Package ‘mikropml’

August 21, 2023

Title User-Friendly R Package for Supervised Machine Learning Pipelines

Version 1.6.1

Date 2023-08-21

Description An interface to build machine learning models for classification and regression problems. 'mikropml' implements the ML pipeline described by Topçuoğlu et al. (2020) <doi:10.1128/mBio.00434-20> with reasonable default options for data preprocessing, hyperparameter tuning, cross-validation, testing, model evaluation, and interpretation steps. See the website <https://www.schlosslab.org/mikropml/> for more information, documentation, and examples.

License MIT + file LICENSE

URL https://www.schlosslab.org/mikropml/,

https://github.com/SchlossLab/mikropml

BugReports https://github.com/SchlossLab/mikropml/issues

Depends R (>= 4.1.0)

Imports caret, dplyr, e1071, glmnet, kernlab, MLmetrics, randomForest, rlang, rpart, stats, utils, xgboost

Suggests assertthat, doFuture, forcats, foreach, future, future.apply, furrr, ggplot2, knitr, progress, progressr, purrr, rmarkdown, rsample, testthat, tidyr

VignetteBuilder knitr

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

Config/testthat/edition 3

NeedsCompilation no

Author Begüm Topçuoğlu [aut] (<https://orcid.org/0000-0003-3140-537X>),
Zena Lapp [aut] (<https://orcid.org/0000-0003-4674-2176>),
Kelly Sovacool [aut, cre] (<https://orcid.org/0000-0003-3283-829X>),
Evan Snitkin [aut] (<https://orcid.org/0000-0001-8409-278X>),
Jenna Wiens [aut] (<https://orcid.org/0000-0002-1057-7722>),
Patrick Schloss [aut] (<https://orcid.org/0000-0002-6935-4275>),
Nick Lesniak [ctb] (<https://orcid.org/0000-0001-9359-5194>),
Courtney Armour [ctb] (<https://orcid.org/0000-0002-5250-1224>),
Sarah Lucas [ctb] (<https://orcid.org/0000-0003-1676-5801>)

Maintainer Kelly Sovacool <sovacool@umich.edu>

Repository CRAN

Date/Publication 2023-08-21 15:10:05 UTC

R topics documented:

`bootstrap_performance` *Calculate a bootstrap confidence interval for the performance on a single train/test split*

Description

Uses `rsample::bootstraps()`, `rsample::int_pctl()`, and `furrr::future_map()`

Usage

```
bootstrap_performance(
  ml_result,
  outcome_colname,
  bootstrap_times = 10000,
  alpha = 0.05
)
```

Arguments

`ml_result` result returned from a single `run_ml()` call

`outcome_colname`
Column name as a string of the outcome variable (default `NULL`; the first column will be chosen automatically).

`bootstrap_times`
the number of boostraps to create (default: `10000`)

`alpha` the alpha level for the confidence interval (default `0.05` to create a 95% confidence interval)

Value

a data frame with an estimate (`.estimate`), lower bound (`.lower`), and upper bound (`.upper`) for each performance metric (`term`).

Author(s)

Kelly Sovacool, <sovacool@umich.edu>

Examples

```
bootstrap_performance(otu_mini_bin_results_glmnet, "dx",
  bootstrap_times = 10, alpha = 0.10
)
## Not run:
outcome_colname <- "dx"
run_ml(otu_mini_bin, "rf", outcome_colname = "dx") %>%
  bootstrap_performance(outcome_colname,
    bootstrap_times = 10000,
    alpha = 0.05
  )

## End(Not run)
```

---

calc_balanced_precision *Calculate balanced precision given actual and baseline precision*

Description

Implements Equation 1 from Wu *et al.* 2021 doi:10.1016/j.ajhg.2021.08.012. It is the same as Equation 7 if `AUPRC` (aka `prAUC`) is used in place of `precision`.

Usage

```
calc_balanced_precision(precision, prior)
```

Arguments

`precision`	actual precision of the model.
`prior`	baseline precision, aka frequency of positives. Can be calculated with calc_baseline_precision

Value

the expected precision if the data were balanced

Author(s)

Kelly Sovacool <sovacool@umich.edu>

Examples

```
prior <- calc_baseline_precision(otu_mini_bin,
  outcome_colname = "dx",
  pos_outcome = "cancer"
)
calc_balanced_precision(otu_mini_bin_results_rf$performance$Precision, prior)

otu_mini_bin_results_rf$performance %>%
  dplyr::mutate(
    balanced_precision = calc_balanced_precision(Precision, prior),
    aubprc = calc_balanced_precision(prAUC, prior)
  ) %>%
  dplyr::select(AUC, Precision, balanced_precision, aubprc)

# cumulative performance for a single model
sensspec_1 <- calc_model_sensspec(
  otu_mini_bin_results_glmnet$trained_model,
  otu_mini_bin_results_glmnet$test_data,
  "dx"
)
head(sensspec_1)
prior <- calc_baseline_precision(otu_mini_bin,
  outcome_colname = "dx",
  pos_outcome = "cancer"
)
sensspec_1 %>%
  dplyr::mutate(balanced_precision = calc_balanced_precision(precision, prior)) %>%
  dplyr::rename(recall = sensitivity) %>%
  calc_mean_perf(group_var = recall, sum_var = balanced_precision) %>%
  plot_mean_prc(ycol = mean_balanced_precision)
```

`calc_baseline_precision` *Calculate the fraction of positives, i.e. baseline precision for a PRC curve*

Description

Calculate the fraction of positives, i.e. baseline precision for a PRC curve

Usage

```
calc_baseline_precision(dataset, outcome_colname = NULL, pos_outcome = NULL)
```

Arguments

`dataset` Data frame with an outcome variable and other columns as features.

`outcome_colname` Column name as a string of the outcome variable (default `NULL`; the first column will be chosen automatically).

pos_outcome   the positive outcome from outcome_colname, e.g. "cancer" for the otu_mini_bin dataset.

Value

the baseline precision based on the fraction of positives

Author(s)

Kelly Sovacool, <sovacool@umich.edu>

Examples

```
# calculate the baseline precision
data.frame(y = c("a", "b", "a", "b")) %>%
  calc_baseline_precision(
    outcome_colname = "y",
    pos_outcome = "a"
  )


calc_baseline_precision(otu_mini_bin,
  outcome_colname = "dx",
  pos_outcome = "cancer"
)


# if you're not sure which outcome was used as the 'positive' outcome during
# model training, you can access it from the trained model and pass it along:
calc_baseline_precision(otu_mini_bin,
  outcome_colname = "dx",
  pos_outcome = otu_mini_bin_results_glmnet$trained_model$levels[1]
)
```

---

calc_mean_perf   *Generic function to calculate mean performance curves for multiple models*

Description

Used by calc_mean_roc() and calc_mean_prc().

