Package 'tidypredict'

October 18, 2026

Title Run Predictions Inside the Database

Version 1.2.2

Description It parses a fitted 'R' model object, and returns a formula in 'Tidy Eval' code that calculates the predictions. It works with several databases back-ends because it leverages 'dplyr' and 'dbplyr' for the final 'SQL' translation of the algorithm. Dozens of model classes are supported; see the ``Supported models'' article at <https://tidypredict.tidymodels.org/articles/models.html> for the current list.

License MIT + file LICENSE

URL https://tidypredict.tidymodels.org, https://github.com/tidymodels/tidypredict

BugReports https://github.com/tidymodels/tidypredict/issues

Depends R (>= 4.1)

Imports cli, dplyr (>= 0.7), generics, jsonlite, knitr, lifecycle, purrr, rlang (>= 1.1.1), tibble, tidyr, utils

Suggests agua, aorsf, baguette, bonsai, C50, covr, Cubist (>= 0.5.1), data.table, dbarts, DBI, dbplyr, discrim, earth (>= 5.1.2), glmnet, h2o, kernlab, klaR, LiblineaR, lightgbm, MASS, mboost, mda, methods, mixOmics, mlbench, modeldata, naivebayes, nnet, nycflights13, parsnip, partykit, plsmod, quantreg, randomForest, ranger (>= 0.14.1), RhpcBLASctl, rmarkdown, rpart (>= 4.1.0), RSQLite, rules, sda, sparsediscrim, survival, testthat (>= 3.2.0), withr, xgboost, xrf, yaml

VignetteBuilder knitr

Config/Needs/website tidyverse/tidytemplate

Config/roxygen2/version 8.0.0

Config/testthat/edition 3

Config/usethis/last-upkeep 2026-08-19

Encoding UTF-8

NeedsCompilation no

Author Emil Hvitfeldt [aut, cre] (ORCID:
<https://orcid.org/0000-0002-0679-1945>),
Edgar Ruiz [aut],
Max Kuhn [aut] (ORCID: <https://orcid.org/0000-0003-2402-136X>),
Posit Software, PBC [cph, fnd] (ROR: <https://ror.org/03wc8by49>)

Maintainer Emil Hvitfeldt <emil.hvitfeldt@posit.co>

Repository CRAN

Date/Publication 2026-10-18 10:30:02 UTC

Contents

acceptable_formula 2
as_parsed_model 3
parse_model 3
set_catboost_categories 5
tidy.pm_regression 6
tidypredict_combine_trees 6
tidypredict_extractors 8
tidypredict_fit 10
tidypredict_interval 11
tidypredict_metadata 11
tidypredict_save 13
tidypredict_sql 14
tidypredict_sql_interval 15
tidypredict_test 15
tidypredict_to_column 16

Index **18**

`acceptable_formula` *Checks that the formula can be parsed*

Description

Uses an S3 method to check that a given formula can be parsed based on its class. It currently scans for contrasts that are not supported and in-line functions. (e.g: lm(wt ~ as.factor(am))). Since this function is meant for function interaction, as opposed to human interaction, a successful check is silent.

Usage

```
acceptable_formula(model)
```

Arguments

`model`	An R model object

Value

NULL (invisibly) when the model's formula can be parsed, or an error when it cannot.

Examples

```
model <- lm(mpg ~ wt, mtcars)
acceptable_formula(model)
```

---

as_parsed_model *Prepares parsed model object*

Description

Prepares parsed model object

Usage

```
as_parsed_model(x)
```

Arguments

x	A parsed model object

Value

The parsed model with its parsed_model and pm_* classes set.

Examples

```
pm <- parse_model(lm(mpg ~ wt, data = mtcars))
as_parsed_model(pm)
```

---

parse_model *Converts an R model object into a parsed model*

Description

Parses a fitted R model's structure and extracts the components needed to create a dplyr formula for prediction. The parsed model can be serialized (e.g., saved to YAML) and later used to generate predictions without the original model object.

Usage

```
parse_model(model)
```

Arguments

model	An R model object.

