Package 'TidyML'

May 23, 2025

Title Machine Learning Modelling For Everyone

Version 0.0.0.9000

Description TidyML is a minimalistic library specifically designed to make the estimation of Machine Learning (ML) techniques as easy and accessible as possible, particularly within the framework of the Knowledge Discovery in Databases (KDD) process in data mining. The package provides all the essential tools needed to efficiently structure and execute each stage of a predictive or classification modeling workflow, aligning closely with the fundamental steps of the KDD methodology, from data selection and preparation, through model building and tuning, to the interpretation and evaluation of results using Sensitivity Analysis. The TidyML workflow is organized into five core steps; preprocessing(), build_model(), fine_tuning(), show_results(), and sensitivity_analysis(). These steps correspond, respectively, to data preparation and transformation, model construction, hyperparameter optimization, resuresults presentation, and sensitivity analysis. By streamlining these phases, TidyML aims to simplify the implementation of ML techniques, allowing analysts and data scientists to focus on extracting actionable insights and meaningful patterns from large datasets, in line with the objectives of the KDD process.

License ``GPL-3"

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Depends R (>= 2.10)

Imports R6,
tidyr,
tidyselect,
magrittr,
dials,
parsnip,
recipes,
rsample,
tune,
workflows,
yardstick,
vip,
glue,
ggpubr,
innsight,
torch,
shapr,
DiagrammeR,

ggbeeswarm,
ggplot2,
stringr,
sensitivity,
brulee,
ranger,
kernlab,
xgboost,
dplyr,
rlang,
tibble,
tidyverse

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

URL https://github.com/JMartinezGarcia/TidyML

BugReports https://github.com/JMartinezGarcia/TidyML/issues

LazyData true

Contents

`build_model` *02 Create ML Model*

Description

The function **build_model()** is designed to construct and attach a ML model to an existing analysis object,which contains the preprocessed dataset generated in the previous step using the preprocessing() function. Based on the specified model type and optional hyperparameters, it supports several popular algorithms—including **Neural Network**, **XGBOOST**, **Random Forest**, and **SVM** (James et al., 2021)— by initializing the corresponding hyperparameter class, updating the analysis object with these settings, and invoking the appropriate model creation function. For SVM models, it further distinguishes between kernel types (rbf, polynomial, linear) to ensure the correct implementation. The function also updates the analysis object with the model name, the fitted model, and the current processing stage before returning the enriched object, thereby streamlining the workflow for subsequent training, evaluation, or prediction steps. This modular approach facilitates flexible and reproducible ML pipelines by encapsulating both the model and its configuration within a single structured object.

Usage

```
build_model(analysis_object, model_name, hyperparameters = NULL)
```

Arguments

`analysis_object`	analysis_object created from preprocessing function.
`model_name`	Name of the ML Model. A string of the model name: "Neural Network", "Random Forest", "SVM" or "XGBOOST".
`hyperparameters`	Hyperparameters of the ML model. List containing the name of the hyperparameter and its value or range of values.

Value

An updated analysis_object containing the fitted machine learning model, the model name, the specified hyperparameters, and the current processing stage. This enriched object retains all previously stored information from the preprocessing step and incorporates the results of the model-building process, ensuring a coherent and reproducible workflow for subsequent training, evaluation, or prediction tasks.

Hyperparameters

Neural Network:

Parsnip model using **brulee** engine. Hyperparameters:

- **hidden_units**: Number of Hidden Neurons. A single value, a vector with range values `c(min_val, max_val)` or NULL for default range c(5, 20).
- **activation**: Activation Function. A vector with any of ("relu", "sigmoid", "tanh") or NULL for default values c("relu", "sigmoid", "tanh").
- **learn_rate**: Learning Rate. A single value, a vector with range values `c(min_val, max_val)` or NULL for default range c(-3, -1) in log10 scale.

Random Forest:

Parsnip model using **ranger** engine. Hyperparameters:

- **trees**: Number of Trees. A single value, a vector with range values `c(min_val, max_val)`. Default range c(100, 300).
- **mtry**: Number of variables randomly selected as candidates at each split. A single value, a vector with range values `c(min_val, max_val)` or NULL for default range c(3, 8).
- **min_n**: Minimum Number of samples to split at each node. A single value, a vector with range values `c(min_val, max_val)` or NULL for default range c(2, 25).

