analysis and visualization of gene lists. If an error occurs during the conversion or annotation process, a warning message is given and the corresponding list entry will be NULL.

### Examples

```
# Simple Usage with Mock Data
# Create a mock PipelineResults object with minimal data
mock_pipeline_results <- new("PipelineResults",
                             inbuilt_feature_importance = list(
                               "GeneSet1" = data.frame(feature = c("BRCA1", "TP53"))),
                             permutation_importance = list(
                               "GeneSet1" = data.frame(feature = c("BRCA1", "TP53"))))

# Mock annotations data frame
mock_annotations_ahb <- data.frame(gene_id = c("BRCA1", "TP53"),
                                   gene_name = c("BRCA1", "TP53"),
                                   entrezid = c(101, 102))

# Convert and annotate gene lists
annotated_lists <- annotate_gene_lists(mock_pipeline_results,
                                       custom_lists = NULL,
                                       mock_annotations_ahb,
                                       "SYMBOL")
print(annotated_lists)

# Using Custom Gene Lists
# Create custom gene lists
custom_gene_lists <- list("CustomList1" = c("BRCA1", "TP53"))

# Convert and annotate gene lists with custom gene lists included
annotated_lists_custom <- annotate_gene_lists(mock_pipeline_results,
                                              custom_gene_lists,
                                              mock_annotations_ahb,
                                              "SYMBOL")
print(annotated_lists_custom)
```

## calculate_mean_cv_scores

*Calculate Mean Cross-Validation Scores for Various Feature Selection Methods*

### Description

Calculate Mean Cross-Validation Scores for Various Feature Selection Methods

### Usage

```
calculate_mean_cv_scores(selected_pipelines, cv_best_score)
```

### Arguments

`selected_pipelines`
A list of pipelines for different feature selection methods.

`cv_best_score`
A list or vector of cross-validation scores.

### Value

A dataframe containing the mean and standard deviation of cross-validation scores for each method.

## calculate_overlap_coefficients

*Calculate Overlap and Similarity Coefficients between Feature Lists*

### Description

This function calculates the Overlap, Jaccard, and Soerensen-Dice coefficients to quantify the similarity between feature lists. In addition to feature importance and permutation importance, you can provide a custom list of feature names to be included in the overlap calculation.

### Usage

```
calculate_overlap_coefficients(pipeline_results, custom_lists = NULL)
```

### Arguments

`pipeline_results`
A PipelineResults object containing the fitted pipelines, cross-validation results, selected features, mean performance, and mean feature importances.

`custom_lists`
An optional named list of character vectors. Each character vector should contain feature names. The names of the list will be used as names in the resulting overlap coefficient matrices.

### Value

A list containing lists of matrices, where each list corresponds to a different type of feature list (inbuilt feature importance, permutation importance, and custom lists if provided). Within each of these lists, there are three matrices showing the Overlap, Jaccard, and Soerensen-Dice coefficients for the feature lists: - @field overlap: A matrix showing the Overlap coefficients. - @field jaccard: A matrix showing the Jaccard coefficients. - @field soerensen: A matrix showing the Soerensen-Dice coefficients. These matrices compare the feature lists against each other, providing a numerical measure of their similarity. Note: If permutation importance data is not present in the `pipeline_results`, the corresponding list entry will be absent.

### Examples

```
# Basic Usage with Mock Data
# Create a mock PipelineResults object with minimal data
mock_pipeline_results <- new("PipelineResults",
                             inbuilt_feature_importance = list(
                        "FeatureSet1" = data.frame(feature = c("feature1", "feature2")),
                       "FeatureSet2" = data.frame(feature = c("feature2", "feature3"))),
                             permutation_importance = list(
                        "FeatureSet1" = data.frame(feature = c("feature3", "feature4")),
                       "FeatureSet2" = data.frame(feature = c("feature1", "feature4"))))

# Calculate overlap coefficients without custom lists
overlap_results <- calculate_overlap_coefficients(mock_pipeline_results)
```

```
# Including Custom Lists
# Create custom feature lists
custom_feature_lists <- list("CustomList1" = c("feature5", "feature6"),
                             "CustomList2" = c("feature6", "feature7"))

# Calculate overlap coefficients with custom lists
overlap_results_custom <- calculate_overlap_coefficients(mock_pipeline_results,
                                                         custom_feature_lists)
```

## calculate_permutation_feature_importance
*Calculate Permutation Feature Importance*

### Description

This function calculates permutation feature importance for a Scikit-learn pipeline with a trained classifier as the final step.

### Usage

```
calculate_permutation_feature_importance(
  pipeline,
  X_train,
  y_train,
  n_repeats = 10L,
  random_state = 0L,
  njobs = njobs,
  pipeline_name,
  iter
)
```

### Arguments

`pipeline` A Scikit-learn pipeline object with a trained classifier as the final step.

`X_train` A DataFrame containing the training data.

`y_train` A DataFrame containing the training labels.

`n_repeats` An integer specifying the number of times to permute each feature.

`random_state` An integer specifying the seed for the random number generator.

`njobs` An integer specifying number of cores to use. Set up by the master GeneSelectR function.

`pipeline_name` Strings (names of the selected_pipelines list) representing pipeline names that were constructed for the feature selection

`iter` An integer that is indicating current iteration of the train-test split

## Value

A dataframe containing the feature names and their permutation importance scores, ranked by importance. Each row represents a feature, with columns for feature names, importances, and ranks.