Usage

```
calc_mean_perf(sensspec_dat, group_var = specificity, sum_var = sensitivity)
```

Arguments

`sensspec_dat`	data frame created by concatenating results of `calc_model_sensspec()` for multiple models.
`group_var`	variable to group by (e.g. specificity or recall).
`sum_var`	variable to summarize (e.g. sensitivity or precision).

Value

data frame with mean & standard deviation of `sum_var` summarized over `group_var`

Author(s)

Courtney Armour

Kelly Sovacool

`calc_model_sensspec` *Calculate and summarize performance for ROC and PRC plots*

Description

Use these functions to calculate cumulative sensitivity, specificity, recall, etc. on single models, concatenate the results together from multiple models, and compute mean ROC and PRC. You can then plot mean ROC and PRC curves to visualize the results. **Note**: These functions assume a binary outcome.

Usage

```
calc_model_sensspec(trained_model, test_data, outcome_colname = NULL)

calc_mean_roc(sensspec_dat)

calc_mean_prc(sensspec_dat)
```

Arguments

`trained_model`	Trained model from `caret::train()`.
`test_data`	Held out test data: dataframe of outcome and features.
`outcome_colname`	Column name as a string of the outcome variable (default `NULL`; the first column will be chosen automatically).
`sensspec_dat`	data frame created by concatenating results of `calc_model_sensspec()` for multiple models.

Value

data frame with summarized performance

Functions

- `calc_model_sensspec()`: Get sensitivity, specificity, and precision for a model.
- `calc_mean_roc()`: Calculate mean sensitivity over specificity for multiple models
- `calc_mean_prc()`: Calculate mean precision over recall for multiple models

Author(s)

Courtney Armour

Kelly Sovacool, <sovacool@umich.edu>

Examples

```
## Not run:
library(dplyr)
# get cumulative performance for a single model
sensspec_1 <- calc_model_sensspec(
  otu_mini_bin_results_glmnet$trained_model,
  otu_mini_bin_results_glmnet$test_data,
  "dx"
)
head(sensspec_1)

# get performance for multiple models
get_sensspec_seed <- function(seed) {
  ml_result <- run_ml(otu_mini_bin, "glmnet", seed = seed)
  sensspec <- calc_model_sensspec(
    ml_result$trained_model,
    ml_result$test_data,
    "dx"
  ) %>%
    dplyr::mutate(seed = seed)
  return(sensspec)
}
sensspec_dat <- purrr::map_dfr(seq(100, 102), get_sensspec_seed)

# calculate mean sensitivity over specificity
roc_dat <- calc_mean_roc(sensspec_dat)
head(roc_dat)

# calculate mean precision over recall
prc_dat <- calc_mean_prc(sensspec_dat)
head(prc_dat)

# plot ROC & PRC
roc_dat %>% plot_mean_roc()
baseline_prec <- calc_baseline_precision(otu_mini_bin, "dx", "cancer")
prc_dat %>%
  plot_mean_prc(baseline_precision = baseline_prec)

# balanced precision
prior <- calc_baseline_precision(otu_mini_bin,
```

```
  outcome_colname = "dx",
  pos_outcome = "cancer"
)
bprc_dat <- sensspec_dat %>%
  dplyr::mutate(balanced_precision = calc_balanced_precision(precision, prior)) %>%
  dplyr::rename(recall = sensitivity) %>%
  calc_mean_perf(group_var = recall, sum_var = balanced_precision)
bprc_dat %>% plot_mean_prc(ycol = mean_balanced_precision) + ylab("Mean Bal. Precision")

## End(Not run)
```

`calc_perf_metrics` *Get performance metrics for test data*

Description

Get performance metrics for test data

Usage

```
calc_perf_metrics(
  test_data,
  trained_model,
  outcome_colname,
  perf_metric_function,
  class_probs
)
```

Arguments

`test_data` Held out test data: dataframe of outcome and features.

`trained_model` Trained model from `caret::train()`.

`outcome_colname`
Column name as a string of the outcome variable (default `NULL`; the first column will be chosen automatically).

`perf_metric_function`
Function to calculate the performance metric to be used for cross-validation and test performance. Some functions are provided by caret (see `caret::defaultSummary()`). Defaults: binary classification = `twoClassSummary`, multi-class classification = `multiClassSummary`, regression = `defaultSummary`.

`class_probs` Whether to use class probabilities (TRUE for categorical outcomes, FALSE for numeric outcomes).

Value

Dataframe of performance metrics.

Author(s)

Zena Lapp, <zenalapp@umich.edu>

Examples

```
## Not run:
results <- run_ml(otu_small, "glmnet", kfold = 2, cv_times = 2)
calc_perf_metrics(results$test_data,
  results$trained_model,
  "dx",
  multiClassSummary,
  class_probs = TRUE
)

## End(Not run)
```

---

combine_hp_performance

Combine hyperparameter performance metrics for multiple train/test splits

---

Description

Combine hyperparameter performance metrics for multiple train/test splits generated by, for instance, looping in R or using a snakemake workflow on a high-performance computer.

Usage

```
combine_hp_performance(trained_model_lst)
```

Arguments

`trained_model_lst`
: List of trained models.

Value

Named list:

- `dat`: Dataframe of performance metric for each group of hyperparameters
- `params`: Hyperparameters tuned.
- `Metric`: Performance metric used.

Author(s)

Zena Lapp, <zenalapp@umich.edu>

Examples

```
## Not run:
results <- lapply(seq(100, 102), function(seed) {
  run_ml(otu_small, "glmnet", seed = seed, cv_times = 2, kfold = 2)
})
models <- lapply(results, function(x) x$trained_model)
combine_hp_performance(models)

## End(Not run)
```

---

compare_models — *Perform permutation tests to compare the performance metric across all pairs of a group variable.*

Description

A wrapper for `permute_p_value()`.

Usage

```
compare_models(merged_data, metric, group_name, nperm = 10000)
```

Arguments

`merged_data`	the concatenated performance data from `run_ml`
`metric`	metric to compare, must be numeric
`group_name`	column with group variables to compare
`nperm`	number of permutations, default=10000

Value

a table of p-values for all pairs of group variable

Author(s)

Courtney R Armour, <armourc@umich.edu>

Examples

```
df <- dplyr::tibble(
  model = c("rf", "rf", "glmnet", "glmnet", "svmRadial", "svmRadial"),
  AUC = c(.2, 0.3, 0.8, 0.9, 0.85, 0.95)
)
set.seed(123)
compare_models(df, "AUC", "model", nperm = 10)
```

define_cv *Define cross-validation scheme and training parameters*

Description

Define cross-validation scheme and training parameters

Usage

```
define_cv(
  train_data,
  outcome_colname,
  hyperparams_list,
  perf_metric_function,
  class_probs,
  kfold = 5,
  cv_times = 100,
  groups = NULL,
  group_partitions = NULL
)
```

Arguments

`train_data` Dataframe for training model.

`outcome_colname`
Column name as a string of the outcome variable (default `NULL`; the first column will be chosen automatically).

