Package ‘HMDA’

February 16, 2026

Type Package

Title Holistic Multimodel Domain Analysis for Exploratory Machine Learning

Version 0.2.0

Depends R (>= 3.5.0)

Description Holistic Multimodel Domain Analysis (HMDA) is a robust and transparent framework designed for exploratory machine learning research, aiming to enhance the process of feature assessment and selection. HMDA addresses key limitations of traditional machine learning methods by evaluating the consistency across multiple high-performing models within a fine-tuned modeling grid, thereby improving the interpretability and reliability of feature importance assessments. Specifically, it computes Weighted Mean SHapley Additive exPlanations (WMSHAP), which aggregate feature contributions from multiple models based on weighted performance metrics. HMDA also provides confidence intervals to demonstrate the stability of these feature importance estimates. This framework is particularly beneficial for analyzing complex, multidimensional datasets common in health research, supporting reliable exploration of mental health outcomes such as suicidal ideation, suicide attempts, and other psychological conditions. Additionally, HMDA includes automated procedures for feature selection based on WMSHAP ratios and performs dimension reduction analyses to identify underlying structures among features. For more details see Haghish (2025) <doi:10.13140/RG.2.2.32473.63846>.

Imports curl (>= 4.3.0), h2o (>= 3.34.0.0), shapley (>= 0.5), autoEnsemble (>= 0.3), h2otools (>= 0.4), splitTools (>= 1.0.1), psych (>= 2.4.6), dplyr (>= 1.1.4), ggplot2 (>= 3.4.2), gridExtra (>= 2.3), reshape2 (>= 1.4.5)

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

URL `http://dx.doi.org/10.13140/RG.2.2.32473.63846`, `https://github.com/haghish/HMDA`

BugReports `https://github.com/haghish/HMDA/issues`

NeedsCompilation no

Author E. F. Haghish [aut, cre, cph]

Maintainer E. F. Haghish <haghish@hotmail.com>

Repository CRAN

Date/Publication 2026-02-16 11:40:02 UTC

Contents

`best_of_family` *Select Best Models by Performance Metrics*

Description

Detects all performance metric columns in a data frame, and for each metric, identifies the best model based on whether a higher or lower value is preferred. The function returns a vector of unique model IDs corresponding to the best models across all detected metrics.

Usage

```
best_of_family(df)
```

Arguments

`df` A data frame containing model performance results. It must include a column named `"model_id"` and one or more numeric columns for performance metrics.

Details

The function first detects numeric columns (other than `"model_id"`) as performance metrics. It then uses a predefined mapping to determine the optimal direction for each metric: for example, higher values of `auc` and `aucpr` are better, while lower values of `logloss`, `mean_per_class_error`, `rmse`, and `mse` are preferred. For any metric not in the mapping, the function assumes that lower values indicate better performance.

For each metric, the function identifies the row index that produces the best value according to the corresponding direction (using `which.max()` or `which.min()`). It then extracts the `model_id` from that row. The final result is a unique set of model IDs that represent the best models across all metrics.

Value

A character vector of unique `model_id` values corresponding to the best model(s) for each detected performance metric.

Author(s)

E. F. Haghish

`check_efa` *Check Exploratory Factor Analysis Suitability*

Description

Checks whether the specified features in a data frame meet basic criteria for performing exploratory factor analysis (EFA). The function verifies that each feature exists, is numeric, has sufficient variability, and does not have an excessive proportion of missing values. For multiple features, it also evaluates whether the correlation matrix is full rank and whether each feature has at least one sufficiently strong intercorrelation with another feature.

Usage

```
check_efa(
  df,
  features,
  min_unique = 5,
  min_intercorrelation = 0.3,
  max_missing_rate = 0.05,
  verbose = FALSE
)
```

Arguments

`df` A dataframe containing the features.

`features` A character vector of feature names to be evaluated.

`min_unique` An integer specifying the minimum number of unique non-missing values required for a feature. Default is 5.

`min_intercorrelation` A numeric threshold for the minimum acceptable intercorrelation among features. (Note: this parameter is not used explicitly in the current implementation.) Default is 0.3.

`max_missing_rate` A numeric threshold for maximum missing values. Features that have a higher missing rate than this threshold will be flagged.

`verbose` Logical; if `TRUE`, a confirmation message is printed when all features appear suitable. Default is `FALSE`.

Details

The function performs several checks:

Existence Verifies that each feature in `features` is present in `df`.

Numeric Type Checks that each feature is numeric.

Variability Ensures that each feature has at least `min_unique` unique non-missing values.

Missing Values Flags features with more than 20% missing values.

If more than one feature is provided, the function computes the correlation matrix (using pairwise complete observations) and checks:

Full Rank Whether the correlation matrix is full rank. A rank lower than the number of features indicates redundancy.

Intercorrelations Identifies features that do not have any correlation (>= 0.4) with the other features.

Value

`TRUE` if all features are deemed suitable for EFA, and `FALSE` otherwise. In the latter case, messages detailing the issues are printed.

Author(s)

E. F. Haghish

Examples

```
# Example: assess feature suitability for EFA using the USJudgeRatings dataset.
# this dataset contains ratings on several aspects of U.S. federal judges' performance.
# Here, we check whether these rating variables are suitable for EFA.
data("USJudgeRatings")
features_to_check <- colnames(USJudgeRatings[,-1])
```

```
result <- check_efa(
  df = USJudgeRatings,
  features = features_to_check,
  min_unique = 3,
  verbose = TRUE
)

# TRUE indicates the features are suitable.
print(result)
```

dictionary *Dictionary of Variable Attributes*

Description

Extracts a specified attribute from each column of a data frame and returns a dictionary as a data frame mapping variable names to their corresponding attribute values.

Usage

```
dictionary(df, attribute = "label", na.rm = TRUE)
```

Arguments

`df`	A data frame whose columns may have attached attributes.
`attribute`	A character string specifying the name of the attribute to extract from each column (e.g., "label").
`na.rm`	Logical; if `TRUE`, rows for which the attribute is missing (`NA`) are omitted from the output. Default is `TRUE`.

Details

The function iterates over each column in the input data frame `df` and retrieves the specified attribute using `attr()`. If the attribute is not found for a column, `NA` is returned as its description. The resulting data frame acts as a dictionary for the variables, which is particularly useful for documenting datasets during exploratory data analysis.

Value

A data frame with two columns:

name The names of the variables in `df`.

description The extracted attribute values from each variable.

Author(s)

E. F. Haghish

Examples

```
# Example: Generate a dictionary of variable labels using the USJudgeRatings dataset.
# This dataset contains ratings on various performance measures for U.S. federal judges.
data("USJudgeRatings")

# Assume that the dataset's variables have been annotated with "label" attributes.
# which is the default label read by dictionary
attr(USJudgeRatings$CONT, "label") <- "Content Quality"
attr(USJudgeRatings$INTG, "label") <- "Integrity"
attr(USJudgeRatings$DMNR, "label") <- "Demeanor"
attr(USJudgeRatings$DILG, "label") <- "Diligence"

# Generate the dictionary of labels
dict <- dictionary(USJudgeRatings, "label")
print(dict)
```

---

hmda.adjust.params *Adjust Hyperparameter Combinations*

Description

Prunes or expands a list of hyperparameters so that the total number of model combinations, computed as the product of the lengths of each parameter vector, is near the desired target (`n_models`). It first prunes the parameter with the largest number of values until the product is less than or equal to `n_models`. Then, if the product is much lower than the target (less than half of `n_models`), it attempts to expand the parameter with the smallest number of values by adding a midpoint value (if numeric).

Usage

```
hmda.adjust.params(params, n_models)
```

Arguments

`params`	A list of hyperparameter vectors.
`n_models`	Integer. The desired target number of model combinations.

Details

The function calculates the current product of the lengths of the hyperparameter vectors. In a loop, it removes the last element from the parameter vector with the largest length until the product is less than or equal to `n_models`. If the resulting product is less than half of `n_models`, the function attempts to expand the parameter with the smallest length by computing a midpoint between the two closest numeric values. The expansion stops if no new value can be added, to avoid an infinite loop.

Value

A list of hyperparameter vectors that has been pruned or expanded so that the product of their lengths is near `n_models`.

Author(s)

E. F. Haghish

Examples

```
# Example 1: Adjust a hyperparameter grid for 100 models.
params <- list(
  alpha = c(0.1, 0.2, 0.3, 0.4),
  beta = c(1, 2, 3, 4, 5),
  gamma = c(10, 20, 30)
)
new_params <- hmda.adjust.params(params, n_models = 100)
print(new_params)

# Example 2: The generated hyperparameters range between min and max of each
# vector in the list
params <- list(
  alpha = c(0.1, 0.2),
  beta = c(1, 2, 3),
  gamma = c(10, 20)
)
new_params <- hmda.adjust.params(params, n_models = 1000)
print(new_params)
```

`hmda.autoEnsemble` *Build Stacked Ensemble Model Using autoEnsemble R package*

Description

Wrapper function in the HMDA package that builds a stacked ensemble model by combining multiple models. It leverages the **autoEnsemble** package to stack a set of trained models (e.g., from HMDA grid) into a stronger meta-learner. For more details on autoEnsemble, please see the GitHub repository at https://github.com/haghish/autoEnsemble and the CRAN package of **autoEnsemble** R package.