---

# check_python_modules_available

*Check Python Module Availability for Examples*

---

## Description

Checks if the specified Python modules are available and returns TRUE if all are available, otherwise returns FALSE.

## Usage

```
check_python_modules_available(module_names)
```

## Arguments

| | |
|---|---|
| `module_names` | Character vector of Python module names to check. |

## Value

Logical TRUE if all modules are available, FALSE otherwise.

---

# compute_GO_child_term_metrics

*Retrieve and Plot the Offspring Nodes of GO Terms*

---

## Description

This function retrieves the children nodes for a set of Gene Ontology (GO) terms from a list of GO terms and can plot the offspring nodes' numbers and fractions for each term.

## Usage

```
compute_GO_child_term_metrics(GO_data, GO_terms, ontology = "BP", plot = FALSE)
```

## Arguments

GO_data A list of GO data where each element corresponds to a different feature list. Each element should have a @result data frame with a column 'ID' containing GO terms.

GO_terms A character vector containing GO term IDs for which offspring nodes are to be fetched.

ontology A character string specifying the type of ontology to be considered. Can be one of 'BP' (Biological Process), 'MF' (Molecular Function), or 'CC' (Cellular Component). Default is 'BP'.

plot A logical. If TRUE, the function plots the number and fraction of offspring nodes for each term in GO_terms across all feature lists in GO_data. Default is FALSE.

## Value

A data frame with columns:

- **feature_list** - The names of the feature lists from GO_data.
- **all_terms_number** - The total number of GO terms in each feature list.
- **offspring_nodes_number** - The number of offspring nodes for the given GO term(s) in each feature list.
- **offspring_terms** - The actual offspring terms for the given GO term(s) in each feature list, concatenated by ';'.
- **fraction** - The fraction (percentage) of offspring nodes out of all terms in each feature list.
- **GO_term** - The GO term being considered.

## Examples

```
# Mock GO terms data frame
all_selection.GO_inbuilt <- data.frame(
  GO_ID = c("GO:0002376", "GO:0008150", "GO:0006955", "GO:0009628"),
  Description = c("immune system process",
                  "biological_process",
                  "immune response",
                  "response to virus"),
  Parent_GO_ID = c(NA, NA, "GO:0002376", "GO:0006955"), # Simplified parent-child
  stringsAsFactors = FALSE
)

# Mock vector of GO terms to compute metrics for
GO_terms_vec <- c("GO:0002376", "GO:0008150")

# Assuming compute_GO_child_term_metrics is defined and available
# df_res <- compute_GO_child_term_metrics(GO_data = all_selection.GO_inbuilt,
#                                         GO_terms = GO_terms_vec,
#                                         plot = TRUE)
```

## configure_environment *Configure Python Environment for GeneSelectR*

### Description

This function checks if Conda is installed, creates a new Conda environment (if it does not already exist), installs necessary Python packages into the environment, and sets it as the active environment for reticulate.

### Usage

```
configure_environment(env_name = "GeneSelectR_env")
```

### Arguments

| | |
|---|---|
| env_name | The name of the Conda environment to be created. Defaults to "GeneSelectR_env". |

### Value

A message indicating the status of the environment configuration. If successful, it informs the user that the environment was created and necessary packages were installed. If Conda is not installed or an error occurs, the function stops with an error message. The function also advises the user to restart their R session for the changes to take effect.

### Examples

```
# Configure the default environment
configure_environment()

# Configure a custom environment
configure_environment("my_env_name")
```

## create_conda_env *Create a specific Conda environment*

### Description

This function creates a Conda environment if it doesn't already exist.

### Usage

```
create_conda_env(conda_env = "GeneSelectR_env")
```

### Arguments

conda_env The name of the Conda environment to create.

### Value

Creates conda environment for GeneSelectR package called 'GeneSelectR_env'. If environment already exists, returns a message indicating that the environment is there.

---

## `create_pipelines` *Create Pipelines*

---

### Description

This function creates a list of Scikit-learn pipelines using the specified feature selection methods, preprocessing steps, and classifier.

### Usage

```
create_pipelines(
  feature_selection_methods,
  preprocessing_steps,
  selected_methods,
  classifier,
  fs_param_grids
)
```

### Arguments

feature_selection_methods
A list of feature selection methods to use for the pipelines.

preprocessing_steps
A list of preprocessing steps to use for the pipelines.

selected_methods
A vector of names of feature selection methods to use from the default set.

classifier A Scikit-learn classifier to use as the final step in the pipelines.

fs_param_grids param grid

### Value

A list of Scikit-learn pipeline objects. Each pipeline is constructed based on the provided feature selection methods, preprocessing steps, and classifier. The list is named by feature selection methods.

## create_test_metrics_df *Create a Dataframe of Test Metrics*

### Description

Create a Dataframe of Test Metrics

### Usage

```
create_test_metrics_df(test_metrics)
```

### Arguments

| | |
|---|---|
| test_metrics | A list or dataframe of test metrics. |

### Value

A dataframe with the processed test metrics.

## define_sklearn_modules *Define Python modules and scikit-learn submodules*

### Description

Define Python modules and scikit-learn submodules

### Usage

```
define_sklearn_modules(python_modules)
```

### Arguments

| | |
|---|---|
| python_modules | A list containing imported Python modules. |

### Value

A list containing the initialized Python modules and scikit-learn submodules, each as a separate list element. The list includes:

- @field preprocessing: Module for data preprocessing.
- @field model_selection: Module for model selection and evaluation.
- @field feature_selection: Module for feature selection methods.
- @field ensemble: Module for ensemble methods.