Value

A parsed model object with class parsed_model and a model-specific subclass (e.g., pm_xgb, pm_tree, pm_regression). The object contains:

- $general: List with model metadata including model (model type), type (used for S3 dispatch), version (parsed model format version), and model-specific parameters.
- Model-specific fields containing coefficients, tree structures, etc.

Parsed model versions

The $general$version field indicates the parsed model format:

- **Version 1**: Original format. Linear models store coefficients in a data frame. Tree models use flat case_when() expressions where all leaf conditions are at the same level.
- **Version 2**: Improved coefficient storage for linear models (lm, earth). Tree models still use flat case_when().
- **Version 3**: Current format. Tree models (rpart, ranger, randomForest, xgboost, lightgbm, catboost, partykit) use nested case_when() expressions that mirror the tree structure. This produces more efficient SQL and R code because conditions are evaluated hierarchically rather than checking all leaf paths.

When loading a parsed model saved with an older version, tidypredict automatically uses the appropriate formula builder for backwards compatibility.

Model types

Each parsed model has a type that determines the S3 class used for dispatch:

- pm_regression: Linear models (lm, glm, earth, glmnet)
- pm_tree: Single trees and forests (rpart, partykit, ranger, randomForest, cubist)
- pm_xgb: XGBoost gradient boosting models
- pm_lgb: LightGBM gradient boosting models
- pm_catboost: CatBoost gradient boosting models
- pm_bart: BART models
- pm_naive_bayes: Naive Bayes models
- pm_nnet: Neural network models
- pm_multiclass_regression: Multiclass regression models
- pm_nullmodel_classification: Null model classification

This list is not exhaustive; new model types are added as support grows.

Examples

```
library(dplyr)
df <- mutate(mtcars, cyl = paste0("cyl", cyl))
model <- lm(mpg ~ wt + cyl * disp, offset = am, data = df)
parse_model(model)
```

---

set_catboost_categories

Set categorical feature mappings for CatBoost model

Description

CatBoost stores categorical features as hash values internally. This function establishes the mapping between hash values and category names by examining a data frame with the same factor columns used during training.

Usage

```
set_catboost_categories(parsed_model, model, data)
```

Arguments

`parsed_model`	A parsed CatBoost model from `parse_model()`
`model`	The original CatBoost model object
`data`	A data frame containing factor columns matching the categorical features used in the model. The factor levels must match those from training.

Details

This function is only needed when using raw CatBoost models (trained with `catboost.train()`). When using parsnip/bonsai, categorical features are handled automatically and this function is not required.

Value

The parsed model with category mappings added

Examples

```
# For raw CatBoost models with categorical features:
pm <- parse_model(catboost_model)
pm <- set_catboost_categories(pm, catboost_model, training_data)
tidypredict_fit(pm)

# For parsnip/bonsai models, this is not needed:
# tidypredict_fit(parsnip_model_fit)  # works automatically
```

tidy.pm_regression *Tidy the parsed model results*

Description

Tidy the parsed model results

Usage

```
## S3 method for class 'pm_regression'
tidy(x, ...)
```

Arguments

x	A parsed_model object
...	Reserved for future use

Value

A tibble with one row per term, containing the `term` name and its `estimate`.

Examples

```
pm <- parse_model(lm(mpg ~ wt, data = mtcars))
tidy(pm)
```

tidypredict_combine_trees *Combine per-tree expressions into a model's prediction*

Description

`tidypredict_trees()` returns one expression per tree. Turning those back into the model's prediction is not simply summing or averaging them, and the rule differs by backend: `mboost::blackboost()` needs an offset and a shrinkage factor, CatBoost needs a scale and a bias, `aorsf` needs a guard that returns `NA` for an incomplete row, and boosters then apply their objective's inverse link on top.

This generic holds that rule, so a caller that has split the trees apart can put them back together without knowing which backend it is holding.

Usage

```
tidypredict_combine_trees(x, trees, ...)
```

Arguments

x	A fitted model object.
trees	A list of expressions, one per tree, in the order tidypredict_trees() returns them. Typically either that return value itself, or symbols naming the columns the individual trees were written to.
...	Additional arguments passed to methods.