Usage

```
hmda.autoEnsemble(
  models,
  training_frame,
  newdata = NULL,
  family = "binary",
  strategy = c("search"),
  model_selection_criteria = c("auc", "aucpr", "mcc", "f2"),
  min_improvement = 1e-05,
  max = NULL,
  top_rank = seq(0.01, 0.99, 0.01),
  stop_rounds = 3,
  reset_stop_rounds = TRUE,
  stop_metric = "auc",
  seed = -1,
  verbatim = FALSE
)
```

Arguments

`models` A grid object (e.g., an H2O grid / HMDA grid) or a character vector of model IDs.

`training_frame` An H2OFrame (or data frame already uploaded to the H2O server) that contains the training data used to build the base models.

`newdata` Optional `H2OFrame` used for evaluating the ensemble during the stacking.

`family` Character string specifying the modeling family (e.g., `"binary"`). See the autoEnsemble package for full documentation.

`strategy` A character vector specifying the ensemble strategy. The available strategy is `"search"` (default). The `"search"` strategy searches for the best combination of top-performing diverse models.

`model_selection_criteria`
A character vector specifying the performance metrics to consider for model selection. The default is `c("auc", "aucpr", "mcc", "f2")`. Other possible criteria include `"f1point5"`, `"f3"`, `"f4"`, `"f5"`, `"kappa"`, `"mean_per_class_error"`, `"gini"`, and `"accuracy"`.

`min_improvement`
Numeric. The minimum improvement in the evaluation metric required to continue the ensemble search.

`max` Integer. The maximum number of models for each selection criterion. If `NULL`, a default value based on the top rank percentage is used.

`top_rank` Numeric vector. Specifies the percentage (or percentages) of the top models that should be considered for ensemble selection. If the strategy is `"search"`, the function searches for the best combination of models from the top to the bottom ranked; if the strategy is `"top"`, only the first value is used. Default is `seq(0.01, 0.99, 0.01)`.

`stop_rounds` Integer. The number of consecutive rounds with no improvement in the performance metric before stopping the search.

`reset_stop_rounds`
Logical. If `TRUE`, the stopping rounds counter is reset each time an improvement is observed.

`stop_metric` Character. The metric used for early stopping; the default is `"auc"`. Other options include `"aucpr"` and `"mcc"`.

seed — Integer. A random seed for reproducibility. Default is -1.

verbatim — Logical. If TRUE, the function prints additional progress information for debugging purposes.

Details

This wrapper function integrates with the HMDA package workflow to build a stacked ensemble model from a set of base models. It calls the ensemble() function from the **autoEnsemble** package to construct the ensemble. The function is designed to work within HMDA's framework, where base models are generated via grid search or AutoML. For more details on the autoEnsemble approach, see:

- GitHub: https://github.com/haghish/autoEnsemble
- CRAN: https://CRAN.R-project.org/package=autoEnsemble

The ensemble strategy "search" (default) searches for the best combination of top-performing and diverse models to improve overall performance. The wrapper returns both the final ensemble model and the list of top-ranked models used in the ensemble.

Value

A list containing:

model The ensemble model built by autoEnsemble.

top_models A data frame of the top-ranked base models that were used in building the ensemble.

Author(s)

E. F. Haghish

Examples

```
## Not run:
  library(HMDA)
  library(h2o)
  hmda.init()

  # Import a sample binary outcome dataset into H2O
  train <- h2o.importFile(
  "https://s3.amazonaws.com/h2o-public-test-data/smalldata/higgs/higgs_train_10k.csv")
  test <- h2o.importFile(
  "https://s3.amazonaws.com/h2o-public-test-data/smalldata/higgs/higgs_test_5k.csv")

  # Identify predictors and response
  y <- "response"
  x <- setdiff(names(train), y)

  # For binary classification, response should be a factor
  train[, y] <- as.factor(train[, y])
  test[, y] <- as.factor(test[, y])
  params <- list(learn_rate = c(0.01, 0.1),
                 max_depth = c(3, 5, 9),
                 sample_rate = c(0.8, 1.0)
  )

  # Train and validate a cartesian grid of GBMs
  hmda_grid1 <- hmda.grid(algorithm = "gbm", x = x, y = y,
                          grid_id = "hmda_grid1",
                          training_frame = train,
                          nfolds = 10,
                          ntrees = 100,
                          seed = 1,
                          hyper_params = params)

  # Assess the performances of the models
  grid_performance <- hmda.grid.analysis(hmda_grid1)

  # Return the best 2 models according to each metric
  hmda.best.models(grid_performance, n_models = 2)

  # build an autoEnsemble model & test it with the testing dataset
  meta <- hmda.autoEnsemble(models = hmda_grid1, training_frame = train)
  print(h2o.performance(model = meta$model, newdata = test))

## End(Not run)
```

hmda.best.models *Select Best Models Across All Models in HMDA Grid*

Description

Scans an HMDA grid analysis data frame for performance metric columns and, for each metric, selects the best-performing models according to the correct optimization direction (lower is better for some metrics; higher is better for others). The function returns a subset of the input data frame containing the union of selected model IDs.

Usage

```
hmda.best.models(
  df,
  n_models = NULL,
  distance_percentage = NULL,
 metrics = c("logloss", "mae", "mse", "rmse", "rmsle", "mean_per_class_error", "auc",
    "aucpr", "r2", "accuracy", "f1", "mcc", "f2"),
  hyperparam = FALSE
)
```

Arguments

`df` A data frame of class `"hmda.grid.analysis"` containing model performance results. It must include a column named `model_ids`.

`n_models` Integer. The number of top models to select per metric. If both `n_models` and `distance_percentage` are `NULL`, defaults to 1.

`distance_percentage`
Numeric in (0, 1). Alternative to `n_models`. Selects all models within a given percentage distance of the best value for each metric (direction-aware). You must specify either `n_models` or `distance_percentage`, not both. distance_percentage is direction-aware. For example, when metric is AUC, if the distance_percentage is set to 1 have AUC equal or lower than 99 specified that lower values mean better performance, such as logloss, then a distance_percentage of 1 the model with the lowest logloss.

`metrics` Character vector of performance metric column names to consider. Supported metrics are "logloss", "mae", "mse", "rmse", "rmsle", "mean_per_class_error", "auc", "aucpr", "r2", "accuracy", "f1", "mcc", "f2".

`hyperparam` Logical. If `TRUE`, returns all columns for the selected models (including hyperparameters). If `FALSE`, returns only `model_ids` plus the selected metric columns.

Details

The function uses a predefined set of H2O performance metrics along with their desired optimization directions:

logloss, mae, mse, rmse, rmsle, mean_per_class_error Lower values are better.

auc, aucpr, r2, accuracy, f1, mcc, f2 Higher values are better.

Value

A data frame containing the union of selected models across all considered metrics. If `hyperparam = FALSE`, the output includes `model_ids` and the metric columns found in `df`. If `hyperparam = TRUE`, the output includes all columns from `df` for the selected models.

Author(s)

E. F. Haghish

Examples

```
## Not run:
  library(HMDA)
  library(h2o)
  hmda.init()

  # Import a sample binary outcome dataset into H2O
  train <- h2o.importFile(
  "https://s3.amazonaws.com/h2o-public-test-data/smalldata/higgs/higgs_train_10k.csv")
  test <- h2o.importFile(
  "https://s3.amazonaws.com/h2o-public-test-data/smalldata/higgs/higgs_test_5k.csv")
  # Identify predictors and response
  y <- "response"
  x <- setdiff(names(train), y)

  # For binary classification, response should be a factor
  train[, y] <- as.factor(train[, y])
  test[, y] <- as.factor(test[, y])

  params <- list(learn_rate = c(0.01, 0.1),
                 max_depth = c(3, 5, 9),
                 sample_rate = c(0.8, 1.0)
  )

  # Train and validate a cartesian grid of GBMs
  hmda_grid1 <- hmda.grid(algorithm = "gbm", x = x, y = y,
                          grid_id = "hmda_grid1",
                          training_frame = train,
                          nfolds = 10,
                          ntrees = 100,
                          seed = 1,
                          hyper_params = params)

  # Assess the performances of the models
  grid_performance <- hmda.grid.analysis(hmda_grid1)

  # Return the best 2 models according to each metric
  hmda.best.models(grid_performance, n_models = 2)

 # return all models with performance metric as high as 98% of the best model, for each metric
  # i.e., the distance of the selected models should be up to 2% from the
  # best model in each metric
  hmda.best.models(grid_performance, distance_percentage = 0.02)

## End(Not run)
```

hmda.compare.shap.plot

Compare SHAP plots across selected models

Description

Produces side-by-side comparison plots of SHAP contributions for multiple models. Models can be provided explicitly via `model_id`, or selected automatically from an `hmda.grid.analysis` data frame using `hmda.best.models()` for each metric in `metrics`. Two plot styles are supported:

- `"shap"` H2O SHAP summary plot (beeswarm-style) for each model.
- `"bar"` Bar plot based on a single-model `shapley::shapley()` run.