`hyperparams_list`
Named list of lists of hyperparameters.

`perf_metric_function`
Function to calculate the performance metric to be used for cross-validation and test performance. Some functions are provided by caret (see `caret::defaultSummary()`). Defaults: binary classification = `twoClassSummary`, multi-class classification = `multiClassSummary`, regression = `defaultSummary`.

`class_probs` Whether to use class probabilities (TRUE for categorical outcomes, FALSE for numeric outcomes).

`kfold` Fold number for k-fold cross-validation (default: `5`).

`cv_times` Number of cross-validation partitions to create (default: `100`).

`groups` Vector of groups to keep together when splitting the data into train and test sets. If the number of groups in the training set is larger than `kfold`, the groups will also be kept together for cross-validation. Length matches the number of rows in the dataset (default: `NULL`).

`group_partitions`
Specify how to assign `groups` to the training and testing partitions (default: `NULL`). If `groups` specifies that some samples belong to group `"A"` and some

belong to group "B", then setting group_partitions = list(train = c("A", "B"), test = c("B")) will result in all samples from group "A" being placed in the training set, some samples from "B" also in the training set, and the remaining samples from "B" in the testing set. The partition sizes will be as close to training_frac as possible. If the number of groups in the training set is larger than kfold, the groups will also be kept together for cross-validation.

Value

Caret object for trainControl that controls cross-validation

Author(s)

Begüm Topçuoğlu, <topcuoglu.begum@gmail.com>

Kelly Sovacool, <sovacool@umich.edu>

Examples

```
training_inds <- get_partition_indices(otu_small %>% dplyr::pull("dx"),
  training_frac = 0.8,
  groups = NULL
)
train_data <- otu_small[training_inds, ]
test_data <- otu_small[-training_inds, ]
cv <- define_cv(train_data,
  outcome_colname = "dx",
  hyperparams_list = get_hyperparams_list(otu_small, "glmnet"),
  perf_metric_function = caret::multiClassSummary,
  class_probs = TRUE,
  kfold = 5
)
```

---

get_caret_processed_df

Get preprocessed dataframe for continuous variables

Description

Get preprocessed dataframe for continuous variables

Usage

```
get_caret_processed_df(features, method)
```

Arguments

features	Dataframe of features for machine learning
method	Methods to preprocess the data, described in caret::preProcess() (default: c("center","scale"), use NULL for no normalization).

Value

Named list:

- processed: Dataframe of processed features.
- removed: Names of any features removed during preprocessing.

Author(s)

Zena Lapp, <zenalapp@umich.edu>

Examples

```
get_caret_processed_df(mikropml::otu_small[, 2:ncol(otu_small)], c("center", "scale"))
```

---

get_feature_importance   *Get feature importance using the permutation method*

Description

Calculates feature importance using a trained model and test data. Requires the future.apply package.

Usage

```
get_feature_importance(
  trained_model,
  test_data,
  outcome_colname,
  perf_metric_function,
  perf_metric_name,
  class_probs,
  method,
  seed = NA,
  corr_thresh = 1,
  groups = NULL,
  nperms = 100,
  corr_method = "spearman"
)
```

Arguments

trained_model	Trained model from caret::train().
test_data	Held out test data: dataframe of outcome and features.
outcome_colname	Column name as a string of the outcome variable (default NULL; the first column will be chosen automatically).

`perf_metric_function`
: Function to calculate the performance metric to be used for cross-validation and test performance. Some functions are provided by caret (see `caret::defaultSummary()`). Defaults: binary classification = `twoClassSummary`, multi-class classification = `multiClassSummary`, regression = `defaultSummary`.

`perf_metric_name`
: The column name from the output of the function provided to perf_metric_function that is to be used as the performance metric. Defaults: binary classification = `"ROC"`, multi-class classification = `"logLoss"`, regression = `"RMSE"`.

`class_probs`
: Whether to use class probabilities (TRUE for categorical outcomes, FALSE for numeric outcomes).

`method`
: ML method. Options: `c("glmnet", "rf", "rpart2", "svmRadial", "xgbTree")`.

  - glmnet: linear, logistic, or multiclass regression
  - rf: random forest
  - rpart2: decision tree
  - svmRadial: support vector machine
  - xgbTree: xgboost

`seed`
: Random seed (default: `NA`). Your results will only be reproducible if you set a seed.

`corr_thresh`
: For feature importance, group correlations above or equal to `corr_thresh` (range `0` to `1`; default: `1`).

`groups`
: Vector of feature names to group together during permutation. Each element should be a string with feature names separated by a pipe character (`|`). If this is `NULL` (default), correlated features will be grouped together based on `corr_thresh`.

`nperms`
: number of permutations to perform (default: `100`).

`corr_method`
: correlation method. options or the same as those supported by `stats::cor`: spearman, pearson, kendall. (default: spearman)

Details

For permutation tests, the p-value is the number of permutation statistics that are greater than the test statistic, divided by the number of permutations. In our case, the permutation statistic is the model performance (e.g. AUROC) after randomizing the order of observations for one feature, and the test statistic is the actual performance on the test data. By default we perform 100 permutations per feature; increasing this will increase the precision of estimating the null distribution, but also increases runtime. The p-value represents the probability of obtaining the actual performance in the event that the null hypothesis is true, where the null hypothesis is that the feature is not important for model performance.

We strongly recommend providing multiple cores to speed up computation time. See our vignette on parallel processing for more details.

Value

Data frame with performance metrics for when each feature (or group of correlated features; `feat`) is permuted (`perf_metric`), differences between the actual test performance metric on and the permuted performance metric (perf_metric_diff; test minus permuted performance), and the p-value (pvalue: the probability of obtaining the actual performance value under the null hypothesis). Features with a larger perf_metric_diff are more important. The performance metric name (perf_metric_name) and seed (seed) are also returned.

Author(s)