XGBOOST:

Parsnip model using **xgboost** engine. Hyperparameters:

- **trees**: Number of Trees. A single value, a vector with range values `c(min_val, max_val)` or NULL for default range c(100, 300).
- **mtry**: Number of variables randomly selected as candidates at each split. A single value, a vector with range values `c(min_val, max_val)` or NULL for default range c(3, 8).
- **min_n**: Minimum Number of samples to split at each node. A single value, a vector with range values `c(min_val, max_val)` or NULL for default range c(5, 25).
- **tree_depth**: Maximum tree depth. A single value, a vector with range values `c(min_val, max_val)` or NULL for default range c(3, 10).
- **learn_rate**: Learning Rate. A single value, a vector with range values `c(min_val, max_val)` or NULL for default range c(-4, -1) in log10 scale.

- **loss_reduction**: Minimum loss reduction required to make a further partition on a leaf node. A single value, a vector with range values `c(min_val, max_val)` or NULL for default range c(-5, 1.5) in log10 scale.

SVM:

Parsnip model using **kernlab** engine. Hyperparameters:

- **cost**: Penalty parameter that regulates model complexity and misclassification tolerance. A single value, a vector with range values `c(min_val, max_val)` or NULL for default range c(-3, 3) in log10 scale.
- **margin**: Distance between the separating hyperplane and the nearest data points. A single value, a vector with range values `c(min_val, max_val)` or NULL for default range c(0, 0.2).
- **type**: Kernel to be used. A single value from ("linear", "rbf", "polynomial")
- **rbf_sigma**: A single value, a vector with range values `c(min_val, max_val)` or NULL for default range c(-5, 0) in log10 scale.
- **degree**: Polynomial Degree (polynomial kernel only). A single value, a vector with range values `c(min_val, max_val)` or NULL for default range c(1, 3).
- **scale_factor**: Scaling coefficient applied to inputs. (polynomial kernel only) A single value, a vector with range values `c(min_val, max_val)` or NULL for default range c(-5, -1) in log10 scale.

References

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). *An Introduction to Statistical Learning: with Applications in R (2nd ed.)*. Springer. https://doi.org/10.1007/978-1-0716-1418-1

Examples

```
# Example 1: Neural Network for regression task

tidy_object <- build_model(
               analysis_object = tidy_object,
               model_name = "Neural Network",
               hyperparameters = list(
                                 hidden_units = 10,
                                 activation = "relu",
                                 learn_rate = 0.01
                                 )
                           )
# It is safe to reuse the same object name (e.g., tidy_object, or whatever) step by step,
as all previous results and information are retained within the updated analysis object.

# Example 2: SVM for classification task

tidy_object <- build_model(
               analysis_object = tidy_object,
               model_name = "SVM",
               hyperparameters = list(
                                 type = "rbf",
                                 cost = 1,
                                 margin = 0.1,
                                 rbf_sigma = 0.05
                                 )
                           )
```

`fine_tuning` *03 Fine Tune ML Model*

Description

The **fine_tuning()** function performs automated hyperparameter optimization for ML workflows encapsulated within an AnalysisObject. It supports different tuning strategies, such as **Bayesian Optimization** and **Grid Search Cross-Validation**, allowing the user to specify evaluation metrics and whether to visualize tuning results. The function first validates arguments and updates the workflow and metric settings within the AnalysisObject. If hyperparameter tuning is enabled, it executes the selected tuning procedure, identifies the best hyperparameter configuration based on the specified metrics, and updates the workflow accordingly. For neural network models, it also manages the creation and integration of new model instances and provides additional visualization of training dynamics. Finally, the function fits the optimized model to the training data and updates the AnalysisObject, ensuring a reproducible and efficient model selection process (Bartz et al., 2023).

Usage

```
fine_tuning(analysis_object, tuner, metrics, plot_results = F, verbose = FALSE)
```

Arguments

`analysis_object`	analysis_object created from build_model function.
`tuner`	Name of the Hyperparameter Tuner. A string of the tuner name: "Bayesian Optimization" or "Grid Search CV".
`metrics`	Metric used for Model Selection. A string of the name of metric (see Metrics).
`plot_results`	Whether to plot the tuning results. Boolean TRUE or FALSE (default).
`verbose`	Whether to show tuning process. Boolean TRUE or FALSE (default).