Usage

```
hmda.compare.shap.plot(
  hmda.grid.analysis,
  newdata = NULL,
  model_id = NULL,
  metrics = c("aucpr", "mcc", "f2"),
  plot = "shap",
  top_n_features = 4,
  ylimits = c(-1, 1)
)
```

Arguments

hmda.grid.analysis
: A data frame of class "hmda.grid.analysis" containing model evaluation results and a model_ids column. Used only when model_id is NULL.

newdata
: An H2OFrame used for SHAP computation. Required for both plot types.

model_id
: Optional character vector of H2O model IDs. If provided, the function compares these models directly and ignores hmda.grid.analysis and metrics.

metrics
: Character vector of metric names used to select the best model per metric from hmda.grid.analysis via hmda.best.models(..., n_models = 1).

plot
: Character. Plot type: "shap" (default) or "bar".

top_n_features
: Integer. Number of top features shown in each plot.

ylimits
: Numeric vector of length 2 giving y-axis limits for plot = "shap". the default is c(-1, 1), which is only specified for aesthetic reasons, to make the plots comparable. Consider expanding these limits based on your data.

Details

When model_id is NULL, the function selects one model per metric from metrics using hmda.best.models(). When model_id is provided, models are labeled as "Model 1", "Model 2", etc.

Value

A gtable/grob object returned by gridExtra::grid.arrange() combining the plots. The combined plot is also printed.

Author(s)

E. F. Haghish

Examples

```
## Not run:
  library(HMDA)
  library(h2o)
  hmda.init()
  # Import a sample binary outcome dataset into H2O
  train <- h2o.importFile(
  "https://s3.amazonaws.com/h2o-public-test-data/smalldata/higgs/higgs_train_10k.csv")
  test <- h2o.importFile(
  "https://s3.amazonaws.com/h2o-public-test-data/smalldata/higgs/higgs_test_5k.csv")

  # Identify predictors and response
  y <- "response"
  x <- setdiff(names(train), y)

  # For binary classification, response should be a factor
  train[, y] <- as.factor(train[, y])
  test[, y] <- as.factor(test[, y])

  params <- list(learn_rate = c(0.01, 0.1),
                 max_depth = c(3, 5, 9),
                 sample_rate = c(0.8, 1.0)
  )

  # Train and validate a cartesian grid of GBMs
  hmda_grid1 <- hmda.grid(algorithm = "gbm", x = x, y = y,
                          grid_id = "hmda_grid1",
                          training_frame = train,
                          nfolds = 10,
                          ntrees = 100,
                          seed = 1,
                          hyper_params = params)

  # Assess the performances of the models
  grid_performance <- hmda.grid.analysis(hmda_grid1)

  # compare the best models acording to each performance metric
  hmda.compare.shap.plot(hmda.grid.analysis = grid_performance,
                         newdata = test,
                         metrics = c("aucpr", "mcc", "f2"),
                         plot = "bar",
                         top_n_features = 5)

  # Return the best 2 models according to each metric
  best_models <- hmda.best.models(grid_performance, n_models = 2)

  # compare the specified models based on their model_ids
  hmda.compare.shap.plot(hmda.grid.analysis = grid_performance,
                         model_id = best_models$model_ids[1:3],
                         newdata = test,
                         metrics = c("aucpr", "mcc", "f2"),
                         plot = "bar",
                         top_n_features = 5)


## End(Not run)
```

hmda.domain *Domain-level WMSHAP summary and plot*

Description

#' Wrapper around `shapley.domain` to compute and visualize weighted mean SHAP ratios (WMSHAP) at the domain/group/factor level. Domains are user-defined clusters of feature names (e.g., latent factors or conceptual groups). The function aggregates feature-level contributions into domain-level contributions and returns a plot and confidence intervals.

Usage

```
hmda.domain(
  wmshap,
  domains,
  plot = "bar",
  print = FALSE,
  colorcode = NULL,
  xlab = "Factors"
)
```

Arguments

`wmshap`	object of class 'shapley', as returned by the 'shapley' function
`domains`	character list, specifying the domains for grouping the features' contributions. Domains are clusters of features' names, that can be used to compute WMSHAP at higher level, along with their 95 better understand how a cluster of features influence the outcome. Note that either of 'features' or 'domains' arguments can be specified at the time.
`plot`	character, specifying the type of the plot, which can be either 'bar' or 'wmshap'. The default is 'bar'.
`print`	logical. if TRUE, the WMSHAP summary table for the given row is printed
`colorcode`	Character vector for specifying the color names for each domain in the plot.
`xlab`	Character label for WMSHAP domains or factors

Value

A ggplot object (invisibly returned) and, depending on `print`, prints the domain summary.

Author(s)

E. F. Haghish

Examples

```
## Not run:
  library(HMDA)
  library(h2o)
  hmda.init()

  # Import a sample binary outcome dataset into H2O
  train <- h2o.importFile(
  "https://s3.amazonaws.com/h2o-public-test-data/smalldata/higgs/higgs_train_10k.csv")
  test <- h2o.importFile(
  "https://s3.amazonaws.com/h2o-public-test-data/smalldata/higgs/higgs_test_5k.csv")

  # Identify predictors and response
  y <- "response"
  x <- setdiff(names(train), y)

  # For binary classification, response should be a factor
  train[, y] <- as.factor(train[, y])
  test[, y] <- as.factor(test[, y])

  params <- list(learn_rate = c(0.01, 0.1),
                 max_depth = c(3, 5, 9),
                 sample_rate = c(0.8, 1.0)
  )

  # Train and validate a cartesian grid of GBMs
  hmda_grid1 <- hmda.grid(algorithm = "gbm", x = x, y = y,
                          grid_id = "hmda_grid1",
                          training_frame = train,
                          nfolds = 10,
                          ntrees = 100,
                          seed = 1,
                          hyper_params = params)

  # compute weighted mean shap values
  wmshap <- hmda.wmshap(models = hmda_grid1,
                        newdata = test,
                        performance_metric = "aucpr",
                        standardize_performance_metric = FALSE,
                        performance_type = "xval",
                        minimum_performance = 0,
                        method = "mean",
                        cutoff = 0.01,
                        plot = TRUE)

  # define domains to combine their WMSHAP values
  # =============================================
  #
  # There are different ways to specify a cluster of features or even
  # a group of factors that touch on a broader domain. HMDA includes
  # exploratory factor analysis procedure to help with this process
  # (see ?hmda.efa function). Here, "assuming" that we have good reasons
  # to combine some of the features under some clusters:

  domains = list(Group1 = c("x22", "x18", "x14", "x1", "x10", "x4"),
                 Group2 = c("x25", "x23", "x6", "x27"),
                 Group3 = c("x28", "x26"))

  hmda.domain(wmshap = wmshap,
              plot = "bar",
              domains = domains,
              print = TRUE)

## End(Not run)
```

hmda.efa *Perform Exploratory Factor Analysis with HMDA*

Description

Performs exploratory factor analysis (EFA) on a specified set of features from a data frame using the **psych** package. The function optionally runs parallel analysis to recommend the number of factors, applies a rotation method, reverses specified features, and cleans up factor loadings by zeroing out values below a threshold. It then computes factor scores and reliability estimates, and finally returns a list containing the EFA results, cleaned loadings, reliability metrics, and factor correlations.

Usage

```
hmda.efa(
  df,
  features,
  algorithm = "minres",
  rotation = "promax",
  parallel.analysis = TRUE,
  nfactors = NULL,
  dict = dictionary(df, attribute = "label"),
  minimum_loadings = 0.3,
  exclude_features = NULL,
  ignore_binary = TRUE,
  intercorrelation = 0.3,
  reverse_features = NULL,
  plot = FALSE,
  factor_names = NULL,
  verbose = TRUE
)
```

Arguments

`df`	A data frame containing the items for EFA.
`features`	A vector of feature names (or indices) in `df` to include in the factor analysis.

algorithm Character. The factor extraction method to use. Default is "minres". Other methods supported by **psych** (e.g., "ml", "minchi") may also be used.

rotation Character. The rotation method to apply to the factor solution. Default is "promax".

parallel.analysis Logical. If TRUE, runs parallel analysis using psych::fa.parallel to recommend the number of factors. Default is TRUE.

nfactors Integer. The number of factors to extract. If NULL and parallel.analysis = TRUE, the number of factors recommended by the parallel analysis is used.

dict A data frame dictionary with at least two columns: "name" and "description". Used to replace feature names with human-readable labels. Default is dictionary(df, attribute = "label").

minimum_loadings Numeric. Any factor loading with an absolute value lower than this threshold is set to zero. Default is 0.30.

exclude_features Character vector. Features to exclude from the analysis. Default is NULL.

ignore_binary Logical. If TRUE, binary items may be ignored in the analysis. Default is TRUE.

intercorrelation Numeric. (Unused in current version) Intended to set a minimum intercorrelation threshold between items. Default is 0.3.

reverse_features A vector of feature names for which the scoring should be reversed prior to analysis. Default is NULL.

plot Logical. If TRUE, a factor diagram is plotted using psych::fa.diagram. Default is FALSE.

factor_names Character vector. Optional names to assign to the extracted factors (i.e., new column names for loadings).

verbose Logical. If TRUE, the factor loadings are printed in the console.

Details

This function first checks that the number of factors is either provided or determined via parallel analysis (if parallel.analysis is TRUE). A helper function trans() is defined to reverse and standardize item scores for features specified in reverse_features. Unwanted features can be excluded via exclude_features. The EFA is then performed using psych::fa() with the chosen extraction algorithm and rotation method. Loadings are cleaned by zeroing out values below the minimum_loadings threshold, rounded, and sorted. Factor scores are computed with psych::factor.scores() and reliability is estimated using the omega() function. Finally, factor correlations are extracted from the EFA object.