Begüm Topçuoğlu, <topcuoglu.begum@gmail.com>

Zena Lapp, <zenalapp@umich.edu>

Kelly Sovacool, <sovacool@umich.edu>

Examples

```
## Not run:
# If you called `run_ml()` with `feature_importance = FALSE` (the default),
# you can use `get_feature_importance()` later as long as you have the
# trained model and test data.
results <- run_ml(otu_small, "glmnet", kfold = 2, cv_times = 2)
names(results$trained_model$trainingData)[1] <- "dx"
feat_imp <- get_feature_importance(results$trained_model,
  results$trained_model$trainingData,
  results$test_data,
  "dx",
  multiClassSummary,
  "AUC",
  class_probs = TRUE,
  method = "glmnet"
)

# We strongly recommend providing multiple cores to speed up computation time.
# Do this before calling `get_feature_importance()`.
doFuture::registerDoFuture()
future::plan(future::multicore, workers = 2)

# Optionally, you can group features together with a custom grouping
feat_imp <- get_feature_importance(results$trained_model,
  results$trained_model$trainingData,
  results$test_data,
  "dx",
  multiClassSummary,
  "AUC",
  class_probs = TRUE,
  method = "glmnet",
  groups = c(
    "Otu00007", "Otu00008", "Otu00009", "Otu00011", "Otu00012",
    "Otu00015", "Otu00016", "Otu00018", "Otu00019", "Otu00020", "Otu00022",
    "Otu00023", "Otu00025", "Otu00028", "Otu00029", "Otu00030", "Otu00035",
    "Otu00036", "Otu00037", "Otu00038", "Otu00039", "Otu00040", "Otu00047",
    "Otu00050", "Otu00052", "Otu00054", "Otu00055", "Otu00056", "Otu00060",
    "Otu00003|Otu00002|Otu00005|Otu00024|Otu00032|Otu00041|Otu00053",
    "Otu00014|Otu00021|Otu00017|Otu00031|Otu00057",
    "Otu00013|Otu00006", "Otu00026|Otu00001|Otu00034|Otu00048",
```

```
      "Otu00033|Otu00010",
      "Otu00042|Otu00004", "Otu00043|Otu00027|Otu00049", "Otu00051|Otu00045",
      "Otu00058|Otu00044", "Otu00059|Otu00046"
    )
  )

  # the function can show a progress bar if you have the `progressr` package installed.
  ## optionally, specify the progress bar format:
  progressr::handlers(progressr::handler_progress(
    format = ":message :bar :percent | elapsed: :elapsed | eta: :eta",
    clear = FALSE,
    show_after = 0
  ))
  ## tell progressr to always report progress
  progressr::handlers(global = TRUE)
  ## run the function and watch the live progress udpates
  feat_imp <- get_feature_importance(results$trained_model,
    results$trained_model$trainingData,
    results$test_data,
    "dx",
    multiClassSummary,
    "AUC",
    class_probs = TRUE,
    method = "glmnet"
  )

  # You can specify any correlation method supported by `stats::cor`:
  feat_imp <- get_feature_importance(results$trained_model,
    results$trained_model$trainingData,
    results$test_data,
    "dx",
    multiClassSummary,
    "AUC",
    class_probs = TRUE,
    method = "glmnet",
    corr_method = "pearson"
  )

  ## End(Not run)
```

`get_hp_performance` *Get hyperparameter performance metrics*

Description

Get hyperparameter performance metrics

Usage

```
get_hp_performance(trained_model)
```

Arguments

`trained_model` trained model (e.g. from `run_ml()`)

Value

Named list:

- `dat`: Dataframe of performance metric for each group of hyperparameters.
- `params`: Hyperparameters tuned.
- `metric`: Performance metric used.

Author(s)

Zena Lapp, <zenalapp@umich.edu>

Kelly Sovacool <sovacool@umich.edu>

Examples

```
get_hp_performance(otu_mini_bin_results_glmnet$trained_model)
```

`get_hyperparams_list` *Set hyperparameters based on ML method and dataset characteristics*

Description

For more details see the vignette on hyperparameter tuning.

Usage

```
get_hyperparams_list(dataset, method)
```

Arguments

`dataset` Data frame with an outcome variable and other columns as features.

`method` ML method. Options: `c("glmnet", "rf", "rpart2", "svmRadial", "xgbTree")`.

- glmnet: linear, logistic, or multiclass regression
- rf: random forest
- rpart2: decision tree
- svmRadial: support vector machine
- xgbTree: xgboost

Value

Named list of hyperparameters.

Author(s)

Kelly Sovacool, <sovacool@umich.edu>

Examples

```
get_hyperparams_list(otu_mini_bin, "rf")
get_hyperparams_list(otu_small, "rf")
get_hyperparams_list(otu_mini_bin, "rpart2")
get_hyperparams_list(otu_small, "rpart2")
```

---

get_outcome_type *Get outcome type.*

Description

If the outcome is numeric, the type is continuous. Otherwise, the outcome type is binary if there are only two outcomes or multiclass if there are more than two outcomes.

Usage

```
get_outcome_type(outcomes_vec)
```

Arguments

outcomes_vec	Vector of outcomes.

Value

Outcome type (continuous, binary, or multiclass).

Author(s)

Zena Lapp, <zenalapp@umich.edu>

Examples

```
get_outcome_type(c(1, 2, 1))
get_outcome_type(c("a", "b", "b"))
get_outcome_type(c("a", "b", "c"))
```

get_partition_indices *Select indices to partition the data into training & testing sets.*

Description

Use this function to get the row indices for the training set.

Usage

```
get_partition_indices(
  outcomes,
  training_frac = 0.8,
  groups = NULL,
  group_partitions = NULL
)
```

Arguments

`outcomes`	vector of outcomes
`training_frac`	Fraction of data for training set (default: `0.8`). Rows from the dataset will be randomly selected for the training set, and all remaining rows will be used in the testing set. Alternatively, if you provide a vector of integers, these will be used as the row indices for the training set. All remaining rows will be used in the testing set.
`groups`	Vector of groups to keep together when splitting the data into train and test sets. If the number of groups in the training set is larger than `kfold`, the groups will also be kept together for cross-validation. Length matches the number of rows in the dataset (default: `NULL`).
`group_partitions`	Specify how to assign `groups` to the training and testing partitions (default: `NULL`). If `groups` specifies that some samples belong to group `"A"` and some belong to group `"B"`, then setting `group_partitions = list(train = c("A", "B"), test = c("B"))` will result in all samples from group `"A"` being placed in the training set, some samples from `"B"` also in the training set, and the remaining samples from `"B"` in the testing set. The partition sizes will be as close to `training_frac` as possible. If the number of groups in the training set is larger than `kfold`, the groups will also be kept together for cross-validation.

Details

If `groups` is `NULL`, uses createDataPartition. Otherwise, uses `create_grouped_data_partition()`.

Set the seed prior to calling this function if you would like your data partitions to be reproducible (recommended).

Value

Vector of row indices for the training set.

Author(s)

Kelly Sovacool, sovacool@umich.edu

Examples

```
training_inds <- get_partition_indices(otu_mini_bin$dx)
train_data <- otu_mini_bin[training_inds, ]
test_data <- otu_mini_bin[-training_inds, ]
```

---

get_performance_tbl *Get model performance metrics as a one-row tibble*

Description

Get model performance metrics as a one-row tibble

Usage

```
get_performance_tbl(
  trained_model,
  test_data,
  outcome_colname,
  perf_metric_function,
  perf_metric_name,
  class_probs,
  method,
  seed = NA
)
```

Arguments

`trained_model` Trained model from `caret::train()`.

`test_data` Held out test data: dataframe of outcome and features.

`outcome_colname` Column name as a string of the outcome variable (default `NULL`; the first column will be chosen automatically).

`perf_metric_function` Function to calculate the performance metric to be used for cross-validation and test performance. Some functions are provided by caret (see `caret::defaultSummary()`). Defaults: binary classification = `twoClassSummary`, multi-class classification = `multiClassSummary`, regression = `defaultSummary`.