Value

An updated analysis_object containing the fitted model with optimized hyperparameters, the tuning results, and all relevant workflow modifications. This object includes the final trained model, the best hyperparameter configuration, tuning diagnostics, and, if applicable, plots of the tuning process. It can be used for further model evaluation, prediction, or downstream analysis within the package workflow.

Tuners

Bayesian Optimization:

- Initial data points: 20
- Maximum number of iterations: 25
- Convergence after 5 iterations without improvement
- Train / Validation / Test : 0.6 / 0.2 / 0.2

Grid Search CV:

- Number of Folds: 5
- Maximum levels per hyperparameter: 10
- Train / Test : 0.75 / 0.25

Metrics

Regression Metrics:

- rmse
- mae
- mpe
- mape
- ccc
- smape
- rpiq
- rsq

Classification Metrics:

- accuracy
- bal_accuracy
- recall
- sensitivity
- specificity
- kap
- f_meas
- mcc
- j_index
- detection_prevelance
- roc_auc
- pr_auc
- gain_capture
- brier_class
- roc_aunp

References

Bartz, E., Bartz-Beielstein, T., Zaefferer, M., & Mersmann, O. (2023). *Hyperparameter tuner for Machine and Deep Learning with R. A Practical Guide*. Springer, Singapore. https://doi.org/10.1007/978-981-19-5170-1

Examples

```
# Example 1: Fine tuning function applied to a regression task

tidy_object <- fine_tuning(tidy_object,
                           tuner = "Bayesian Optimization",
                           metrics = c("rmse", "mape"),
                           plot_results = TRUE
                           )

# Example 2: Fine tuning function applied to a classification task

tidy_object <- fine_tuning(tidy_object,
                           tuner = "Grid Search CV",
                           metrics = c("roc_auc", "f_meas"),
                           plot_results = TRUE
                           )
```

preprocessing *01 Preprocessing Data Matrix*

Description

The **preprocessing()** function streamlines data preparation for regression and classification tasks by integrating variable selection, type conversion, normalization, and categorical encoding into a single workflow. It takes a data frame and a formula, applies user-specified transformations to numeric and categorical variables using the recipes package, and ensures the outcome variable is properly formatted. The function returns an AnalysisObject containing both the processed data and the transformation pipeline, supporting reproducible and efficient modeling (Kuhn & Wickham, 2020).

Usage

```
preprocessing(
  df,
  formula,
  task = "regression",
  num_vars = NULL,
  cat_vars = NULL,
  norm_num_vars = "all",
  encode_cat_vars = "all",
  y_levels = NULL
)
```

Arguments

df	Input DataFrame. Either a data.frame or tibble.
formula	Modelling Formula. A string of characters or formula.
task	Modelling Task. Either "regression" or "classification".
num_vars	Optional vector of names of the numerical features.
cat_vars	Optional vector of names of the categorical features.
norm_num_vars	Normalize numeric features as z-scores. Either vector of names of numerical features to be normalized or "all" (default).
encode_cat_vars	One Hot Encode Categorical Features. Either vector of names of categorical features to be encoded or "all" (default).
y_levels	Optional ordered vector with names of the target variable levels (Classification task only).

Value

The object returned by the preprocessing function encapsulates a dataset specifically prepared for ML analysis. This object contains the preprocessed data—where variables have been selected, standardized, encoded, and formatted according to the requirements of the chosen modeling task (regression or classification) —as well as a recipes::recipe object that documents all preprocessing steps applied. By automating essential transformations such as normalization, one-hot encoding of categorical variables, and the handling of missing values, the function ensures the data is optimally

structured for input into machine learning algorithms. This comprehensive preprocessing not only exposes the underlying structure of the data and reduces the risk of errors, but also provides a robust foundation for subsequent modeling, validation, and interpretation within the machine learning workflow (Kuhn & Johnson, 2019).

References

Kuhn, M., & Johnson, K. (2019). *Feature Engineering and Selection: A Practical Approach for Predictive Models (1st ed.)*. Chapman and Hall/CRC. https://doi.org/10.1201/9781315108230

Kuhn, M., & Wickham, H. (2020). *Tidymodels: a collection of packages for modeling and machine learning using tidyverse principles*. https://www.tidymodels.org.