Value

A list with the following components:

parallel.analysis The output from the parallel analysis, if run.

efa The full exploratory factor analysis object returned by `psych::fa`.

efa_loadings A matrix of factor loadings after zeroing out values below the `minimum_loadings` threshold, rounded and sorted.

efa_reliability The reliability results (omega) computed from the factor scores.

factor_correlations A matrix of factor correlations, rounded to 2 decimal places.

Author(s)

E. F. Haghish

Examples

```
# Example: assess feature suitability for EFA using the USJudgeRatings dataset.
# this dataset contains ratings on several aspects of U.S. federal judges' performance.
# Here, we check whether these rating variables are suitable for EFA.
data("USJudgeRatings")
features_to_check <- colnames(USJudgeRatings[,-1])
result <- check_efa(
  df = USJudgeRatings,
  features = features_to_check,
  min_unique = 3,
  verbose = TRUE
)

# TRUE indicates the features are suitable.
print(result)
```

---

hmda.feature.selection *Feature Selection Based on Weighted SHAP Values*

Description

This function selects "important", "inessential", and "irrelevant" features based on a summary of weighted mean SHAP values obtained from a prior analysis. It uses the SHAP summary table (found in the wmshap object) to identify features that are deemed important according to a specified method and cutoff. Features with a lower confidence interval (lowerCI) below zero are labeled as "irrelevant", while the remaining features are classified as "inessential" if they do not meet the importance criteria.

Usage

```
hmda.feature.selection(
  wmshap,
  method = "mean",
  cutoff = 0.01,
  top_n_features = NULL
)
```

Arguments

`wmshap` A list object (typically returned by a weighted SHAP analysis) that must contain a data frame `summaryShaps` with at least the columns `"feature"`, `"mean"`, and `"lowerCI"`. It may also contain additional columns for alternative selection methods.

`method` Character. Specify the method for selecting important features based on their weighted mean SHAP ratios. The default is `"mean"`, which selects features whose weighted mean shap ratio (WMSHAP) exceeds the `cutoff`. The alternative is `"lowerCI"`, which selects features whose lower bound of confidence interval exceeds the `cutoff`.

`cutoff` Numeric. The threshold cutoff for the selection method. Features with a weighted SHAP value (or ratio) greater than or equal to this value are considered important. Default is `0.01`.

`top_n_features` Integer. If specified, the function selects the top `top_n_features` features (based on the sorted SHAP mean values), overriding the cutoff and method arguments. If `NULL`, all features that meet the cutoff criteria are used. Default is `NULL`.

Details

The function performs the following steps:

1. Retrieves the SHAP summary table from the `wmshap` object.
2. Sorts the summary table in descending order based on the `mean` SHAP value.
3. Identifies all features available in the summary.
4. Classifies features as **irrelevant** if their `lowerCI` value is below zero.
5. If `top_n_features` is not specified, selects **important** features as those whose value for the specified `method` column meets or exceeds the `cutoff`; the remaining features (excluding those marked as irrelevant) are classified as **inessential**.
6. If `top_n_features` is provided, the function selects the top `n` features (based on the sorted order) as important, with the rest (excluding irrelevant ones) being inessential.

Value

A list with three elements:

important A character vector of features deemed important.

inessential A character vector of features considered inessential (present in the data but not meeting the importance criteria).

irrelevant A character vector of features deemed irrelevant, defined as those with a lower confidence interval (lowerCI) below zero.

Author(s)

E. F. Haghish

Examples

```
## Not run:
library(HMDA)
library(h2o)
hmda.init()
h2o.removeAll()

# Import a sample binary outcome dataset into H2O
train <- h2o.importFile(
"https://s3.amazonaws.com/h2o-public-test-data/smalldata/higgs/higgs_train_10k.csv")
test <- h2o.importFile(
"https://s3.amazonaws.com/h2o-public-test-data/smalldata/higgs/higgs_test_5k.csv")

# Identify predictors and response
y <- "response"
x <- setdiff(names(train), y)

# For binary classification, response should be a factor
train[, y] <- as.factor(train[, y])
test[, y] <- as.factor(test[, y])

params <- list(learn_rate = c(0.01, 0.1),
               max_depth = c(3, 5, 9),
               sample_rate = c(0.8, 1.0)
)

# Train and validate a cartesian grid of GBMs
hmda_grid1 <- hmda.grid(algorithm = "gbm", x = x, y = y,
                        grid_id = "hmda_grid1",
                        training_frame = train,
                        nfolds = 10,
                        ntrees = 100,
                        seed = 1,
                        hyper_params = params)

# Assess the performances of the models
grid_performance <- hmda.grid.analysis(hmda_grid1)

# Return the best 2 models according to each metric
hmda.best.models(grid_performance, n_models = 2)

# build an autoEnsemble model & test it with the testing dataset
meta <- hmda.autoEnsemble(models = hmda_grid1, training_frame = train)
print(h2o.performance(model = meta$model, newdata = test))

# compute weighted mean shap values
wmshap <- hmda.wmshap(models = hmda_grid1,
                      newdata = test,
                      performance_metric = "aucpr",
                      standardize_performance_metric = FALSE,
                      performance_type = "xval",
                      minimum_performance = 0,
                      method = "mean",
                      cutoff = 0.01,
                      plot = TRUE)

# identify the important features
selected <- hmda.feature.selection(wmshap,
                                   method = "mean",
                                   cutoff = 0.01)
print(selected)

## End(Not run)
```

`hmda.grid` *Tune a Cartesian Hyperparameter Grid in HMDA*

Description

Runs an grid search for a single tree-based algorithm supported by HMDA (currently `"drf"` or `"gbm"`). The function validates inputs, optionally generates a grid ID, runs a Cartesian grid search, and (optionally) saves the grid for recovery and reproducibility.

Usage

```
hmda.grid(
  algorithm = c("drf", "gbm"),
  grid_id = NULL,
  x,
  y,
  training_frame = h2o.getFrame("hmda.train.hex"),
  validation_frame = NULL,
  hyper_params = list(),
  nfolds = 10,
  seed = NULL,
  keep_cross_validation_predictions = TRUE,
  recovery_dir = NULL,
  sort_by = "logloss",
  ...
)
```

Arguments

`algorithm` Character. The algorithm to tune. Supported values are "drf" (Distributed Random Forest) and "gbm" (Gradient Boosting Machine). Only one algorithm can be specified. (Case-insensitive)

`grid_id` Character. Optional identifier for the grid search. If `NULL`, an automatic grid_id is generated using the algorithm name and the current time.

`x` Vector. Predictor column names or indices.

`y` Character. The response column name or index.

`training_frame` An H2OFrame containing the training data. Default is `h2o.getFrame("hmda.train.hex")`, as defined by `hmda.partition` function.

`validation_frame`
An H2OFrame for early stopping. Default is `NULL`.

`hyper_params` List. A list of hyperparameter vectors for tuning. If you do not have a clue about how to specify the hyperparameters, consider consulting `hmda.suggest.param` and `hmda.search.param` functions, which provide suggestions based on default values or random search.

`nfolds` Integer. Number of folds for cross-validation. Default is 10.

`seed` Integer. A seed for reproducibility. Default is `NULL`.

`keep_cross_validation_predictions`
Logical. Whether to keep cross-validation predictions. Default is `TRUE`.

`recovery_dir` Character. Directory path to save the grid search output. If provided, the grid is saved using `h2o.saveGrid()`.

`sort_by` Character. Metric used to sort the grid. Default is "logloss".

`...` Additional arguments passed to `h2o.grid()`.

Details

The function executes the following steps:

1. **Input Validation:** Ensures only one algorithm is specified and verifies that the training frame is an H2OFrame.
2. **Grid ID Generation:** If no `grid_id` is provided, it creates one using the algorithm name and the current time.
3. **Grid Search Execution:** Calls `h2o.grid()` with the provided hyperparameters and cross-validation settings.
4. **Grid Saving:** If a recovery directory is specified, the grid is saved to disk using `h2o.saveGrid()`.

The output is an H2O grid object that contains all the trained models.

Value

An object of class `H2OGrid` containing the grid search results.

Author(s)

E. F. Haghish

See Also

`hmda.grid.analysis`, `hmda.best.models`

Examples

```
## Not run:
  library(HMDA)
  library(h2o)
  hmda.init()

  # Import a sample binary outcome dataset into H2O
  train <- h2o.importFile(
  "https://s3.amazonaws.com/h2o-public-test-data/smalldata/higgs/higgs_train_10k.csv")
  test <- h2o.importFile(
  "https://s3.amazonaws.com/h2o-public-test-data/smalldata/higgs/higgs_test_5k.csv")

  # Identify predictors and response
  y <- "response"
  x <- setdiff(names(train), y)

  # For binary classification, response should be a factor
  train[, y] <- as.factor(train[, y])
  test[, y] <- as.factor(test[, y])

  params <- list(learn_rate = c(0.01, 0.1),
                 max_depth = c(3, 5, 9),
                 sample_rate = c(0.8, 1.0)
  )

  # Train and validate a cartesian grid of GBMs
  hmda_grid1 <- hmda.grid(algorithm = "gbm", x = x, y = y,
                          grid_id = "hmda_grid1",
                          training_frame = train,
                          nfolds = 10,
                          ntrees = 100,
                          seed = 1,
                          hyper_params = params)

  # Assess the performances of the models
  grid_performance <- hmda.grid.analysis(hmda_grid1)

  # Return the best 2 models according to each metric
  hmda.best.models(grid_performance, n_models = 2)

## End(Not run)
```

`hmda.grid.analysis` *Analyze Hyperparameter Grid Performance*

Description

Reorders an HMDA grid based on a specified performance metric and supplements the grid's summary table with additional performance metrics extracted via cross-validation. The function returns a data frame of performance metrics for each model in the grid. This enables a detailed analysis of model performance across various metrics such as logloss, AUC, RMSE, etc.