`perf_metric_name` The column name from the output of the function provided to perf_metric_function that is to be used as the performance metric. Defaults: binary classification = `"ROC"`, multi-class classification = `"logLoss"`, regression = `"RMSE"`.

`class_probs` Whether to use class probabilities (TRUE for categorical outcomes, FALSE for numeric outcomes).

`method` ML method. Options: `c("glmnet", "rf", "rpart2", "svmRadial", "xgbTree")`.

- glmnet: linear, logistic, or multiclass regression
- rf: random forest
- rpart2: decision tree
- svmRadial: support vector machine
- xgbTree: xgboost

`seed` Random seed (default: `NA`). Your results will only be reproducible if you set a seed.

Value

A one-row tibble with a column for the cross-validation performance, columns for each of the performance metrics for the test data, plus the `method`, and `seed`.

Author(s)

Kelly Sovacool, <sovacool@umich.edu>

Zena Lapp, <zenalapp@umich.edu>

Examples

```
## Not run:
results <- run_ml(otu_small, "glmnet", kfold = 2, cv_times = 2)
names(results$trained_model$trainingData)[1] <- "dx"
get_performance_tbl(results$trained_model, results$test_data,
  "dx",
  multiClassSummary, "AUC",
  class_probs = TRUE,
  method = "glmnet"
)

## End(Not run)
```

`get_perf_metric_fn` *Get default performance metric function*

Description

Get default performance metric function

Usage

```
get_perf_metric_fn(outcome_type)
```

Arguments

`outcome_type`	Type of outcome (one of: `"continuous"`,`"binary"`,`"multiclass"`).

Value

Performance metric function.

Author(s)

Zena Lapp, `<zenalapp@umich.edu>`

Examples

```
get_perf_metric_fn("continuous")
get_perf_metric_fn("binary")
get_perf_metric_fn("multiclass")
```

`get_perf_metric_name` *Get default performance metric name*

Description

Get default performance metric name for cross-validation.

Usage

```
get_perf_metric_name(outcome_type)
```

Arguments

`outcome_type`	Type of outcome (one of: `"continuous"`,`"binary"`,`"multiclass"`).

Value

Performance metric name.

Author(s)

Zena Lapp, `<zenalapp@umich.edu>`

Examples

```
get_perf_metric_name("continuous")
get_perf_metric_name("binary")
get_perf_metric_name("multiclass")
```

get_tuning_grid *Generate the tuning grid for tuning hyperparameters*

Description

Generate the tuning grid for tuning hyperparameters

Usage

```
get_tuning_grid(hyperparams_list, method)
```

Arguments

`hyperparams_list`
: Named list of lists of hyperparameters.

`method`
: ML method. Options: `c("glmnet", "rf", "rpart2", "svmRadial", "xgbTree")`.
  - glmnet: linear, logistic, or multiclass regression
  - rf: random forest
  - rpart2: decision tree
  - svmRadial: support vector machine
  - xgbTree: xgboost

Value

The tuning grid.

Author(s)

Begüm Topçuoğlu, <topcuoglu.begum@gmail.com>

Kelly Sovacool, <sovacool@umich.edu>

Examples

```
ml_method <- "glmnet"
hparams_list <- get_hyperparams_list(otu_small, ml_method)
get_tuning_grid(hparams_list, ml_method)
```

group_correlated_features

Group correlated features

Description

Group correlated features

Usage

```
group_correlated_features(
  features,
  corr_thresh = 1,
  group_neg_corr = TRUE,
  corr_method = "spearman"
)
```

Arguments

features	a dataframe with each column as a feature for ML
corr_thresh	For feature importance, group correlations above or equal to corr_thresh (range 0 to 1; default: 1).
group_neg_corr	Whether to group negatively correlated features together (e.g. c(0,1) and c(1,0)).
corr_method	correlation method. options or the same as those supported by stats::cor: spearman, pearson, kendall. (default: spearman)

Value

vector where each element is a group of correlated features separated by pipes (|)

Author(s)

Kelly Sovacool, <sovacool@umich.edu>

Examples

```
features <- data.frame(
  a = 1:3, b = 2:4, c = c(1, 0, 1),
  d = (5:7), e = c(5, 1, 4), f = c(-1, 0, -1)
)
group_correlated_features(features)
```

otu_data_preproc *Mini OTU abundance dataset - preprocessed*

Description

This is the result of running `preprocess_data("otu_mini_bin")`

Usage

```
otu_data_preproc
```

Format

An object of class `list` of length 3.

---

otu_mini_bin *Mini OTU abundance dataset*

Description

A dataset containing relatives abundances of OTUs for human stool samples with a binary outcome, `dx`. This is a subset of `otu_small`.

Usage

```
otu_mini_bin
```

Format

A data frame The `dx` column is the diagnosis: healthy or cancerous (colorectal). All other columns are OTU relative abundances.

---

otu_mini_bin_results_glmnet *Results from running the pipeline with L2 logistic regression on `otu_mini_bin` with feature importance and grouping*

Description

Results from running the pipeline with L2 logistic regression on `otu_mini_bin` with feature importance and grouping

Usage

```
otu_mini_bin_results_glmnet
```

Format

An object of class list of length 4.

---

otu_mini_bin_results_rf

Results from running the pipeline with random forest on otu_mini_bin

---

Description

Results from running the pipeline with random forest on otu_mini_bin

Usage

```
otu_mini_bin_results_rf
```

Format

An object of class list of length 4.

---

otu_mini_bin_results_rpart2

Results from running the pipeline with rpart2 on otu_mini_bin

---

Description

Results from running the pipeline with rpart2 on otu_mini_bin

Usage

```
otu_mini_bin_results_rpart2
```

Format

An object of class list of length 4.

`otu_mini_bin_results_svmRadial`
Results from running the pipeline with svmRadial on `otu_mini_bin`

Description

Results from running the pipeline with svmRadial on `otu_mini_bin`

Usage

```
otu_mini_bin_results_svmRadial
```

Format

An object of class `list` of length 4.

`otu_mini_bin_results_xgbTree`
Results from running the pipeline with xbgTree on `otu_mini_bin`

Description

Results from running the pipeline with xbgTree on `otu_mini_bin`

Usage

```
otu_mini_bin_results_xgbTree
```

Format

An object of class `list` of length 4.

otu_mini_cont_results_glmnet

Results from running the pipeline with glmnet on otu_mini_bin *with* Otu00001 *as the outcome*

Description

Results from running the pipeline with glmnet on otu_mini_bin with Otu00001 as the outcome

Usage

```
otu_mini_cont_results_glmnet
```

Format

An object of class list of length 4.

otu_mini_cont_results_nocv

Results from running the pipeline with glmnet on otu_mini_bin *with* Otu00001 *as the outcome column, using a custom train control scheme that does not perform cross-validation*

Description

Results from running the pipeline with glmnet on otu_mini_bin with Otu00001 as the outcome column, using a custom train control scheme that does not perform cross-validation

Usage

```
otu_mini_cont_results_nocv
```

Format

An object of class list of length 4.