Examples

```
# Example 1: Dataset with preformatted categorical variables
# In this case, internal options for variable types are not needed since categorical features
are already formatted as factors.

data(sim_data) # sim_data is a simulated dataset wtih psychological variables

tidy_object <- preprocessing(
                            df = sim_data,
                    formula = psych_well ~ depression + emot_intel + resilience + life_sat,
                            task = "regression"
                            )

# Example 2: Dataset where neither the outcome nor the categorical features are formatted as factors
and all categorical variables are specified to be formatted as factors

tidy_object <- preprocessing(
                            df = raw_data,
                            formula = outcome ~ gender + group + age + score,
                            task = "classification",
                            cat_vars = c("outcome", "gender", "group")
                            )
```

---

sensitivity_analysis *05 Perform Sensitivity Analysis and Interpretable ML methods*

Description

As the final step in the TidyML package workflow, this function performs Sensitivity Analysis (SA) on a fitted ML model stored in an `analysis_object` (in the examples, e.g., tidy_object). It evaluates the importance of features using various methods such as Permutation Feature Importance (PFI), SHAP (SHapley Additive exPlanations), Integrated Gradients, Olden sensitivity analysis, and Sobol indices. The function generates numerical results and visualizations (e.g., bar plots, box plots, beeswarm plots) to help interpret the impact of each feature on the model's predictions for both regression and classification tasks, providing critical insights after model training and evaluation.

Usage

```
sensitivity_analysis(analysis_object, methods = c("PFI"), metric = NULL)
```

Arguments

analysis_object	analysis_object created from fine_tuning function.
methods	Method to be used. A string of the method name: "PFI" (Permutation Feature Importance), "SHAP" (SHapley Additive exPlanations), "Integrated Gradients" (Neural Network only), "Olden" (Neural Network only).
metric	Metric used for "PFI" method (Permutation Feature Importance). A string of the name of metric (see Metrics).

Details

Following the steps of data preprocessing, model fitting, and performance assessment in the TidyML pipeline, *sensitivity_analysis()* processes the training and test data using the preprocessing recipe stored in the analysis_object, applies the specified SA methods, and stores the results within the analysis_object. It supports different metrics for evaluation and handles multi-class classification by producing class-specific analyses and plots, ensuring a comprehensive understanding of model behavior (Iooss & Lemaître, 2015).

As the concluding phase of the TidyML workflow—after data preparation, model training, and evaluation—this function enables users to interpret their models by quantifying and visualizing feature importance. It first validates the input arguments using `check_args_sensitivity_analysis()`. Then, it preprocesses the training and test data using the recipe stored in `analysis_object$transformer`. Depending on the specified `methods`, it calculates feature importance using:

- **PFI (Permutation Feature Importance):** Assesses importance by shuffling feature values and measuring the change in model performance (using the specified or default `metric`).
- **SHAP (SHapley Additive exPlanations):** Computes SHAP values to explain individual predictions by attributing contributions to each feature.
- **Integrated Gradients:** Evaluates feature importance by integrating gradients of the model's output with respect to input features.
- **Olden:** Calculates sensitivity based on connection weights, typically for neural network models, to determine feature contributions.
- **Sobol_Jansen:** Performs variance-based global sensitivity analysis by decomposing the model output variance into contributions from individual features and their interactions, quantifying how much each feature and combination of features accounts for the variability in predictions. Only for continuous outcomes, not for categorical. Specifically, estimates first-order and total-order Sobol' sensitivity indices simultaneously using the Jansen (1999) Monte Carlo estimator.

For classification tasks with more than two outcome levels, the function generates separate results and plots for each class. Visualizations include bar plots for importance metrics, box plots for distribution of values, and beeswarm plots for detailed feature impact across observations. All results are stored in the `analysis_object` under the `sensitivity_analysis` slot, finalizing the TidyML pipeline with a deep understanding of model drivers.

Value

An updated `analysis_object` with the results of the sensitivity analysis stored in the `sensitivity_analysis` slot as a list. Each method's results are accessible under named elements (e.g., `sensitivity_analysis[["PFI"]]`). Additionally, the function produces various plots (bar plots, box plots, beeswarm plots) for visual interpretation of feature importance, tailored to the task type and number of outcome levels, completing the TidyML workflow with actionable model insights.