Usage

```
hmda.grid.analysis(
  grid,
  performance_metrics = c("logloss", "mse", "rmse", "rmsle", "auc", "aucpr",
    "mean_per_class_error", "r2", "f2", "mcc", "kappa"),
  sort_by = "logloss"
)
```

Arguments

`grid` A HMDA grid object from which the performance summary will be extracted.

`performance_metrics` A character vector of additional performance metric names to be included in the analysis. Default is `c("logloss", "mse", "rmse", "rmsle", "auc", "aucpr","mean_per_class_err` `"r2")`.

`sort_by` A character string indicating the performance metric to sort the grid by. Default is `"logloss"`. For metrics such as logloss, mae, mse, rmse, and rmsle, lower values are better, while for metrics like AUC, AUCPR, and R2, higher values are preferred.

Details

The function performs the following steps:

1. **Grid Reordering:** It calls `h2o.getGrid()` to reorder the grid based on the `sort_by` metric. For metrics like "logloss", "mse", "rmse", and "rmsle", sorting is in ascending order; for others, it is in descending order.
2. **Performance Table Extraction:** The grid's summary table is converted into a data frame.
3. **Additional Metric Calculation:** For each metric specified in `performance_metrics` (other than the one used for sorting), the function initializes a column with NA values and iterates over each model in the grid (via its `model_ids`) to extract the corresponding cross-validated performance metric using functions such as `h2o.auc()`, `h2o.rmse()`, etc. For threshold-based metrics (e.g., `f2`, `mcc`, `kappa`), it retrieves performance via `h2o.performance()`.
4. **Return:** The function returns the merged data frame of performance metrics.

Value

A data frame of class `"hmda.grid.analysis"` that contains the merged performance summary table. This table includes the default metrics from the grid summary along with the additional metrics specified by `performance_metrics` (if available). The data frame is sorted according to the `sort_by` metric.

Author(s)

E. F. Haghish

Examples

```
## Not run:
  # NOTE: This example may take a long time to run on your machine

  # Initialize H2O (if not already running)
  library(HMDA)
  library(h2o)
  hmda.init()

  # Import a sample binary outcome train/test set into H2O
  train <- h2o.importFile(
  "https://s3.amazonaws.com/h2o-public-test-data/smalldata/higgs/higgs_train_10k.csv")
  test <- h2o.importFile(
  "https://s3.amazonaws.com/h2o-public-test-data/smalldata/higgs/higgs_test_5k.csv")

  # Identify predictors and response
  y <- "response"
  x <- setdiff(names(train), y)

  # For binary classification, response should be a factor
  train[, y] <- as.factor(train[, y])
  test[, y] <- as.factor(test[, y])

  # Run the hyperparameter search using DRF and GBM algorithms.
  result <- hmda.search.param(algorithm = c("gbm"),
                              x = x,
                              y = y,
                              training_frame = train,
                              max_models = 100,
                              nfolds = 10,
                              stopping_metric = "AUC",
                              stopping_rounds = 3)

  # Assess the performances of the models
  grid_performance <- hmda.grid.analysis(gbm_grid1)

  # Return the best 2 models according to each metric
  hmda.best.models(grid_performance, n_models = 2)

## End(Not run)
```

hmda.init *Initialize or Restart H2O Cluster for HMDA Analysis*

Description

Initializes or restarts an H2O cluster configured for Holistic Multimodel Domain Analysis. It sets up the cluster with specified CPU threads, memory, and connection settings. It first checks for an existing cluster, shuts it down if found, and then repeatedly attempts to establish a new connection, retrying up to 10 times if necessary.

Usage

```
hmda.init(
  cpu = -1,
  ram = NULL,
  java = NULL,
  ip = "localhost",
  port = 54321,
  verbatim = FALSE,
  restart = TRUE,
  shutdown = FALSE,
  ignore_config = TRUE,
  bind_to_localhost = FALSE,
  ...
)
```

Arguments

`cpu`	integer. The number of CPU threads to use. -1 indicates all available threads. Default is -1.
`ram`	numeric. Minimum memory (in GB) for the cluster. If NULL, all available memory is used.
`java`	character. Path to the Java JDK. If provided, sets JAVA_HOME accordingly.
`ip`	character. The IP address for the H2O server. Default is "localhost".
`port`	integer. The port for the H2O server. Default is 54321.
`verbatim`	logical. If TRUE, prints detailed cluster info. Default is FALSE.
`restart`	logical. if TRUE, the server is erased and restarted
`shutdown`	logical. if TRUE, the server is closed
`ignore_config`	logical. If TRUE, ignores any existing H2O configuration. Default is TRUE.
`bind_to_localhost`	logical. If TRUE, restricts access to the cluster to the local machine. Default is FALSE.
`...`	Additional arguments passed to h2o.init().

Details

The function sets JAVA_HOME if a Java path is provided. It checks for an existing cluster via h2o.clusterInfo(). If found, the cluster is shut down and the function waits 5 seconds. It then attempts to initialize a new cluster using h2o.init() with the specified settings. On failure, it retries every 3 seconds, up to 10 attempts. If all attempts fail, an error is thrown.

Value

An `H2OConnection` object (invisibly printed by H2O) if `shutdown = FALSE`; otherwise returns `NULL`.

Author(s)

E. F. Haghish

Examples

```
## Not run:
  # Example 1: Initialize the H2O cluster with default settings.
  library(HMDA)
  hmda.init()

  # Example 2: Initialize with specific settings such as Java path.
  conn <- hmda.init(
      cpu = 4,
      ram = 8,
      java = "/path/to/java",     #e.g., "C:/Program Files/Java/jdk1.8.0_241"
      ip = "localhost",
      port = 54321,
      verbatim = TRUE
  )

  # check the status of the h2o connection
  h2o::h2o.clusterInfo(conn) #you can use h2o functions to interact with the server

## End(Not run)
```

---

hmda.partition   *Partition Data for HMDA Analysis*

Description

Partition a data frame into training, testing, and optionally validation sets, and upload these sets to a local H2O server. If an outcome column y is provided and is a factor or character, stratified splitting is used; otherwise, a random split is performed. The proportions must sum to 1.

Usage

```
hmda.partition(
  df,
  y = NULL,
  train = 0.8,
  test = 0.2,
  validation = NULL,
  seed = 2025
)
```

Arguments

df	A data frame to partition.
y	Optional character string naming the outcome column in df. If provided and df[[y]] is factor/character, a stratified split is used.
train	A numeric value for the proportion of the training set.

`test`	A numeric value for the proportion of the testing set.
`validation`	Optional numeric. Proportion of observations assigned to the validation set. Default is `NULL`. If specified, `train` + `test` + `validation` must equal 1.
`seed`	A numeric seed for reproducibility. Default is 2025.

Details

This function uses the `splitTools` package to perform the partition. When `y` is provided and is a factor or character, a stratified split is performed to preserve class proportions. Otherwise, a basic random split is used. The partitions are then converted to H2O frames using `h2o::as.h2o()`.

Value

A named list containing the partitioned data frames and their corresponding H2O frames:

hmda.train Training set (data frame).

hmda.test Testing set (data frame).

hmda.validation Validation set (data frame), if any.

hmda.train.hex Training set as an H2O frame.

hmda.test.hex Testing set as an H2O frame.

hmda.validation.hex Validation set as an H2O frame, if applicable.

Author(s)

E. F. Haghish

Examples

```
## Not run:
  # Example: Random split (80% train, 20% test) using iris data
  data(iris)
  splits <- hmda.partition(
              df = iris,
              train = 0.8,
              test = 0.2,
              seed = 2025
            )
  train_data <- splits$hmda.train
  test_data  <- splits$hmda.test

  # Example: Stratified split (70% train, 15% test, 15% validation)
  # using iris data, stratified by Species
  splits_strat <- hmda.partition(
                     df = iris,
                     y = "Species",
                     train = 0.7,
                     test = 0.15,
                     validation = 0.15,
                     seed = 2025
```

```
                )
   train_strat <- splits_strat$hmda.train
   test_strat  <- splits_strat$hmda.test
   valid_strat <- splits_strat$hmda.validation

## End(Not run)
```

`hmda.plot` *Plot WMSHAP contributions*

Description

This function applies different criteria to visualize WMSHAP contributions

Usage

```
hmda.plot(
  wmshap,
  plot = "bar",
  method = "mean",
  cutoff = 0.01,
  top_n_features = NULL,
  features = NULL,
  legendstyle = "continuous",
  scale_colour_gradient = NULL,
  labels = NULL
)
```

Arguments

`wmshap` object of class 'shapley', as returned by the 'shapley' function or hmda.wmshap function

`plot` Character. Plot type passed to `shapley.plot()`. Common options are `"bar"` and `"wmshap"`.