`otu_mini_cv` *Cross validation on `train_data_mini` with grouped features.*

Description

Cross validation on `train_data_mini` with grouped features.

Usage

```
otu_mini_cv
```

Format

An object of class `list` of length 27.

`otu_mini_multi` *Mini OTU abundance dataset with 3 categorical variables*

Description

A dataset containing relatives abundances of OTUs for human stool samples

Usage

```
otu_mini_multi
```

Format

A data frame The `dx` column is the colorectal cancer diagnosis: adenoma, carcinoma, normal. All other columns are OTU relative abundances.

`otu_mini_multi_group` *Groups for otu_mini_multi*

Description

Groups for otu_mini_multi

Usage

```
otu_mini_multi_group
```

Format

An object of class `character` of length 490.

`otu_mini_multi_results_glmnet`
Results from running the pipeline with glmnet on `otu_mini_multi` *for multiclass outcomes*

Description

Results from running the pipeline with glmnet on `otu_mini_multi` for multiclass outcomes

Usage

```
otu_mini_multi_results_glmnet
```

Format

An object of class `list` of length 4.

`otu_small` *Small OTU abundance dataset*

Description

A dataset containing relatives abundances of 60 OTUs for 60 human stool samples. This is a subset of the data provided in `extdata/otu_large.csv`, which was used in Topçuoğlu *et al.* 2020.

Usage

```
otu_small
```

Format

A data frame with 60 rows and 61 variables. The `dx` column is the diagnosis: healthy or cancerous (colorectal). All other columns are OTU relative abundances.

permute_p_value *Calculated a permuted p-value comparing two models*

Description

Calculated a permuted p-value comparing two models

Usage

```
permute_p_value(
  merged_data,
  metric,
  group_name,
  group_1,
  group_2,
  nperm = 10000
)
```

Arguments

merged_data	the concatenated performance data from run_ml
metric	metric to compare, must be numeric
group_name	column with group variables to compare
group_1	name of one group to compare
group_2	name of other group to compare
nperm	number of permutations, default=10000

Value

numeric p-value comparing two models

Author(s)

Begüm Topçuoğlu, <topcuoglu.begum@gmail.com>

Courtney R Armour, <armourc@umich.edu>

Examples

```
df <- dplyr::tibble(
  model = c("rf", "rf", "glmnet", "glmnet", "svmRadial", "svmRadial"),
  AUC = c(.2, 0.3, 0.8, 0.9, 0.85, 0.95)
)
set.seed(123)
permute_p_value(df, "AUC", "model", "rf", "glmnet", nperm = 100)
```

plot_hp_performance *Plot hyperparameter performance metrics*

Description

Plot hyperparameter performance metrics

Usage

```
plot_hp_performance(dat, param_col, metric_col)
```

Arguments

dat	dataframe of hyperparameters and performance metric (e.g. from `get_hp_performance()` or `combine_hp_performance()`)
param_col	hyperparameter to be plotted. must be a column in `dat`.
metric_col	performance metric. must be a column in `dat`.

Value

ggplot of hyperparameter performance.

Author(s)

Zena Lapp, <zenalapp@umich.edu>

Kelly Sovacool <sovacool@umich.edu>

Examples

```
# plot for a single `run_ml()` call
hp_metrics <- get_hp_performance(otu_mini_bin_results_glmnet$trained_model)
hp_metrics
plot_hp_performance(hp_metrics$dat, lambda, AUC)
## Not run:
# plot for multiple `run_ml()` calls
results <- lapply(seq(100, 102), function(seed) {
  run_ml(otu_small, "glmnet", seed = seed)
})
models <- lapply(results, function(x) x$trained_model)
hp_metrics <- combine_hp_performance(models)
plot_hp_performance(hp_metrics$dat, lambda, AUC)

## End(Not run)
```

plot_mean_roc *Plot ROC and PRC curves*

Description

Plot ROC and PRC curves

Usage

```
plot_mean_roc(dat, ribbon_fill = "#C6DBEF", line_color = "#08306B")

plot_mean_prc(
  dat,
  baseline_precision = NULL,
  ycol = mean_precision,
  ribbon_fill = "#C7E9C0",
  line_color = "#00441B"
)
```

Arguments

`dat`	sensitivity, specificity, and precision data calculated by `calc_mean_roc()`
`ribbon_fill`	ribbon fill color (default: "#D9D9D9")
`line_color`	line color (default: "#000000")
`baseline_precision`	baseline precision from `calc_baseline_precision()`
`ycol`	column for the y axis (Default: `mean_precision`)

Functions

- `plot_mean_roc()`: Plot mean sensitivity over specificity
- `plot_mean_prc()`: Plot mean precision over recall

Author(s)

Courtney Armour

Kelly Sovacool <sovacool@umich.edu>

Examples

```
## Not run:
library(dplyr)
# get performance for multiple models
get_sensspec_seed <- function(seed) {
  ml_result <- run_ml(otu_mini_bin, "glmnet", seed = seed)
  sensspec <- calc_model_sensspec(
```

```
    ml_result$trained_model,
    ml_result$test_data,
    "dx"
  ) %>%
    mutate(seed = seed)
  return(sensspec)
}
sensspec_dat <- purrr::map_dfr(seq(100, 102), get_sensspec_seed)

# plot ROC & PRC
sensspec_dat %>%
  calc_mean_roc() %>%
  plot_mean_roc()
baseline_prec <- calc_baseline_precision(otu_mini_bin, "dx", "cancer")
sensspec_dat %>%
  calc_mean_prc() %>%
  plot_mean_prc(baseline_precision = baseline_prec)

## End(Not run)
```

`plot_model_performance` *Plot performance metrics for multiple ML runs with different parameters*

Description

ggplot2 is required to use this function.

Usage

```
plot_model_performance(performance_df)
```

Arguments

`performance_df` dataframe of performance results from multiple calls to `run_ml()`

Value

A ggplot2 plot of performance.

Author(s)

Begüm Topçuoglu, <topcuoglu.begum@gmail.com>

Kelly Sovacool, <sovacool@umich.edu>

Examples

```
## Not run:
# call `run_ml()` multiple times with different seeds
results_lst <- lapply(seq(100, 104), function(seed) {
  run_ml(otu_small, "glmnet", seed = seed)
})
# extract and combine the performance results
perf_df <- lapply(results_lst, function(result) {
  result[["performance"]]
}) %>%
  dplyr::bind_rows()
# plot the performance results
p <- plot_model_performance(perf_df)


# call `run_ml()` with different ML methods
param_grid <- expand.grid(
  seeds = seq(100, 104),
  methods = c("glmnet", "rf")
)
results_mtx <- mapply(
  function(seed, method) {
    run_ml(otu_mini_bin, method, seed = seed, kfold = 2)
  },
  param_grid$seeds, param_grid$methods
)
# extract and combine the performance results
perf_df2 <- dplyr::bind_rows(results_mtx["performance", ])
# plot the performance results
p <- plot_model_performance(perf_df2)

# you can continue adding layers to customize the plot
p +
  theme_classic() +
  scale_color_brewer(palette = "Dark2") +
  coord_flip()

## End(Not run)
```

---

preprocess_data *Preprocess data prior to running machine learning*

Description

Function to preprocess your data for input into run_ml().