- @field pipeline: scikit-learn pipeline object.
- @field forest: Random Forest classifier for feature selection.
- @field randomized_grid: Randomized grid search for hyperparameter tuning.
- @field grid: Grid search for hyperparameter tuning.
- @field bayesianCV: Bayesian optimization using cross-validation.
- @field lasso: Lasso method for feature selection.
- @field univariate: Univariate feature selection method.
- @field select_model: Model-based feature selection method.
- @field GradBoost: Gradient Boosting classifier.

## Examples

```
required_modules <- c("sklearn", "boruta")
modules_available <- sapply(required_modules, reticulate::py_module_available)

if (all(modules_available)) {
  # All required Python modules are available
  # Define scikit-learn modules and submodules
  sklearn_modules <- define_sklearn_modules()

  # Access different modules and submodules
  preprocessing_module <- sklearn_modules$preprocessing
  model_selection_module <- sklearn_modules$model_selection
  feature_selection_module <- sklearn_modules$feature_selection
  ensemble_module <- sklearn_modules$ensemble
  # Additional code to explore each module as needed in your analysis
} else {
  unavailable_modules <- names(modules_available[!modules_available])
  message(paste(
  "Required Python modules not available:",
  paste(unavailable_modules, collapse=', '), ". Skipping example."))
}
```

---

# enable_multiprocess *Enable Multiprocessing in Python Environment*

---

## Description

This function sets up the necessary executable for Python's multiprocessing functionality. Only used on Windows

## Usage

```
enable_multiprocess(python_modules)
```

### Arguments

| | |
|---|---|
| `python_modules` | a list containing imported Python modules |

### Value

Doesn't return anything, enables multiprocessing on Windows

## `evaluate_test_metrics` *Evaluate Test Metrics for a Grid Search Model*

### Description

This function takes a grid search object, test data, and test labels to evaluate the performance of the best model found during grid search.

### Usage

```
evaluate_test_metrics(grid_search, X_test, y_test, modules)
```

### Arguments

| | |
|---|---|
| `grid_search` | A grid search object containing the best estimator. |
| `X_test` | A data frame or matrix of test features. |
| `y_test` | A vector of test labels. |
| `modules` | A list of Python modules used in the function. |

### Value

A list containing key performance metrics of the best model: - @field precision: The weighted precision score. - @field recall: The weighted recall score. - @field f1: The weighted F1 score. - @field accuracy: The overall accuracy score. These metrics are crucial for evaluating the effectiveness of the model on test data.

## `GeneList-class` *GeneList class*

### Description

A class to hold annotated gene list for a single method.

### Slots

`SYMBOL` A character vector of gene names.

`ENSEMBL` A character vector of Ensembl IDs.

`ENTREZID` A character vector of Entrez IDs.

`GeneSelectR` *Gene Selection and Evaluation with GeneSelectR*

### Description

This function performs gene selection using different methods on a given training set and evaluates their performance using cross-validation. Optionally, it also calculates permutation feature importances.

### Usage

```
GeneSelectR(
  X,
  y,
  pipelines = NULL,
  custom_fs_methods = NULL,
  selected_methods = NULL,
  custom_fs_grids = NULL,
  classifier = NULL,
  classifier_grid = NULL,
  preprocessing_steps = NULL,
  testsize = 0.2,
  validsize = 0.2,
  scoring = "accuracy",
  njobs = -1,
  n_splits = 5,
  search_type = "random",
  n_iter = 10,
  max_features = 50,
  calculate_permutation_importance = FALSE,
  perform_test_split = FALSE,
  random_state = NULL
)
```

### Arguments

`X` A matrix or data frame with features as columns and observations as rows.

`y` A vector of labels corresponding to the rows of X_train.

`pipelines` An optional list of pre-defined pipelines to use for fitting and evaluation. If this argument is provided, the feature selection methods and preprocessing steps will be ignored.

`custom_fs_methods`
An optional list of feature selection methods to use for fitting and evaluation. If this argument is not provided, a default set of feature selection methods will be used.

`selected_methods`
An optional vector of names of feature selection methods to use from the default set. If this argument is provided, only the specified methods will be used.

`custom_fs_grids`
An optional list of hyperparameter grids for the feature selection methods. Each element of the list should be a named list of parameters for a specific feature selection method. The names of the elements should match the names of the feature selection methods. If this argument is provided, the function will perform hyperparameter tuning for the specified feature selection methods in addition to the final estimator.

`classifier`
An optional sklearn classifier. If left NULL then sklearn RandomForestClassifier is used.

`classifier_grid`
An optional named list of classifier parameters. If none are provided then default grid is used (check vignette for exact params).

`preprocessing_steps`
An optional named list of sklearn preprocessing procedures. If none provided defaults are used (check vignette for exact params).

`testsize`
The size of the test set used in the evaluation.

`validsize`
The size of the validation set used in the evaluation.

`scoring`
A string representing what scoring metric to use for hyperparameter adjustment. Default value is 'accuracy'

`njobs`
Number of jobs to run in parallel.

`n_splits`
Number of train/test splits.

`search_type`
A string indicating the type of search to use. 'grid' for GridSearchCV and 'random' for RandomizedSearchCV. Default is 'random'.

`n_iter`
An integer indicating the number of parameter settings that are sampled in RandomizedSearchCV. Only applies when search_type is 'random'.