`method` Character. The column name in `summaryShaps` used for feature selection. Default is `"mean"`, which selects important features which have weighted mean shap ratio (WMSHAP) higher than the specified cutoff. Other alternative is "lowerCI", which selects features which their lower bound of confidence interval is higher than the cutoff.

`cutoff` numeric, specifying the cutoff for the method used for selecting the top features.

`top_n_features` Integer. If specified, the top n features with the highest weighted SHAP values will be selected, overrullung the 'cutoff' and 'method' arguments.

`features` character vector, specifying the feature to be plotted.

`legendstyle` character, specifying the style of the plot legend, which can be either 'continuous' (default) or 'discrete'. the continuous legend is only applicable to 'wmshap' plots and other plots only use 'discrete' legend.

`scale_colour_gradient`
character vector for specifying the color gradients for the plot.

`labels` Optional named character vector mapping feature names to display labels, e.g., `c(feature1 = "Label 1", feature2 = "Label 2")`.

Value

ggplot object

Author(s)

E. F. Haghish

Examples

```
## Not run:
  library(HMDA)
  library(h2o)
  hmda.init()

  # Import a sample binary outcome dataset into H2O
  train <- h2o.importFile(
  "https://s3.amazonaws.com/h2o-public-test-data/smalldata/higgs/higgs_train_10k.csv")
  test <- h2o.importFile(
  "https://s3.amazonaws.com/h2o-public-test-data/smalldata/higgs/higgs_test_5k.csv")

  # Identify predictors and response
  y <- "response"
  x <- setdiff(names(train), y)

  # For binary classification, response should be a factor
  train[, y] <- as.factor(train[, y])
  test[, y] <- as.factor(test[, y])

  params <- list(learn_rate = c(0.01, 0.1),
                 max_depth = c(3, 5, 9),
                 sample_rate = c(0.8, 1.0)
  )

  # Train and validate a cartesian grid of GBMs
  hmda_grid1 <- hmda.grid(algorithm = "gbm", x = x, y = y,
                          grid_id = "hmda_grid1",
                          training_frame = train,
                          nfolds = 10,
                          ntrees = 100,
                          seed = 1,
                          hyper_params = params)

  # compute weighted mean shap values
  wmshap <- hmda.wmshap(models = hmda_grid1,
                        newdata = test,
                        performance_metric = "aucpr",
                          standardize_performance_metric = FALSE,
                          performance_type = "xval",
                          minimum_performance = 0,
                          method = "mean",
                          cutoff = 0.01,
                          plot = TRUE)

########################################################
### PLOT THE WEIGHTED MEAN SHAP VALUES
########################################################
hmda.plot(wmshap, plot = "bar")

## End(Not run)
```

hmda.plot.metrics *Plot model performance metrics across a grid of models*

Description

Creates a line plot comparing multiple (maximize-type) performance metrics across a set of models. The input data frame is typically the output of hmda.grid.analysis() and must contain a model_ids column and one or more numeric metric columns (e.g., aucpr, mcc, f2).

The function can either plot the first top_models rows (criteria = "top_models") or include all models that achieve at least distance_percentage times the best value for at least one metric (criteria = "distance_percentage").

Usage

```
hmda.plot.metrics(
  df,
  metrics = c("auc", "aucpr", "r2", "mcc", "f2"),
  criteria = "distance_percentage",
  top_models = 100,
  distance_percentage = 0.95,
  plot = TRUE,
  title = NULL
)
```

Arguments

df A data frame of class "hmda.grid.analysis" containing a column model_ids and numeric metric columns.

metrics Character vector of column names in df to be plotted.

criteria Character. One of "top_models" or "distance_percentage" (default).

top_models Integer. Number of top rows to plot when criteria = "top_models".

`distance_percentage`
: Numeric in (0, 1]. When `criteria` = `"distance_percentage"`, includes models with metric values $\geq$ best(metric) * distance_percentage for at least one metric.

`plot`
: Logical. If `TRUE`, prints the plot.

`title`
: Character. Add title to the plot.

Author(s)

E. F. Haghish

Examples

```
## Not run: 
  # Example: Create a hyperparameter grid for GBM models.
  predictors <- c("var1", "var2", "var3")
  response <- "target"

  # Define hyperparameter ranges
  hyper_params <- list(
    ntrees = seq(50, 150, by = 25),
    max_depth = c(5, 10, 15),
    learn_rate = c(0.01, 0.05, 0.1),
    sample_rate = c(0.8, 1.0),
    col_sample_rate = c(0.8, 1.0)
  )

  # Run the grid search
  grid <- hmda.grid(
    algorithm = "gbm",
    x = predictors,
    y = response,
    training_frame = h2o.getFrame("hmda.train.hex"),
    hyper_params = hyper_params,
    nfolds = 10,
    stopping_metric = "AUTO"
  )

  # Assess the performances of the models
  grid_performance <- hmda.grid.analysis(grid)

  # plot the metrics of models that are within 95% of the best models
  # for each of the specified metrics
  hmda.plot.metrics(grid_performance,
                    criteria = "distance_percentage",
                    distance_percentage = 0.95,
                    metrics = c("auc", "aucpr", "r2", "mcc", "f2"))


## End(Not run)
```

hmda.search.param *Search for Hyperparameters via Random Search*

Description

Runs an automated hyperparameter search and returns several summaries of the hyperparameter grids as well as detailed hyperparameters from each model, and then produces multiple summaries based on different strategies. These strategies include:

Best of Family Selects the best model for each performance metric (avoiding duplicate model IDs).

Top 2 Extracts hyperparameter settings from the top 2 models (according to a specified ranking metric).

Top 5 Extracts hyperparameter settings from the top 5 models.

Top 10 Extracts hyperparameter settings from the top 10 models.

These summaries help in identifying candidate hyperparameter ranges for further manual tuning. Note that a good suggestion depends on the extent of random search you carry out.

Usage

```
hmda.search.param(
  algorithm = c("drf", "gbm"),
  sort_by = "logloss",
  x,
  y,
  training_frame = h2o.getFrame("hmda.train.hex"),
  validation_frame = NULL,
  max_models = 100,
  max_runtime_secs = 3600,
  nfolds = 10,
  seed = NULL,
  fold_column = NULL,
  weights_column = NULL,
  keep_cross_validation_predictions = TRUE,
  stopping_rounds = NULL,
  stopping_metric = "AUTO",
  stopping_tolerance = NULL,
  ...
)
```

Arguments

`algorithm` Character. The algorithm to include in the random search. Supported values include "drf" (Distributed Random Forest) and "gbm" (Gradient Boosting Machine). The input is case-insensitive.

`sort_by` Character string specifying the metric used to rank models.

`x` Vector of predictor column names or indices.

`y` Character string specifying the response column.

`training_frame` An H2OFrame containing the training data. Default is `h2o.getFrame("hmda.train.hex")`.

`validation_frame`
An H2OFrame for early stopping. Default is `NULL`.

`max_models` Integer. Maximum number of models to build. Default is 100.

`max_runtime_secs`
integer. Amount of time (in seconds) that the model should keep searching. Default is 3600.

`nfolds` Integer. Number of folds for cross-validation. Default is 10.

`seed` Integer. A seed for reproducibility. Default is `NULL`.

`fold_column` Character. Column name for cross-validation fold assignment. Default is `NULL`.

`weights_column` Character. Column name for observation weights. Default is `NULL`.

`keep_cross_validation_predictions`
Logical. Whether to keep cross-validation predictions. Default is `TRUE`.

`stopping_rounds`
Integer. Number of rounds with no improvement before early stopping. Default is `NULL`.

`stopping_metric`
Character. Metric to use for early stopping. Default is "AUTO".

`stopping_tolerance`
Numeric. Relative tolerance for early stopping. Default is `NULL`.

`...` Additional arguments passed to `h2o.automl()`.

Details

The function executes an automated hyperparameter search for the specified algorithm. It then extracts the leaderboard from the H2OAutoML object and retrieves detailed hyperparameter information for each model using `automlModelParam()` from the h2otools package. The leaderboard and hyperparameter data are merged by the `model_id` column. Sorting of the merged results is performed based on the `sort_by` metric. For metrics not in `"logloss"`, `"mean_per_class_error"`, `"rmse"`, `"mse"`, lower values are considered better; for these four metrics, higher values are preferred.

After sorting, the function applies three strategies to summarize the hyperparameter search:

1. **Best of Family**: Selects the best model for each performance metric, ensuring that no model ID appears more than once.
2. **Top 2**: Gathers hyperparameter settings from the top 2 models.
3. **Top 5 and Top 10**: Similarly, collects hyperparameter settings from the top 5 and top 10 models, respectively.
4. **All**: List all the hyperparameters that were tried

These strategies provide different levels of granularity for analyzing the hyperparameter space and can be used for prototyping and further manual tuning.