Usage

```
preprocess_data(
```

```
    dataset,
    outcome_colname,
    method = c("center", "scale"),
    remove_var = "nzv",
    collapse_corr_feats = TRUE,
    to_numeric = TRUE,
    group_neg_corr = TRUE,
    prefilter_threshold = 1
  )
```

Arguments

`dataset` Data frame with an outcome variable and other columns as features.

`outcome_colname` Column name as a string of the outcome variable (default `NULL`; the first column will be chosen automatically).

`method` Methods to preprocess the data, described in `caret::preProcess()` (default: `c("center","scale")`, use `NULL` for no normalization).

`remove_var` Whether to remove variables with near-zero variance (`'nzv'`; default), zero variance (`'zv'`), or none (`NULL`).

`collapse_corr_feats` Whether to keep only one of perfectly correlated features.

`to_numeric` Whether to change features to numeric where possible.

`group_neg_corr` Whether to group negatively correlated features together (e.g. c(0,1) and c(1,0)).

`prefilter_threshold` Remove features which only have non-zero & non-NA values N rows or fewer (default: 1). Set this to -1 to keep all columns at this step. This step will also be skipped if `to_numeric` is set to `FALSE`.

Value

Named list including:

- `dat_transformed`: Preprocessed data.
- `grp_feats`: If features were grouped together, a named list of the features corresponding to each group.
- `removed_feats`: Any features that were removed during preprocessing (e.g. because there was zero variance or near-zero variance for those features).

If the `progressr` package is installed, a progress bar with time elapsed and estimated time to completion can be displayed.

More details

See the preprocessing vignette for more details.

Note that if any values in `outcome_colname` contain spaces, they will be converted to underscores for compatibility with `caret`.

Author(s)

Zena Lapp, <zenalapp@umich.edu>

Kelly Sovacool, <sovacool@umich.edu>

Examples

```
preprocess_data(mikropml::otu_small, "dx")

# the function can show a progress bar if you have the progressr package installed
## optionally, specify the progress bar format
progressr::handlers(progressr::handler_progress(
  format = ":message :bar :percent | elapsed: :elapsed | eta: :eta",
  clear = FALSE,
  show_after = 0
))
## tell progressor to always report progress
## Not run:
progressr::handlers(global = TRUE)
## run the function and watch the live progress udpates
dat_preproc <- preprocess_data(mikropml::otu_small, "dx")

## End(Not run)
```

---

randomize_feature_order

Randomize feature order to eliminate any position-dependent effects

---

Description

Randomize feature order to eliminate any position-dependent effects

Usage

```
randomize_feature_order(dataset, outcome_colname)
```

Arguments

`dataset`	Data frame with an outcome variable and other columns as features.
`outcome_colname`	Column name as a string of the outcome variable (default `NULL`; the first column will be chosen automatically).

Value

Dataset with feature order randomized.

Author(s)

Nick Lesniak, <nlesniak@umich.edu>

Kelly Sovacool, <sovacool@umich.edu>

Examples

```
dat <- data.frame(
  outcome = c("1", "2", "3"),
  a = 4:6, b = 7:9, c = 10:12, d = 13:15
)
randomize_feature_order(dat, "outcome")
```

`remove_singleton_columns`

Remove columns appearing in only `threshold` *row(s) or fewer.*

Description

Removes columns which only have non-zero & non-NA values in `threshold` row(s) or fewer.

Usage

```
remove_singleton_columns(dat, threshold = 1)
```

Arguments

`dat`	dataframe
`threshold`	Number of rows. If a column only has non-zero & non-NA values in `threshold` row(s) or fewer, it will be removed.

Value

dataframe without singleton columns

Author(s)

Kelly Sovacool, <sovacool@umich.edu>

Courtney Armour

Examples

```
remove_singleton_columns(data.frame(a = 1:3, b = c(0, 1, 0), c = 4:6))
remove_singleton_columns(data.frame(a = 1:3, b = c(0, 1, 0), c = 4:6), threshold = 0)
remove_singleton_columns(data.frame(a = 1:3, b = c(0, 1, NA), c = 4:6))
remove_singleton_columns(data.frame(a = 1:3, b = c(1, 1, 1), c = 4:6))
```

`replace_spaces` *Replace spaces in all elements of a character vector with underscores*

Description

Replace spaces in all elements of a character vector with underscores

Usage

```
replace_spaces(x, new_char = "_")
```

Arguments

`x`	a character vector
`new_char`	the character to replace spaces (default: `_`)

Value

character vector with all spaces replaced with `new_char`

Author(s)

Kelly Sovacool, <sovacool@umich.edu>

Examples

```
dat <- data.frame(
  dx = c("outcome 1", "outcome 2", "outcome 1"),
  a = 1:3, b = c(5, 7, 1)
)
dat$dx <- replace_spaces(dat$dx)
dat
```

`run_ml` *Run the machine learning pipeline*

Description

This function splits the data set into a train & test set, trains machine learning (ML) models using k-fold cross-validation, evaluates the best model on the held-out test set, and optionally calculates feature importance using the framework outlined in Topçuoğlu *et al.* 2020 (doi:10.1128/mBio.0043420). Required inputs are a data frame (must contain an outcome variable and all other columns as features) and the ML method. See `vignette('introduction')` for more details.

Usage

```
run_ml(
  dataset,
  method,
  outcome_colname = NULL,
  hyperparameters = NULL,
  find_feature_importance = FALSE,
  calculate_performance = TRUE,
  kfold = 5,
  cv_times = 100,
  cross_val = NULL,
  training_frac = 0.8,
  perf_metric_function = NULL,
  perf_metric_name = NULL,
  groups = NULL,
  group_partitions = NULL,
  corr_thresh = 1,
  seed = NA,
  ...
)
```

Arguments

`dataset` Data frame with an outcome variable and other columns as features.

`method` ML method. Options: `c("glmnet", "rf", "rpart2", "svmRadial", "xgbTree")`.

- glmnet: linear, logistic, or multiclass regression
- rf: random forest
- rpart2: decision tree
- svmRadial: support vector machine
- xgbTree: xgboost

`outcome_colname` Column name as a string of the outcome variable (default `NULL`; the first column will be chosen automatically).

`hyperparameters` Dataframe of hyperparameters (default `NULL`; sensible defaults will be chosen automatically).