`max_features`
Maximum number of features to be selected by default feature selection methods. Max features cannot exceed the total number of features in a dataset.

`calculate_permutation_importance`
A boolean indicating whether to calculate permutation feature importance. Default is FALSE.

`perform_test_split`
Whether to perform train and test split, to have an evaluation on unseen test set. The default value is set to FALSE

`random_state`
An integer value setting the random seed for feature selection algorithms and cross validation procedure. By default set to NULL to use different random seed every time an algorithm is used. For reproducibility could be fixed, otherwise for an unbiased estimation should be left as NULL.

## Value

Returns an object of class `PipelineResults` with the following elements:

- @field best_pipeline: A list of the best-fitted pipelines for each feature selection method and data split.
- @field cv_results: A list containing cross-validation results for each pipeline, including scores and other metrics.
- @field inbuilt_feature_importance: A list of the inbuilt feature importance scores for each pipeline, aggregated across all data splits.
- @field test_metrics: A data frame summarizing test metrics (precision, recall, F1 score, accuracy) for each pipeline, if a test split was performed.
- @field cv_mean_score: A data frame summarizing the mean cross-validation scores for each pipeline across all data splits.
- @field permutation_importance: A list of permutation importance scores for each pipeline, if permutation importance calculation was enabled. This comprehensive return structure allows for in-depth analysis of the feature selection methods and model performance.

## Examples

```
if (GeneSelectR:::check_python_modules_available(c("numpy", "pandas", "sklearn", 'boruta'))) {
  # Create a mock dataset with 29 feature columns and 1 binary label column
  set.seed(123) # for reproducibility
  n_rows <- 10
  n_features <- 100

  # Randomly generate feature data
  X <- as.data.frame(matrix(rnorm(n_rows * n_features), nrow = n_rows, ncol = n_features))
  # Ensure each feature has a variance greater than 0.85
  for(i in 1:ncol(X)) {
    while(var(X[[i]]) <= 0.85) {
      X[[i]] <- X[[i]] * 1.1
    }
  }
  colnames(X) <- paste0("Feature", 1:n_features)

  # Create a mock binary label column
  y <- factor(sample(c("Class1", "Class2"), n_rows, replace = TRUE))

  # Set up the environment
  GeneSelectR::configure_environment()
  GeneSelectR::set_reticulate_python()

  # Run GeneSelectR
  results <- GeneSelectR(X, y)

  # Perform gene selection and evaluation using user-defined methods
  fs_methods <- list("Lasso" = select_model(lasso(penalty = 'l1',
                                                  C = 0.1,
                                                  solver = 'saga'),
                                            threshold = 'median'))
  custom_fs_grids <- list("Lasso" = list('C' = c(0.1, 1, 10)))
  results <- GeneSelectR(X,
                         y,
                            max_features = 15,
                            custom_fs_methods = fs_methods,
                            custom_fs_grids = custom_fs_grids)
} else {
  message("Skipping example as not all required Python modules are available.")
}
```

## get_feature_importances

*Get Feature Importances*

### Description

This function extracts feature importances from a Scikit-learn pipeline that has a Gradient Boosting Classifier as the final step.

### Usage

```
get_feature_importances(pipeline, X_train, pipeline_name, iter)
```

### Arguments

| | |
|---|---|
| pipeline | A Scikit-learn pipeline object with a Gradient Boosting Classifier as the final step. |
| X_train | A DataFrame containing the training data. |
| pipeline_name | Strings (names of the selected_pipelines list) representing pipeline names that were constructed for the feature selection |
| iter | An integer that is indicating current iteration of the train-test split |

### Value

A dataframe containing the selected feature names and their importances, ranked by importance, or NULL if the classifier does not have the appropriate attributes or the feature selector does not have the ’get_support’ or ’support_’ method. Each row represents a feature, with columns for feature names, importances, and ranks.

## GO_enrichment_analysis

*Perform gene set enrichment analysis using clusterProfiler*

### Description

This function performs gene set enrichment analysis on a list of gene sets extracted from an AnnotatedGeneLists object.

### Usage

```
GO_enrichment_analysis(
  annotated_gene_lists,
  list_type = "inbuilt",
  background = NULL,
  organism = "org.Hs.eg.db",
  keyType = "ENTREZID",
  ont = "BP",
  pvalueCutoff = 0.05,
  qvalueCutoff = 0.2,
  pAdjMethod = "fdr",
  ...
)
```

### Arguments

| | |
|---|---|
| `annotated_gene_lists` | An AnnotatedGeneLists object containing a list of GeneList objects. |
| `list_type` | A type of AnnotatedGeneList from annotate_gene_lists function. Either 'inbuilt' or 'permutation'. (default: 'inbuilt') |
| `background` | A character vector representing the background gene set. |
| `organism` | A character string corresponding to the organism of interest. Default: "org.Hs.eg.db" (for human). |
| `keyType` | A character string indicating the type of gene identifiers. Default: "ENTREZID". |
| `ont` | A character string representing GO term ontology. Default: "BP" (for Biological Process). |
| `pvalueCutoff` | A numeric value specifying the significance cutoff. Default: 0.05. |
| `qvalueCutoff` | A numeric value specifying the q-value cutoff. Default: 0.2. |
| `pAdjMethod` | A p-value adjustment method. Should be the one from "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none". Default is 'fdr |
| `...` | Other parameters to be passed to clusterProfiler::enrichGO function. |

## Value

A list containing the enrichment results for each gene set. Each element in the list is named after a gene set and contains an object produced by the `enrichGO` function from `clusterProfiler`. This object includes various details about the enrichment analysis, such as enriched GO terms, their associated p-values, q-values, and other relevant statistics. The list excludes results for the "background" gene set, focusing only on the gene sets of interest.