Details

The point of separating trees from this function is that a caller can compute each tree into its own column, for a database to evaluate in parallel, and then pass symbols referring to those columns rather than the expressions themselves. The combination is the same either way.

A caller may also pass fewer elements than the model has trees, where each one is a subtotal of several trees added together. This is what a caller does to keep any single generated expression short enough for a database parser. Every method supports it: those that average take the divisor from the model rather than from length(trees), and the rest sum. The one thing a caller must not do is reorder or drop trees, since a method is free to treat position as meaningful.

Every ensemble satisfies tidypredict_combine_trees(x, tidypredict_trees(x)) computing the same values as tidypredict_fit(x), and that identity is what the tests for these methods assert.

Not every ensemble has a method. C50::C5.0() boosting combines its trees by a confidence-weighted vote that yields a class label, so there is no arithmetic to apply to per-tree numbers and no method is provided.

Value

A single language object.

See Also

tidypredict_extractors for the generics that produce trees, including a table of which model classes implement them, and tidypredict_metadata for what the combined value means.

Examples

```
model <- randomForest::randomForest(mpg ~ ., data = mtcars, ntree = 3)

trees <- tidypredict_trees(model)
tidypredict_combine_trees(model, trees)

# Or referring to columns the trees were written to first.
tidypredict_combine_trees(model, rlang::syms(c("t1", "t2", "t3")))
```

`tidypredict_extractors` *Extract model internals as expressions*

Description

These generics expose the pieces `tidypredict_fit()` is assembled from, rather than the finished formula. They exist so that packages generating their own code from a fitted model, such as orbital, can reuse tidypredict's parsing instead of reimplementing it per model class.

Each generic has a single fixed return shape, described below. A model class implements whichever generics make sense for it: a random forest has trees and a tree count, a `glmnet` multinomial model has neither.

Usage

```
tidypredict_trees(x, ...)

tidypredict_class_trees(x, ...)

tidypredict_class_exprs(x, ...)

tidypredict_n_trees(x, ...)
```

Arguments

`x`	A fitted model object.
`...`	Additional arguments passed to methods. `multnet` models accept `penalty`, which is required when the model was fitted with more than one value of lambda.

Details

Two shapes that look similar are worth keeping apart. `tidypredict_class_trees()` returns many trees per level that a caller has to sum or average, and needs `tidypredict_n_trees()` to do it. `tidypredict_class_exprs()` returns one finished expression per level. Both are named by outcome level so that callers never have to assume the order matches `levels()` of the outcome.

Value

`tidypredict_trees()` returns an unnamed list with one element per tree, each a language object.

`tidypredict_class_trees()` returns a list named by outcome level, in model order. Each element is itself an unnamed list of per-tree language objects for that level, so the result is `tidypredict_trees()` nested one level deeper. What the leaves hold depends on the model: `randomForest` gives 0/1 votes, `ranger` gives class probabilities.

`tidypredict_class_exprs()` returns a list named by outcome level, in model order, with one language object per level. Unlike `tidypredict_class_trees()` there is no per-tree structure and nothing to combine: each expression computes that level's value on its own.

`tidypredict_n_trees()` returns a single integer, the number of trees in the ensemble. For multiclass boosters this counts every tree, including the per-class copies, so it is not the same as the number of boosting rounds.

Wherever an expression is described above, a **bare numeric value** may appear in its place when the model has nothing to branch on. Callers must handle both. This happens for a single-leaf tree, a stump, and also for a degenerate expression such as a `glmnet` class whose coefficients are all zero. Note that the constant can appear alongside language objects in the same result, so the element type is not uniform within one list.

Which models implement which generic

`.` means the generic is not implemented for that class and will error.

	trees	class_trees	class_exprs	n_trees	combine
blackboost	x	.	.	x	x
C5.0	.	.	.	.	x
catboost.Model	x	.	.	x	x
cforest	x	.	.	x	x
earth	.	.	x	.	.
lgb.Booster	x	.	.	x	x
multnet	.	.	x	.	.
ObliqueForest	x	.	.	x	x
party	.	.	x	.	.
randomForest	x	x	.	x	x
ranger	x	x	.	x	x
rpart	.	.	x	.	.
xgb.Booster	x	.	.	x	x

`C50::C5.0()` is the one row with a `tidypredict_combine_trees()` method and no `tidypredict_trees()`. That method exists only to refuse, with an explanation, rather than to let the caller reach the `.default` error and guess why.