`find_feature_importance` Run permutation importance (default: `FALSE`). `TRUE` is recommended if you would like to identify features important for predicting your outcome, but it is resource-intensive.

`calculate_performance` Whether to calculate performance metrics (default: `TRUE`). You might choose to skip this if you do not perform cross-validation during model training.

`kfold` Fold number for k-fold cross-validation (default: `5`).

`cv_times` Number of cross-validation partitions to create (default: `100`).

cross_val
: a custom cross-validation scheme from caret::trainControl() (default: NULL, uses kfold cross validation repeated cv_times). kfold and cv_times are ignored if the user provides a custom cross-validation scheme. See the caret::trainControl() docs for information on how to use it.

training_frac
: Fraction of data for training set (default: 0.8). Rows from the dataset will be randomly selected for the training set, and all remaining rows will be used in the testing set. Alternatively, if you provide a vector of integers, these will be used as the row indices for the training set. All remaining rows will be used in the testing set.

perf_metric_function
: Function to calculate the performance metric to be used for cross-validation and test performance. Some functions are provided by caret (see caret::defaultSummary()). Defaults: binary classification = twoClassSummary, multi-class classification = multiClassSummary, regression = defaultSummary.

perf_metric_name
: The column name from the output of the function provided to perf_metric_function that is to be used as the performance metric. Defaults: binary classification = "ROC", multi-class classification = "logLoss", regression = "RMSE".

groups
: Vector of groups to keep together when splitting the data into train and test sets. If the number of groups in the training set is larger than kfold, the groups will also be kept together for cross-validation. Length matches the number of rows in the dataset (default: NULL).

group_partitions
: Specify how to assign groups to the training and testing partitions (default: NULL). If groups specifies that some samples belong to group "A" and some belong to group "B", then setting group_partitions = list(train = c("A", "B"), test = c("B")) will result in all samples from group "A" being placed in the training set, some samples from "B" also in the training set, and the remaining samples from "B" in the testing set. The partition sizes will be as close to training_frac as possible. If the number of groups in the training set is larger than kfold, the groups will also be kept together for cross-validation.

corr_thresh
: For feature importance, group correlations above or equal to corr_thresh (range 0 to 1; default: 1).

seed
: Random seed (default: NA). Your results will only be reproducible if you set a seed.

...
: All additional arguments are passed on to caret::train(), such as case weights via the weights argument or ntree for rf models. See the caret::train() docs for more details.

Value

Named list with results:

- trained_model: Output of caret::train(), including the best model.
- test_data: Part of the data that was used for testing.

- `performance`: Data frame of performance metrics. The first column is the cross-validation performance metric, and the last two columns are the ML method used and the seed (if one was set), respectively. All other columns are performance metrics calculated on the test data. This contains only one row, so you can easily combine performance data frames from multiple calls to `run_ml()` (see `vignette("parallel")`).
- `feature_importance`: If feature importances were calculated, a data frame where each row is a feature or correlated group. The columns are the performance metric of the permuted data, the difference between the true performance metric and the performance metric of the permuted data (true - permuted), the feature name, the ML method, the performance metric name, and the seed (if provided). For AUC and RMSE, the higher perf_metric_diff is, the more important that feature is for predicting the outcome. For log loss, the lower perf_metric_diff is, the more important that feature is for predicting the outcome.

More details

For more details, please see the vignettes.

Author(s)

Begüm Topçuoğlu, <topcuoglu.begum@gmail.com>

Zena Lapp, <zenalapp@umich.edu>

Kelly Sovacool, <sovacool@umich.edu>

Examples

```
## Not run:

# regression
run_ml(otu_small, "glmnet",
  seed = 2019
)

# random forest w/ feature importance
run_ml(otu_small, "rf",
  outcome_colname = "dx",
  find_feature_importance = TRUE
)

# custom cross validation & hyperparameters
run_ml(otu_mini_bin[, 2:11],
  "glmnet",
  outcome_colname = "Otu00001",
  seed = 2019,
  hyperparameters = list(lambda = c(1e-04), alpha = 0),
  cross_val = caret::trainControl(method = "none"),
  calculate_performance = FALSE
)

## End(Not run)
```

`tidy_perf_data` *Tidy the performance dataframe*

Description

Used by `plot_model_performance()`.

Usage

```
tidy_perf_data(performance_df)
```

Arguments

`performance_df` dataframe of performance results from multiple calls to `run_ml()`

Value

Tidy dataframe with model performance metrics.

Author(s)

Begüm Topçuoglu, <topcuoglu.begum@gmail.com>

Kelly Sovacool, <sovacool@umich.edu>

Examples

```
## Not run:
# call `run_ml()` multiple times with different seeds
results_lst <- lapply(seq(100, 104), function(seed) {
  run_ml(otu_small, "glmnet", seed = seed)
})
# extract and combine the performance results
perf_df <- lapply(results_lst, function(result) {
  result[["performance"]]
}) %>%
  dplyr::bind_rows()
# make it pretty!
tidy_perf_data(perf_df)

## End(Not run)
```

train_model *Train model using caret::train().*

Description

Train model using caret::train().

Usage

```
train_model(
  train_data,
  outcome_colname,
  method,
  cv,
  perf_metric_name,
  tune_grid,
  ...
)
```

Arguments

train_data
: Training data. Expected to be a subset of the full dataset.

outcome_colname
: Column name as a string of the outcome variable (default NULL; the first column will be chosen automatically).

method
: ML method. Options: c("glmnet", "rf", "rpart2", "svmRadial", "xgbTree").
  - glmnet: linear, logistic, or multiclass regression
  - rf: random forest
  - rpart2: decision tree
  - svmRadial: support vector machine
  - xgbTree: xgboost

cv
: Cross-validation caret scheme from define_cv().

perf_metric_name
: The column name from the output of the function provided to perf_metric_function that is to be used as the performance metric. Defaults: binary classification = "ROC", multi-class classification = "logLoss", regression = "RMSE".

tune_grid
: Tuning grid from get_tuning_grid().#'

...
: All additional arguments are passed on to caret::train(), such as case weights via the weights argument or ntree for rf models. See the caret::train() docs for more details.

Value

Trained model from caret::train().

Author(s)

Zena Lapp, <zenalapp@umich.edu>

Examples

```
## Not run:
training_data <- otu_mini_bin_results_glmnet$trained_model$trainingData %>%
  dplyr::rename(dx = .outcome)
method <- "rf"
hyperparameters <- get_hyperparams_list(otu_mini_bin, method)
cross_val <- define_cv(training_data,
  "dx",
  hyperparameters,
  perf_metric_function = caret::multiClassSummary,
  class_probs = TRUE,
  cv_times = 2
)
tune_grid <- get_tuning_grid(hyperparameters, method)

rf_model <- train_model(
  training_data,
  "dx",
  method,
  cross_val,
  "AUC",
  tune_grid,
  ntree = 1000
)
rf_model$results %>% dplyr::select(mtry, AUC, prAUC)

## End(Not run)
```

Index