## References

To use clusterProfiler in published research, please cite: Yu G, Wang LG, Han Y, He QY. clusterProfiler: an R package for comparing biological themes among gene clusters. OMICS: A Journal of Integrative Biology. 2012;16(5):284-287. doi:10.1089/omi.2011.0118.

## Examples

```
# Creating a mock AnnotatedGeneLists object
gene_symbols <- c("GENE1", "GENE2", "GENE3")
ensembl_ids <- c("ENSG000001", "ENSG000002", "ENSG000003")
entrez_ids <- c("1001", "1002", "1003")

create_mock_gene_list <- function() {
  new("GeneList",
       SYMBOL = gene_symbols,
       ENSEMBL = ensembl_ids,
       ENTREZID = entrez_ids)
}

mock_gene_list1 <- create_mock_gene_list()
mock_gene_list2 <- create_mock_gene_list()

annotated_gene_lists <- new("AnnotatedGeneLists",
                            inbuilt = list(Lasso = mock_gene_list1,
                                           Univariate = mock_gene_list2),
                            permutation = list(Lasso = mock_gene_list1,
                                               Univariate = mock_gene_list2))

# Define a background gene set
background <- c("GENE1", "GENE2", "GENE3")

# Perform GO enrichment analysis
results <- GO_enrichment_analysis(annotated_gene_lists,
                                  background = background,
                                  organism = "org.Hs.eg.db",
                                  keyType = "SYMBOL")
print(results)
```

`import_python_packages`

*Import Python Libraries*

### Description

This function imports the necessary Python libraries for the package.

### Usage

```
import_python_packages()
```

### Value

A list containing references to imported Python libraries used in the package:

- @field sklearn: Scikit-learn machine learning library.
- @field pandas: Data manipulation and analysis library.
- @field numpy: Library for numerical computing.
- @field lightgbm: Gradient boosting framework.
- @field xgboost: Optimized distributed gradient boosting library.
- @field boruta: Feature selection algorithm.
- @field sys: Python system-specific parameters and functions.
- @field multiprocessing: Support for concurrent execution using processes.

`install_python_packages`

*Install necessary Python packages in a specific Conda environment*

### Description

This function installs the necessary Python packages in a specific Conda environment.

### Usage

```
install_python_packages(conda_env = "GeneSelectR_env")
```

### Arguments

| | |
|---|---|
| `conda_env` | The name of the Conda environment to use. |

### Value

Installs necessary version of Python packages into the GeneSelectR_env.

perform_grid_search *Perform Grid Search or Random Search for Hyperparameter Tuning*

## Description

Perform Grid Search or Random Search for Hyperparameter Tuning

## Usage

```
perform_grid_search(
  X_train,
  y_train,
  pipeline,
  scoring,
  params,
  search_type,
  n_iter,
  njobs,
  modules,
  random_state
)
```

## Arguments

| | |
|---|---|
| X_train | Training data for predictors. |
| y_train | Training data for outcomes. |
| pipeline | A pipeline specifying the steps for feature selection and model training. |
| scoring | A string representing what scoring metric to use for hyperparameter adjustment. Default value is 'accuracy' |
| params | A list of parameters or parameter distributions to search over. |
| search_type | A character string specifying the type of search ('grid' or 'random'). |
| n_iter | The number of parameter settings that are sampled in a random search. |
| njobs | The number of CPU cores to use. |
| modules | A list containing the definitions for the Python modules and submodules. |
| random_state | An integer value setting the random seed for feature selection algorithms and randomized search CV procedure. By default set to NULL to use different random seed every time an algorithm is used. For reproducibility could be fixed, otherwise for an unbiased estimation should be left as NULL. |

## Value

Returns a scikit-learn GridSearchCV, RandomizedSearchCV, or BayesSearchCV object, depending on the `search_type` specified. This object includes several attributes useful for analyzing the hyperparameter tuning process: - @field best_estimator_: The best estimator chosen by the search. -

@field best_score_: The score of the best_estimator on the left-out data. - @field best_params_: The parameter setting that gave the best results on the hold-out data. - @field cv_results_: A dict with keys as column headers and values as columns, that can be imported into a pandas DataFrame. - @field scorer_: Scoring method used on the held-out data. - @field n_splits_: The number of cross-validation splits (folds/iterations). These attributes provide insights into the model's performance and the effectiveness of the selected hyperparameters.

## Examples

```
required_modules <- c("sklearn", "boruta")
modules_available <- sapply(required_modules, reticulate::py_module_available)

if (all(modules_available)) {
# Assuming X_train, y_train, pipeline, and params are predefined
# Define sklearn modules (assuming 'define_sklearn_modules' is defined)
sklearn_modules <- define_sklearn_modules()

# Perform a grid search
optimal_model <- perform_grid_search(X_train, y_train, pipeline, "accuracy",
                                     params, "grid", NULL, 1, sklearn_modules, NULL)


# Perform a random search
optimal_model_random <- perform_grid_search(X_train, y_train, pipeline, "accuracy",
                                            params, "random", 10, 1, sklearn_modules, 42)

} else {
unavailable_modules <- names(modules_available[!modules_available])
message(paste("Required Python modules not available:",
  paste(unavailable_modules, collapse=', '), ". Skipping example."))
}
```

---

# `PipelineResults-class` *PipelineResults class*

---

## Description

A class to hold the results of GeneSelectR function.