Implementing these for a new model class

The table above shows the grouping to follow. `tidypredict_trees()`, `tidypredict_n_trees()` and `tidypredict_combine_trees()` are a set: implement all three or none. Per-tree expressions are not usable without a count to size them and a rule to recombine them, and shipping the first without the third invites a caller to sum the trees, which is wrong for every backend that carries an offset, a scale or a link.

A useful check on a new method is that `tidypredict_combine_trees(x, tidypredict_trees(x))` computes the same values as `tidypredict_fit(x)`. That identity is what the tests for the existing methods assert, and it catches a combination rule that was assumed rather than read out of the model.

If a model's trees genuinely cannot be recombined arithmetically, give it a `tidypredict_combine_trees()` method that refuses and no `tidypredict_trees()` method, as `C50::C5.0()` does. Splitting trees apart that cannot be put back together only enables a wrong answer.

See Also

tidypredict_combine_trees() for turning per-tree expressions back into a prediction, and tidypredict_metadata for what the resulting values mean.

Examples

```
model <- randomForest::randomForest(mpg ~ ., data = mtcars, ntree = 5)

tidypredict_n_trees(model)

trees <- tidypredict_trees(model)
length(trees)
trees[[1]]
```

---

tidypredict_fit *Returns a Tidy Eval formula to calculate fitted values*

Description

It parses a model or uses an already parsed model to return a Tidy Eval formula that can then be used inside a dplyr command.

Usage

```
tidypredict_fit(model)
```

Arguments

model	An R model or a list with a parsed model.

Value

A Tidy Eval formula that calculates the fitted values. Models that produce one formula per class or per outcome return a list of formulas.

Examples

```
model <- lm(mpg ~ wt + cyl * disp, offset = am, data = mtcars)
tidypredict_fit(model)
```

tidypredict_interval *Returns a Tidy Eval formula to calculate prediction interval.*

Description

It parses a model or uses an already parsed model to return a Tidy Eval formula that can then be used inside a dplyr command.

Usage

```
tidypredict_interval(model, interval = 0.95)
```

Arguments

model	An R model or a list with a parsed model
interval	The prediction interval, defaults to 0.95

Details

The result still has to be added to and subtracted from the fit to obtain the upper and lower bound respectively.

Value

A Tidy Eval formula that calculates the half width of the prediction interval. It must be added to and subtracted from the fit to obtain the upper and lower bounds.

Examples

```
model <- lm(mpg ~ wt + cyl * disp, offset = am, data = mtcars)
tidypredict_interval(model)
```

tidypredict_metadata *Describe what a model's fitted expressions compute*

Description

`tidypredict_fit()` returns expressions, but not what those expressions mean. A single expression could be a numeric prediction, a probability, or an uncalibrated decision value, and the three call for different handling downstream. These generics answer that question, so that a package generating code from the result does not have to keep its own list of which backend produces which shape.

The metadata is asked of the *model*, not of the fitted expressions, deliberately. Attributes on the result do not survive the subsetting, `lapply()` and `unlist()` that callers apply to a multiclass result, which is the case that most needs describing.

Usage

```
tidypredict_output_type(x, ...)

tidypredict_outcome_levels(x, ...)

tidypredict_normalized(x, ...)
```

Arguments

x	A fitted model object.
...	Additional arguments passed to methods.

Details

None of this is recoverable from the shape of the result, which is the whole reason for recording it. Two concrete cases:

A binary "prob" model and a "decision" model both return exactly one expression. LiblineaR produces either, depending only on its type argument. Treating a decision value as a probability and cutting it at 0.5 gives silently wrong classes for every row whose value falls between 0 and 0.5.