Value

A list with the following components:

grid The H2OAutoML object returned by random search

leaderboard A merged data frame that combines leaderboard performance metrics with hyperparameter settings for each model. The data frame is sorted based on the specified ranking metric.

hyperparameters_best_of_family A summary list of the best hyperparameter settings for each performance metric. This strategy selects the best model per metric while avoiding duplicate model IDs.

hyperparameters_top2 A list of hyperparameter settings from the top 2 models as ranked by the chosen metric.

hyperparameters_top5 A list of hyperparameter settings from the top 5 models.

hyperparameters_top10 A list of hyperparameter settings from the top 10 models.

hyperparameters_all A list of all hyperparameter settings.

Examples

```
## Not run:
  # NOTE: This example may take a long time to run on your machine

  # Initialize H2O (if not already running)
  library(HMDA)
  library(h2o)
  hmda.init()

  # Import a sample binary outcome train/test set into H2O
  train <- h2o.importFile(
  "https://s3.amazonaws.com/h2o-public-test-data/smalldata/higgs/higgs_train_10k.csv")
  test <- h2o.importFile(
  "https://s3.amazonaws.com/h2o-public-test-data/smalldata/higgs/higgs_test_5k.csv")

  # Identify predictors and response
  y <- "response"
  x <- setdiff(names(train), y)

  # For binary classification, response should be a factor
  train[, y] <- as.factor(train[, y])
  test[, y] <- as.factor(test[, y])

  # Run the hyperparameter search using DRF and GBM algorithms.
  result <- hmda.search.param(algorithm = c("gbm"),
                              x = x,
                              y = y,
                              training_frame = train,
                              max_models = 100,
                              nfolds = 10,
                              stopping_metric = "AUC",
                              stopping_rounds = 3)
  # Access the hyperparameter list of the best_of_family strategy:
  result$best_of_family

  # Access the hyperparameter of the top5 models based on the specified ranking parameter
  result$top5

## End(Not run)
```

hmda.suggest.param *Suggest Hyperparameters for tuning HMDA Grids*

Description

Suggests candidate hyperparameter values for tree-based algorithms. It computes a hyperparameter grid whose total number of model combinations is near a specified target.

The function builds a default candidate grid for "gbm" or "drf", and then prunes or expands the grid using hmda.adjust.params() so the total number of model combinations is near n_models.

For GBM models, default candidates include max_depth, ntrees, learn_rate, sample_rate, and col_sample_rate. For DRF models, if x (predictors) and family ("regression" or "classification") are provided, a vector of mtries is also suggested via suggest_mtries().

Usage

```
hmda.suggest.param(algorithm, n_models, x = NULL, family = NULL)
```

Arguments

algorithm A character string specifying the algorithm, which can be either "gbm" (gradient boosting machines) or "drf" (distributed random forest).

n_models An integer for the desired approximate number of model combinations in the grid. Must be at least 100.

x (Optional) A vector of predictor names. If provided and its length is at least 20, it is used to compute mtries for DRF.

family (Optional) A character string indicating the modeling family. Must be either "classification" or "regression". This is used with x to suggest mtries.

Details

The function first checks that n_models is at least 100, then validates the family parameter if provided. The algorithm name is normalized to lowercase and must be either "gbm" or "drf". For "gbm", a default grid of hyperparameters is defined. For "drf", if both x and family are provided, the function computes mtries via suggest_mtries(). If not, a default grid is set without mtries. Finally, the candidate grid is pruned or expanded using hmda.adjust.params() so that the total number of combinations is near n_models.

Value

A named list of hyperparameter value vectors. This list is suitable for use with HMDA and H2O grid search functions.

Examples

```
## Not run:
  library(h2o)
  h2o.init()

  # Example 1: Suggest hyperparameters for GBM with about 120 models.
  params_gbm <- hmda.suggest.param("gbm", n_models = 120)
  print(params_gbm)

  # Example 2: Suggest hyperparameters for DRF (classification) with
  # 100 predictors.
  params_drf <- hmda.suggest.param(
                  algorithm = "drf",
                  n_models = 150,
                  x = paste0("V", 1:100),
                  family = "classification"
                )
  print(params_drf)

## End(Not run)
```

---

hmda.test *Normalize a vector based on specified minimum and maximum values*

Description

This function normalizes a vector based on specified minimum and maximum values. If the minimum and maximum values are not specified, the function will use the minimum and maximum values of the vector.

Usage

```
hmda.test(wmshap, features = NULL, domains = NULL, n = 5000)
```

Arguments

wmshap	object of class 'shapley', as returned by the 'shapley' function
features	character, name of two features to be compared with permutation test
domains	list of two character vectors, each including a number of features.
n	integer, number of permutations

Value

normalized numeric vector

Author(s)

E. F. Haghish

Examples

```
## Not run:
# load the required libraries for building the base-learners and the ensemble models
library(h2o)            #shapley supports h2o models
library(autoEnsemble)  #autoEnsemble models, particularly useful under severe class imbalance
library(shapley)

# initiate the h2o server
h2o.init(ignore_config = TRUE, nthreads = 2, bind_to_localhost = FALSE, insecure = TRUE)

# upload data to h2o cloud
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.importFile(path = prostate_path, header = TRUE)

### H2O provides 2 types of grid search for tuning the models, which are
### AutoML and Grid. Below, I demonstrate how weighted mean shapley values
### can be computed for both types.

set.seed(10)

#######################################################
### PREPARE AutoML Grid (takes a couple of minutes)
#######################################################
# run AutoML to tune various models (GBM) for 60 seconds
y <- "CAPSULE"
prostate[,y] <- as.factor(prostate[,y])  #convert to factor for classification
aml <- h2o.automl(y = y, training_frame = prostate, max_runtime_secs = 120,
                 include_algos=c("GBM"),

              # this setting ensures the models are comparable for building a meta learner
                 seed = 2023, nfolds = 10,
                 keep_cross_validation_predictions = TRUE)

### call 'shapley' function to compute the weighted mean and weighted confidence intervals
### of SHAP values across all trained models.
### Note that the 'newdata' should be the testing dataset!
result <- shapley(models = aml, newdata = prostate, plot = TRUE)

#######################################################
### Significance testing of contributions of two features
#######################################################
# testing the WMSHAP contributions of two features
hmda.test(result, features = c("GLEASON", "PSA"), n = 5000)
```

```
# testing the WMSHAP contributions of two domains (groups of features, latent factors, etc.)
hmda.test(result,
          n = 5000,
          domains = list(Demographic = c("RACE", "AGE"),
                         Cancer = c("VOL", "PSA", "GLEASON")))

## End(Not run)
```

hmda.wmshap *Compute Weighted Mean SHAP Values and Confidence Intervals via shapley algorithm*

Description

Wrapper for `shapley` that computes weighted mean SHAP ratios (WMSHAP) and (when available) confidence intervals across a set of H2O models. Model contributions are aggregated across multiple models and weighted by a chosen performance metric (e.g., `"r2"` for regression; `"auc"`, `"aucpr"`, or `"f2"` for classification). This approach considers the variability of feature importance across multiple models rather than reporting SHAP values from a single model. For more details about shapley algotithm, see `https://github.com/haghish/shapley`

Usage

```
hmda.wmshap(
  models,
  newdata,
  plot = TRUE,
  performance_metric = "r2",
  standardize_performance_metric = FALSE,
  performance_type = "xval",
  minimum_performance = 0,
  method = "lowerCI",
  cutoff = 0.01,
  top_n_features = NULL,
  n_models = 10,
  sample_size = NULL
)
```

Arguments

`models` A grid object, an AutoML grid, an autoEnsemble object, or a character vector of H2O model IDs from which the SHAP values will be computed.

`newdata` An H2OFrame (or data frame already uploaded to the H2O server) on which the SHAP values will be evaluated.

`plot` Logical. If `TRUE`, a plot of the weighted mean SHAP values along with their confidence intervals is generated. Default is `TRUE`.

`performance_metric`
: Character. Specifies the performance metric to be used as weights for the SHAP values. The default is `"r2"`. For binary classification, alternatives include `"aucpr"`, `"auc"`, and `"f2"`.

`standardize_performance_metric`
: Logical. If `TRUE`, the performance metric (used as the weights vector) is standardized so that the sum of the weights equals the length of the vector. Default is `FALSE`.

`performance_type`
: Character. Specifies whether the performance metric should be retrieved from the training data ("train"), validation data ("valid"), or cross-validation ("xval"). Default is `"xval"`.

`minimum_performance`
: Numeric. The minimum performance threshold; any model with a performance equal to or lower than this threshold will have a weight of zero in the weighted SHAP calculation. Default is `0`.

`method`
: Character. Specify the method for selecting important features based on their weighted mean SHAP ratios. The default is `"lowerCI"`, which selects features whose lower bound of confidence interval exceeds the `cutoff`. The alternative is `"mean"`, which selects features whose weighted mean shap ratio (WMSHAP) exceeds the `cutoff`.

`cutoff`
: Numeric. The cutoff value used in the feature selection method (default is `0.01`).

`top_n_features`
: Integer. If specified, only the top `n` features with the highest weighted SHAP values will be selected, overriding the cutoff and method. Default is `NULL`, which means all features are considered.

`n_models`
: Integer. The minimum number of models that must meet the `minimum_performance` criterion in order to compute the weighted mean and confidence intervals of SHAP values. Set to 1 if a global summary for a single model is desired. The default is `10`.