## Slots

`best_pipeline` A named list containing parameters of the best performer pipeline

`cv_results` A list of the cross-validation results for each pipeline.

`inbuilt_feature_importance` A list of the inbuilt mean feature importances for each method across all splits.

`permutation_importance` A list of the permutation importances for each method.

`cv_mean_score` A data.frame of mean scores from cross-validation.

test_metrics A data.frame containing metrics (F1, accuracy, precision, and recall) calculated on the unseen test set. Contains mean values across splits as well as standard deviation.

## pipeline_to_list *Convert Scikit-learn Pipeline to Named List*

### Description

This function takes a Scikit-learn Pipeline object and converts it to a named list in R. Each step in the pipeline becomes an element in the list with the name of the step as the name of the list element.

### Usage

```
pipeline_to_list(pipeline)
```

### Arguments

| | |
|---|---|
| pipeline | A Scikit-learn Pipeline object. |

### Value

A named list where each element represents a step in the Scikit-learn Pipeline. The names of the list elements correspond to the names of the steps in the pipeline. Each element of the list is an R representation of the respective step in the pipeline.

### Examples

```
# Assuming a Scikit-learn pipeline object 'sklearn_pipeline' is defined in Python
# and available in R via reticulate
sklearn_pipeline <- reticulate::import("sklearn.pipeline")$Pipeline(steps = list(
  list("scaler", reticulate::import("sklearn.preprocessing")$StandardScaler()),
  list("classifier", reticulate::import("sklearn.ensemble")$RandomForestClassifier())
))

# Convert the Scikit-learn pipeline to a named list in R
pipeline_list <- pipeline_to_list(sklearn_pipeline)
print(pipeline_list)
```

plot_feature_importance *Plot Feature Importance*

### Description

This function plots the feature importance scores from `inbuilt_feature_importance` and `permutation_importance` in the `PipelineResults` object.

### Usage

```
plot_feature_importance(pipelineresults, top_n_features = 10)
```

### Arguments

`pipelineresults`
An object of class `PipelineResults`.

`top_n_features` An integer specifying the top N features to plot based on their mean importance.

### Value

A list of grid plot objects (ggplot objects) for each feature selection method in the `PipelineResults` object. Each plot visualizes the top N features based on their mean importance scores, including both inbuilt and permutation importances (if available). The plots are arranged in a grid layout for easy comparison.

### Examples

```
# Assuming `pipelineresults` is a PipelineResults object

pipelineresults <- new("PipelineResults",
                       inbuilt_feature_importance = list("Method1" = data.frame(
                                                  feature = LETTERS[1:10],
                                                  mean_importance = runif(10)),
                                                  "Method2" = data.frame(
                                                  feature = LETTERS[1:10],
                                                  mean_importance = runif(10))),
                       permutation_importance = list("Method1" = data.frame(
                                                         feature = LETTERS[1:10],
                                                   mean_importance = runif(10))))

# Plot the feature importance
importance_plots <- plot_feature_importance(pipelineresults, top_n_features = 5)
print(importance_plots)
```

## plot_metrics *Plot Performance Metrics*

### Description

This function creates separate plots for f1_mean, recall_mean, precision_mean, and accuracy_mean from a given dataframe, then arranges these plots using cowplot::plot_grid. Requires ggplot2 and cowplot packages.

### Usage

```
plot_metrics(pipeline_results)
```

### Arguments

`pipeline_results`
: An object of class "PipelineResults" containing the performance metrics. This object is expected to contain a dataframe with the columns 'method', 'f1_mean', 'f1_sd', 'recall_mean', 'recall_sd', 'precision_mean', 'precision_sd', 'accuracy_mean', 'accuracy_sd'.

### Value

A combined ggplot object displaying the performance metrics. - If test metrics are provided, it includes separate plots for F1 mean, recall mean, precision mean, and accuracy mean, along with cross-validation mean score, arranged in a grid layout. - If no test metrics are available, it returns only the cross-validation mean score plot.

### Examples

```
# Assuming `pipeline_results` is a PipelineResults object with test metrics and CV mean score
pipeline_results <- new("PipelineResults",
                        test_metrics = data.frame(
                        method = c("Method1", "Method2"),
                        f1_mean = c(0.8, 0.85), f1_sd = c(0.05, 0.04),
                        recall_mean = c(0.75, 0.78), recall_sd = c(0.06, 0.05),
                        precision_mean = c(0.85, 0.88), precision_sd = c(0.05, 0.04),
                        accuracy_mean = c(0.9, 0.92), accuracy_sd = c(0.03, 0.02)),
                        cv_mean_score = data.frame(
                        method = c("Method1", "Method2"),
                        mean_score = c(0.88, 0.9), sd_score = c(0.02, 0.02)))

# Plot the performance metrics
metric_plots <- plot_metrics(pipeline_results)
print(metric_plots)
```

## plot_overlap_heatmaps *Generate Heatmaps to Visualize Overlap and Similarity Coefficients between Feature Lists*

### Description

This function takes a list of matrices of overlap and similarity coefficients and generates heatmaps to visualize them.