References

Iooss, B., & Lemaître, P. (2015). A review on global sensitivity analysis methods. In C. Meloni & G. Dellino (Eds.), *Uncertainty Management in Simulation-Optimization of Complex Systems: Algorithms and Applications* (pp. 101-122). Springer. https://doi.org/10.1007/978-1-4899-7547-8_5

Jansen, M. J. W. (1999). Analysis of variance designs for model output. Computer Physics Communications, 117(1-2), 35–43. https://doi.org/10.1016/S0010-4655(98)00154-4

Examples

```
# Example 1: Using SHAP

tidy_object <- sensitivity_analysis(analysis_object, methods = "SHAP")

# Example 2: using PFI

tidy_object <- sensitivity_analysis(analysis_object, methods = "PFI", metric = "mape")
```

---

show_results *04 Showcase Summary Results and Plots*

Description

The **show_results()** function is a central component of the ML workflow established by the package, following the stages of data preprocessing, model construction (build_model), and hyperparameter optimization (fine_tuning). After a model has been trained and tuned, show_results() enables users to generate comprehensive **visualizations and summaries of model performance**, including metrics tables, ROC and PR curves, gain and lift curves, confusion matrices, calibration plots, and regression diagnostics, tailored to both regression and classification tasks. This function provides a thorough and interpretable assessment of the fitted model, supporting informed evaluation and communication of results. Importantly, show_results() is not the final step of the workflow, as further analyses such as sensitivity analysis can be performed subsequently to deepen the understanding of model robustness and behavior (Molnar, 2025).

Usage

```
show_results(
  analysis_object,
  summary = FALSE,
  roc_curve = FALSE,
  pr_curve = FALSE,
  gain_curve = FALSE,
  lift_curve = FALSE,
  dist_by_class = FALSE,
  reliability_plot = FALSE,
  confusion_matrix = FALSE,
  scatter_residuals = FALSE,
  scatter_predictions = FALSE,
  residuals_dist = FALSE,
  new_data = "test"
)
```

Arguments

analysis_object	analysis_object created from fine_tuning function.
summary	Whether to plot summary results table. Boolean (FALSE by default).
roc_curve	Whether to plot ROC Curve (Classification task only). Boolean (FALSE by default).
pr_curve	Whether to plot ROC Curve (Classification task only). Boolean (FALSE by default).
gain_curve	Whether to plot ROC Curve (Classification task only). Boolean (FALSE by default).
lift_curve	Whether to plot ROC Curve (Classification task only). Boolean (FALSE by default).
dist_by_class	Whether to plot distribution of output probability by class (Classification task only). Boolean (FALSE by default).
reliability_plot	Whether to plot Reliability Plot (Binary Classification task only). Boolean (FALSE by default).
confusion_matrix	Whether to Confusion Matrix (Classification task only). Boolean (FALSE by default).
scatter_residuals	Whether to plot Residuals vs Predictions (Regression task only). Boolean (FALSE by default).
scatter_predictions	Whether to plot Predictions vs Observed (Regression task only). Boolean (FALSE by default).
residuals_dist	Whether to plot Residuals Distribution (Regression task only). Boolean (FALSE by default).
new_data	Data to be used for Confusion Matrix, Reliability Plot, Distribution by Class Plot, Residuals vs Predictions Plot, Predictions vs Observed Plot and Residuals Distribution Plot. A string with the name of the data_set: "train", "validation", "test" (default) or "all".

Value

An updated analysis_object containing the generated predictions, summary statistics, and any visualizations or diagnostic outputs selected by the user. This object reflects the results of model evaluation and can be further used for reporting, interpretation, or additional analyses within the ML tidyML workflow.

References

Molnar, C. (2025). *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable (3rd. ed.)*. cristophm.github.io/interpretable-ml-book/

Examples

```
# Example 1: Classification Task
# Display summary metrics, ROC curve, and confusion matrix for a classification model with test partition
```

```
tidy_object<-show_results(tidy_object,
                          summary = TRUE,
                          roc_curve = TRUE,
                          confusion_matrix = TRUE,
                          new_data = "test")

# Example 2: Regression Task
# Display summary metrics, scatter plot of predictions, and residuals distribution for a regression model
with train partition

tidy_object<-show_results(tidy_object,
                          summary = TRUE,
                          scatter_predictions = TRUE,
                          residuals_dist = TRUE,
                          new_data = "train")
```

Index