A multiclass "prob" model and a quantreg::rq() fit with several tau both return a named list of expressions of the same length and structure. In the first the values sum to one across the list; in the second they are unrelated numeric predictions.

Value

tidypredict_output_type() returns a single string, one of:

"numeric" A numeric prediction. tidypredict_fit() returns one expression, or a named list of them for a multivariate outcome or for a quantile regression with several tau.

"prob" A probability. Either one expression giving the probability of the second outcome level, for a binary model, or a list with one expression per level.

"decision" An uncalibrated decision value whose *sign* selects the class. Not a probability, and not comparable to one: the cut is at 0, not 0.5.

"class" A hard class prediction, with no probability available. Usually the class label as a string, but xgboost's binary:hinge objective gives a 0/1 indicator instead. What makes it "class" rather than "numeric" is that only the class values can occur, so using it as a numeric prediction is a mistake even when its type is numeric.

tidypredict_outcome_levels() returns a character vector of outcome levels in model order, or NULL.

NULL means two different things, and tidypredict_output_type() distinguishes them. For a "numeric" model it means there are no levels. For a "prob" or "class" model it means the fitted model **did not retain the outcome levels**, so any names on the result are positional placeholders and the caller has to supply the real levels from elsewhere. LightGBM and CatBoost multiclass models are in this position: they store integer labels and their expressions come back named class_0, class_1 and so on.

`tidypredict_normalized()` returns `TRUE` if the per-level values already sum to one across levels, `FALSE` if the caller has to normalize them, and `NA` when there are no per-level values to sum, which includes every single-expression model.

At present no backend returns `FALSE`: every multiclass probability list goes through one shared softmax, so the values are always normalized already. The generic exists so that a caller can rely on that rather than having to know it, and so a future backend that does not normalize can say so instead of silently breaking the assumption.

See Also

tidypredict_extractors for the generics that expose a model's per-tree and per-level expressions, and `tidypredict_combine_trees()` for recombining them.

Examples

```
model <- lm(mpg ~ wt, data = mtcars)
tidypredict_output_type(model)
tidypredict_outcome_levels(model)
tidypredict_normalized(model)
```

`tidypredict_save` *Save and re-load a parsed model*

Description

`tidypredict_save()` writes a parsed model to a YAML file, and `tidypredict_load()` reads one back. Together they persist a model's prediction formula without needing the original model object, or the package that fitted it, to be available later.

Use these rather than calling `yaml::write_yaml()` directly. `yaml` defaults to 7 significant digits, which is not enough to represent a split threshold exactly: a re-loaded tree model can then send rows down a different branch than the model it was saved from.

Usage

```
tidypredict_save(x, file)

tidypredict_load(file)
```

Arguments

`x`	A fitted model, or a parsed model from `parse_model()`. Fitted models are parsed before being saved.
`file`	Path to write the YAML file to, or read it from.

Value

`tidypredict_save()` returns `x`, invisibly, so it can be used in a pipe. `tidypredict_load()` returns a parsed model object.

Examples

```
model <- lm(mpg ~ wt + cyl, data = mtcars)

path <- tempfile(fileext = ".yml")
tidypredict_save(model, path)

loaded <- tidypredict_load(path)
tidypredict_fit(loaded)
```

---

tidypredict_sql *Returns a SQL query with formula to calculate fitted values*

Description

Returns a SQL query with formula to calculate fitted values

Usage

```
tidypredict_sql(model, con)
```

Arguments

`model`	An R model or a list with a parsed model
`con`	Database connection object. It is used to select the correct SQL translation syntax.

Value

A SQL query, as returned by `dbplyr::translate_sql()`. Models that produce one formula per class or per outcome return a list of queries.

Examples

```
model <- lm(mpg ~ wt + am + cyl, data = mtcars)
tidypredict_sql(model, dbplyr::simulate_dbi())
```

tidypredict_sql_interval *Returns a SQL query with formula to calculate predicted interval*

Description

Returns a SQL query with formula to calculate predicted interval

Usage

```
tidypredict_sql_interval(model, con, interval = 0.95)
```

Arguments

model	An R model or a tibble with a parsed model
con	Database connection object. It is used to select the correct SQL translation syntax.
interval	The prediction interval, defaults to 0.95

Value

A SQL query, as returned by dbplyr::translate_sql(), giving the half width of the prediction interval.