### Usage

```
plot_overlap_heatmaps(coefficients, save_plot = FALSE, filename = NULL)
```

### Arguments

| | |
|---|---|
| `coefficients` | A list of matrices showing the Overlap, Jaccard, and Soerensen-Dice coefficients for the feature lists. |
| `save_plot` | A logical value indicating whether to save the heatmap plots to a file or not. Default is FALSE. |
| `filename` | A character string specifying the filename for the saved heatmap plots (if save_plot = TRUE). |

### Value

A grid of ggplot2 heatmap objects visualizing the Overlap, Jaccard, and Soerensen-Dice coefficients. The grid layout includes heatmaps for both 'inbuilt' and 'permutation' feature importance coefficients (if available). If `save_plot` is TRUE, the heatmaps are also saved to the specified file.

### Examples

```
# Assuming `coefficients` is a list containing matrices for Overlap, Jaccard,
# and Soerensen-Dice coefficients
# For demonstration, let's create a mock coefficients list
mock_matrix <- matrix(runif(25), nrow = 5)
coefficients <- list(inbuilt_feature_importance_coefficient = list(overlap = mock_matrix,
                     jaccard = mock_matrix, soerensen = mock_matrix),
                     permutation_importance_coefficients = list(overlap = mock_matrix,
                     jaccard = mock_matrix, soerensen = mock_matrix))

# Plot the overlap heatmaps
heatmap_plots <- plot_overlap_heatmaps(coefficients)
print(heatmap_plots)
```

## plot_upset *Plot Feature Overlaps Using UpSet Plots*

### Description

This function produces separate UpSet plots for inbuilt feature importances and permutation importances, allowing you to visualize the overlap of feature lists. Optionally, you can include custom lists.

### Usage

```
plot_upset(pipeline_results, custom_lists = NULL)
```

### Arguments

`pipeline_results`
A PipelineResults object containing the fitted pipelines, cross-validation results, selected features, mean performance, and mean feature importances.

`custom_lists` An optional named list of character vectors. Each character vector should contain feature names. The names of the list will be used as names in the UpSet plots.

### Value

A named list containing two UpSet plots:

- @field inbuilt_importance: An UpSet plot visualizing overlaps of inbuilt feature importances.
- @field permutation_importance: An UpSet plot (if permutation importance is available) visualizing overlaps of permutation importances. Each plot provides an interactive way to explore the intersections and unique elements of the feature lists.

### Examples

```
# Mock data for PipelineResults
pipeline_results <- new("PipelineResults",
                        inbuilt_feature_importance = list(
                          Method1 = data.frame(feature = c("gene1", "gene2", "gene3")),
                          Method2 = data.frame(feature = c("gene2", "gene4"))),
                        permutation_importance = list(
                          Method1 = data.frame(feature = c("gene1", "gene5")),
                          Method2 = data.frame(feature = c("gene3", "gene6"))))

# Mock custom lists
custom_lists <- list("custom1" = c("gene1", "gene2"), "custom2" = c("gene3", "gene4"))

# Generate UpSet plots
result <- plot_upset(pipeline_results, custom_lists)
print(result$inbuilt_importance)
print(result$permutation_importance)
```

## run_simplify_enrichment

*Run simplifyGOFromMultipleLists with specified measure and method*

### Description

This function is simply a wrapper for the simplifyGOFromMultipleLists function in the simplifyEnrichment package, created for the ease of data input. All credit for the underlying functionality goes to the authors of the simplifyEnrichment package.

### Usage

```
run_simplify_enrichment(
  fs_GO_results,
  padj_column = "p.adjust",
  padj_cutoff,
  ont,
  measure,
  method,
  ...
)
```

### Arguments

| | |
|---|---|
| `fs_GO_results` | A list of dataframes containing GO enrichment results for feature selection methods. The GO object for the simplifyGOFromMultipleLists function. |
| `padj_column` | Character. The column name for the p-value adjustment. |
| `padj_cutoff` | Numeric. The cutoff for the p-value adjustment. |
| `ont` | Character. The ontology for the simplifyGOFromMultipleLists function. |
| `measure` | Character. The semantic similarity measure for the simplifyGOFromMultipleLists function. |
| `method` | Character. The clustering method for the simplifyGOFromMultipleLists function. |
| `...` | Other parameters that can be passed to simplifyGOFromMultipleLists |

### Value

The result of the `simplifyGOFromMultipleLists` function, typically comprising a heatmap or other visualization that displays the simplified GO enrichment results. The specific output format depends on the chosen semantic similarity measure and clustering method.

### References

For more information on the simplifyEnrichment package, see the original publication: Gu Z, Hübschmann D. simplifyEnrichment: A Bioconductor Package for Clustering and Visualizing Functional Enrichment Results. Genomics Proteomics Bioinformatics. 2023 Feb;21(1):190-202. doi: 10.1016/j.gpb.2022.04.008. Epub 2022 Jun 6. PMID: 35680096; PMCID: PMC10373083.

## Examples

```
# Mock GO enrichment results for two feature selection methods
fs_GO_results <- list(
  method1 = list(result = data.frame(GO_ID = c("GO:0008150", "GO:0009987"),
                          Description = c("Biological Process 1", "Biological Process 2"),
                                    'p.adjust' = c(0.01, 0.02))),
  method2 = list(result = data.frame(GO_ID = c("GO:0008150", "GO:0008152"),
                          Description = c("Biological Process 1", "Biological Process 3"),
                                    'p.adjust' = c(0.03, 0.04)))
)

# Run the wrapper function with mock data
enrichment_result <- run_simplify_enrichment(fs_GO_results,
                                             padj_column = 'p.adjust',
                                             padj_cutoff = 0.05,
                                             ont = "BP",
                                             measure = "Wang",
                                             method = "kmeans")
print(enrichment_result)
```

---

# set_default_fs_methods

*Set Default Feature Selection Methods*

---

## Description

Set Default Feature Selection Methods

## Usage

```
set_default_fs_methods(modules, max_features, random_state)
```

## Arguments

`modules` A list containing the definitions for the Python modules and submodules.