`sample_size`
: Integer. The number of rows in `newdata` to use for the SHAP evaluation. By default, all rows of `newdata` are used.

Details

This function is designed as a wrapper for the HMDA package and calls the `shapley()` function from the **shapley** package. It computes the weighted average of SHAP values across multiple models, using a specified performance metric (e.g., R Squared, AUC, etc.) as the weight. The performance metric can be standardized if required. Additionally, the function selects top features based on different methods (e.g., `"mean"` or `"lowerCI"`) and can limit the number of features considered via `top_n_features`. The `n_models` parameter controls how many models must meet a minimum performance threshold to be included in the weighted SHAP calculation.

For more information on the shapley and WMSHAP approaches used in HMDA, please refer to the shapley package documentation and the following resources:

- shapley GitHub: `https://github.com/haghish/shapley`
- shapley CRAN: `https://CRAN.R-project.org/package=shapley`

Value

A list with the following components:

A shapley object that includes the following compnents

plot A ggplot2 object showing the weighted mean SHAP values and confidence intervals (if `plot = TRUE`).

shap_values A data frame of the weighted mean SHAP (WMSHAP) values and confidence intervals for each feature.

performance A data frame of performance metrics for all models used in the analysis.

model_ids A vector of model IDs corresponding to the models evaluated.

Author(s)

E. F. Haghish

Examples

```
## Not run:
  library(HMDA)
  library(h2o)
  hmda.init()

  # Import a sample binary outcome dataset into H2O
  train <- h2o.importFile(
  "https://s3.amazonaws.com/h2o-public-test-data/smalldata/higgs/higgs_train_10k.csv")
  test <- h2o.importFile(
  "https://s3.amazonaws.com/h2o-public-test-data/smalldata/higgs/higgs_test_5k.csv")

  # Identify predictors and response
  y <- "response"
  x <- setdiff(names(train), y)

  # For binary classification, response should be a factor
  train[, y] <- as.factor(train[, y])
  test[, y] <- as.factor(test[, y])

  params <- list(learn_rate = c(0.01, 0.1),
                 max_depth = c(3, 5, 9),
                 sample_rate = c(0.8, 1.0)
  )

  # Train and validate a cartesian grid of GBMs
  hmda_grid1 <- hmda.grid(algorithm = "gbm", x = x, y = y,
                          grid_id = "hmda_grid1",
                          training_frame = train,
                          nfolds = 10,
                          ntrees = 100,
                          seed = 1,
                          hyper_params = gbm_params1)

  # Assess the performances of the models
  grid_performance <- hmda.grid.analysis(hmda_grid1)

  # Return the best 2 models according to each metric
  hmda.best.models(grid_performance, n_models = 2)

  # build an autoEnsemble model & test it with the testing dataset
  meta <- hmda.autoEnsemble(models = hmda_grid1, training_frame = train)
  print(h2o.performance(model = meta$model, newdata = test))

  # compute weighted mean shap values
  wmshap <- hmda.wmshap(models = hmda_grid1,
                        newdata = test,
                        performance_metric = "aucpr",
                        standardize_performance_metric = FALSE,
                        performance_type = "xval",
                        minimum_performance = 0,
                        method = "mean",
                        cutoff = 0.01,
                        plot = TRUE)

  # identify the important features
  selected <- hmda.feature.selection(wmshap,
                                     method = c("mean"),
                                     cutoff = 0.01)
  print(selected)

  # View the plot of weighted mean SHAP values and confidence intervals
  print(wmshap$plot)


## End(Not run)
```

hmda.wmshap.table *Create SHAP Summary Table Based on the Given Criterion*

Description

Generates a summary table of weighted mean SHAP (WMSHAP) values and confidence intervals for each feature based on a weighted SHAP analysis. The function filters the SHAP summary table (from a wmshap object) by selecting features that meet or exceed a specified cutoff using a selection method (default "mean"). It then sorts the table by the mean SHAP value, formats the SHAP values along with their 95% confidence intervals into a single string, and optionally adds human-readable feature descriptions from a provided dictionary. The output is returned as a markdown table using the **pander** package, or as a data frame if requested.

Usage

```
hmda.wmshap.table(
  wmshap,
  method = "lowerCI",
  cutoff = 0.01,
  round = 3,
  exclude_features = NULL,
  dict = NULL,
  markdown.table = TRUE,
  split.tables = 120,
  split.cells = 50
)
```

Arguments

`wmshap` A wmshap object, returned by the hmda.wmshap function containing a data frame `summaryShaps`.

`method` #' @param method Character. Specify the method for selecting important features based on their weighted mean SHAP ratios. The default is `"lowerCI"`, which selects features whose lower bound of confidence interval exceeds the `cutoff`. The alternative is `"mean"`, which selects features whose weighted mean shap ratio (WMSHAP) exceeds the `cutoff`.

`cutoff` Numeric. The threshold cutoff for the selection method; only features with a value in the `method` column greater than or equal to this value are retained. Default is `0.01`.

`round` Integer. The number of decimal places to round the SHAP mean and confidence interval values. Default is `3`.

`exclude_features`

Character vector. A vector of feature names to be excluded from the summary table. Default is `NULL`.

`dict` A data frame containing at least two columns named `"name"` and `"description"`. If provided, the function uses this dictionary to add human-readable feature descriptions. Default is `NULL`. If you define `"label"` attribute for each feature in the dataset, the `dictionary` function can be used to extract these attributes into a dataset.

`markdown.table` Logical. If `TRUE`, the output is formatted as a markdown table using the **pander** package; otherwise, a data frame is returned. Default is `TRUE`.

`split.tables` Integer. Controls table splitting in `pander()`. Default is `120`.

`split.cells` Integer. Controls cell splitting in `pander()`. Default is `50`.

Value

If `markdown.table = TRUE`, returns a markdown table (invisibly) showing two columns: `"Description"` and `"WMSHAP"`. If `markdown.table = FALSE`, returns a data frame with these columns.

Author(s)

E. F. Haghish

See Also

dictionary

Examples

```
## Not run:
  library(HMDA)
  library(h2o)
  hmda.init()

  # Import a sample binary outcome dataset into H2O
  train <- h2o.importFile(
  "https://s3.amazonaws.com/h2o-public-test-data/smalldata/higgs/higgs_train_10k.csv")
  test <- h2o.importFile(
  "https://s3.amazonaws.com/h2o-public-test-data/smalldata/higgs/higgs_test_5k.csv")

  # Identify predictors and response
  y <- "response"
  x <- setdiff(names(train), y)

  # For binary classification, response should be a factor
  train[, y] <- as.factor(train[, y])
  test[, y] <- as.factor(test[, y])

  params <- list(learn_rate = c(0.01, 0.1),
                 max_depth = c(3, 5, 9),
                 sample_rate = c(0.8, 1.0)
  )

  # Train and validate a cartesian grid of GBMs
  hmda_grid1 <- hmda.grid(algorithm = "gbm", x = x, y = y,
                          grid_id = "hmda_grid1",
                          training_frame = train,
                          nfolds = 10,
                          ntrees = 100,
                          seed = 1,
                          hyper_params = params)

  # Assess the performances of the models
  grid_performance <- hmda.grid.analysis(hmda_grid1)

  # Return the best 10 models according to each metric
  hmda.best.models(grid_performance, n_models = 10)


  # compute weighted mean shap values for all models
  #Note: you may compute WMSHAP for a selected well-performing models
  wmshap <- hmda.wmshap(models = hmda_grid1,
                        newdata = test,
                        performance_metric = "aucpr",
                        performance_type = "xval",
                        minimum_performance = 0,
                        method = "mean",
                        cutoff = 0.01,
                        plot = TRUE)

  # identify the important features
  selected <- hmda.feature.selection(wmshap,
                                     method = c("mean"),
                                     cutoff = 0.01)
  print(selected)

  # View the plot of weighted mean SHAP values and confidence intervals
  print(wmshap$plot)

  # get the wmshap table output in Markdown format:
  md_table <- hmda.wmshap.table(wmshap = wmshap,
                                method = "mean",
                                cutoff = 0.01,
                                round = 3,
                                markdown.table = TRUE)
  head(md_table)

## End(Not run)
```

suggest_mtries *Suggest Alternative mtries Values*

Description

Provides a set of candidate values for the mtries parameter used in Random Forest models. The suggestions are computed based on the number of predictors (p) and the modeling family. For classification, the common default is sqrt(p), while for regression it is typically p/3. For family, alternative candidates are offered to aid model tuning.

Usage

```
suggest_mtries(p, family = c("classification", "regression"))
```

Arguments

p	Integer. The number of features (predictors) in the dataset. This value is used to compute candidate mtries.
family	Character. Must be either "classification" or "regression". This determines the set of candidate values.

Details

For classification, the default is often sqrt(p); alternative suggestions include log2(p) and p^(1/3). For regression, the typical default is p/3, but candidates such as p/2 or p/5 may also be useful. The best choice depends on the data structure and predictor correlations.

Value

An integer vector of candidate values for `mtries`.

Author(s)

E. F. Haghish

Examples

```
## Not run:
  # For a classification task with 100 predictors:
  suggest_mtries(p = 100, family = "classification")

  # For a regression task with 100 predictors:
  suggest_mtries(p = 100, family = "regression")

## End(Not run)
```

Index