Examples

```
model <- lm(mpg ~ wt + am + cyl, data = mtcars)
tidypredict_sql_interval(model, dbplyr::simulate_dbi())
```

tidypredict_test *Tests base predict function against tidypredict*

Description

Compares the results of predict() and tidypredict_to_column() functions.

Usage

```
tidypredict_test(
  model,
  df = model$model,
  threshold = 1e-12,
  include_intervals = FALSE,
  max_rows = NULL,
  xg_df = NULL
)
```

Arguments

model	An R model or a list with a parsed model.
df	A data frame that contains all of the needed fields to run the prediction. It defaults to the "model" data frame object inside the model object.
threshold	The number that a given result difference, between predict() and tidypredict_to_column() should not exceed. For continuous predictions, the default value is 0.000000000001 (1e-12), and for categorical predictions, the default value is 0.
include_intervals	Switch to indicate if the prediction intervals should be included in the test. It defaults to FALSE.
max_rows	The number of rows in the object passed in the df argument. Highly recommended for large data sets.
xg_df	The prediction matrix used to obtain the model's own predictions. Required for XGBoost, LightGBM and CatBoost models, which cannot predict from a data frame. Pass an `xgb.DMatrix` for XGBoost and a numeric matrix for LightGBM and CatBoost. It defaults to NULL.

Value

A list of test results comparing `predict()` and `tidypredict_to_column()`, including the maximum difference and whether it stays within `threshold`.

Examples

```
model <- lm(mpg ~ wt + cyl * disp, offset = am, data = mtcars)
tidypredict_test(model)
```

---

tidypredict_to_column *Adds the prediction columns to a piped command set.*

Description

Adds a new column with the results from tidypredict_fit() to a piped command set. If add_interval is set to TRUE, it will add two additional columns- one for the lower and another for the upper prediction interval bounds.

Usage

```
tidypredict_to_column(
  df,
  model,
  add_interval = FALSE,
  interval = 0.95,
  vars = c("fit", "upper", "lower")
)
```

Arguments

df	A data.frame or tibble
model	An R model or a parsed model inside a data frame
add_interval	Switch that indicates if the prediction interval columns should be added. Defaults to FALSE
interval	The prediction interval, defaults to 0.95. Ignored if add_interval is set to FALSE
vars	The name of the variables that this function will produce. Defaults to "fit", "upper", and "lower".

Value

The input data frame with one new column (the fit) added, or three new columns (fit, upper and lower bounds) when `add_interval` is `TRUE`.

Examples

```
model <- lm(mpg ~ wt, data = mtcars)
tidypredict_to_column(mtcars, model)
```

Index

acceptable_formula, 2
as_parsed_model, 3

dbplyr::translate_sql(), *14, 15*

parse_model, 3
parse_model(), *13*

set_catboost_categories, 5

tidy.pm_regression, 6
tidypredict_class_exprs
        *(*tidypredict_extractors*)*, 8
tidypredict_class_trees
        *(*tidypredict_extractors*)*, 8
tidypredict_combine_trees, 6
tidypredict_combine_trees(), *10*, *13*
tidypredict_extractors, *7*, 8, *13*
tidypredict_fit, 10
tidypredict_interval, 11
tidypredict_load *(*tidypredict_save*)*, 13
tidypredict_metadata, *7*, *10*, 11
tidypredict_n_trees
        *(*tidypredict_extractors*)*, 8
tidypredict_normalized
        *(*tidypredict_metadata*)*, 11
tidypredict_outcome_levels
        *(*tidypredict_metadata*)*, 11
tidypredict_output_type
        *(*tidypredict_metadata*)*, 11
tidypredict_save, 13
tidypredict_sql, 14
tidypredict_sql_interval, 15
tidypredict_test, 15
tidypredict_to_column, 16
tidypredict_trees
        *(*tidypredict_extractors*)*, 8
tidypredict_trees(), *6, 7*

yaml::write_yaml(), *13*