`max_features` The maximum number of features to consider.

`random_state` An integer value setting the random seed for feature selection algorithms and cross validation procedure. By default set to NULL to use different random seed every time an algorithm is used. For reproducibility could be fixed, otherwise for an unbiased estimation should be left as NULL.

## Value

A list containing preprocessing steps and default feature selection methods.

## `set_default_param_grids` *Set Default Parameter Grids for Feature Selection*

### Description

Set Default Parameter Grids for Feature Selection

### Usage

```
set_default_param_grids(max_features)
```

### Arguments

| | |
|---|---|
| `max_features` | An integer indicating max_features to select in Univariate select |

### Value

A list containing the default parameter grids for feature selection methods.

## `set_reticulate_python` *Set RETICULATE_PYTHON for the Current Session*

### Description

This function sets the RETICULATE_PYTHON environment variable to the path of the Python interpreter in the specified Conda environment, but only for the current R session. The change will not persist after the R session is closed.

### Usage

```
set_reticulate_python(env_name = "GeneSelectR_env")
```

### Arguments

| | |
|---|---|
| `env_name` | The name of the Conda environment. Default is 'GeneSelectR_env'. |

### Details

This function checks if the specified Conda environment exists. If it does, the function sets the RETICULATE_PYTHON environment variable to the path of the Python interpreter in that environment. If the environment does not exist, the function stops with an error message.

Users need to run this function in every new R session where they want to use your package. Also, they should run this function before loading your package with library(), because the RETICULATE_PYTHON environment variable needs to be set before reticulate is loaded.

### Value

This function does not return a value. Instead, it sets the RETICULATE_PYTHON environment variable for the current R session and prints a message indicating the new value of RETICULATE_PYTHON. If the specified environment does not exist, the function stops with an error message.

### Examples

```
set_reticulate_python('GeneSelectR_env')
```

## skip_if_no_modules *Check if Python Modules are Available*

### Description

This helper function checks if a list of Python modules are available. If any are not, it skips the tests.

### Usage

```
skip_if_no_modules(module_names)
```

### Arguments

| | |
|---|---|
| module_names | A vector of names of the Python modules to check. |

### Value

Nothing is returned explicitly, but if a specified module is not available, the function invokes `testthat::skip` to skip the tests that require that module.

### Examples

```
# Example usage within a test file:
module_names <- c("numpy", "pandas", "sklearn")
skip_if_no_modules(module_names)
```

## split_data *Split Data into Training and Test Sets*

### Description

Split Data into Training and Test Sets

### Usage

```
split_data(X, y, test_size, modules)
```

### Arguments

| | |
|---|---|
| `X` | A dataframe or matrix of predictors. |
| `y` | A vector of outcomes. |
| `test_size` | Proportion of the data to be used as the test set. |
| `modules` | A list containing the definitions for the Python modules and submodules. |

### Value

A list containing the split datasets:

- @field X_train: Training set for predictors, converted to Python format.
- @field X_test: Test set for predictors, converted to Python format.
- @field y_train: Training set for outcomes, converted to Python format.
- @field y_test: Test set for outcomes, converted to Python format. The function ensures that the data is appropriately partitioned and formatted for use in Python-based analysis.

### Examples

```
# Assuming 'data' is your dataset with predictors and 'outcome' is the target variable
# Define sklearn modules (assuming 'define_sklearn_modules' is defined)
sklearn_modules <- define_sklearn_modules()

# Split the data into training and test sets
split_results <- split_data(data, outcome, test_size = 0.2, modules = sklearn_modules)
```

## `steps_to_tuples` *Convert Steps to Tuples*

### Description

This function converts a list of steps to tuples for use in a Scikit-learn pipeline.

### Usage

```
steps_to_tuples(steps)
```

### Arguments

`steps` A list of steps to convert to tuples.

### Value

A list of tuples, where each tuple represents a step in a Scikit-learn pipeline. The tuple contains the name of the step and the corresponding step object.

# Index

∗ **hidden**
    create_conda_env, 10

aggregate_feature_importances, 3
annotate_gene_lists, 4
AnnotatedGeneLists-class, 3

calculate_mean_cv_scores, 5
calculate_overlap_coefficients, 6
calculate_permutation_feature_importance, 7
check_python_modules_available, 8
compute_GO_child_term_metrics, 8
configure_environment, 10
create_conda_env, 10
create_pipelines, 11
create_test_metrics_df, 12

define_sklearn_modules, 12

enable_multiprocess, 13
evaluate_test_metrics, 14

GeneList-class, 14
GeneSelectR, 15
get_feature_importances, 18
GO_enrichment_analysis, 19

import_python_packages, 21
install_python_packages, 21

perform_grid_search, 22
pipeline_to_list, 24
PipelineResults-class, 23
plot_feature_importance, 25
plot_metrics, 26
plot_overlap_heatmaps, 27
plot_upset, 28

run_simplify_enrichment, 29

set_default_fs_methods, 30
set_default_param_grids, 31
set_reticulate_python, 31
skip_if_no_modules, 32
split_data, 33
steps_to_tuples, 34