Package ‘tabpfn’

March 18, 2026

Title Prior-Data Fitted Network Foundational Model for Tabular Data

Version 0.1.0

Description Provides a consistent API for classification and regression models based on the 'TabPFN' model of Hollmann et al. (2025), ``Accurate predictions on small data with a tabular foundation model,'' Nature, 637(8045) <doi:10.1038/s41586-024-08328-6>. The calculations are served via 'Python' to train and predict the model.

License Apache License (>= 2)

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

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

Depends R (>= 4.1.0)

Imports cli, dplyr, generics, hardhat, purrr, reticulate (>= 1.41.0.1), rlang (>= 1.1.0), tibble

Suggests covr, modeldata, recipes, spelling, testthat (>= 3.0.0)

Config/Needs/website tidyverse/tidytemplate

Config/testthat/edition 3

Encoding UTF-8

Language en-US

RoxygenNote 7.3.3

NeedsCompilation no

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

Maintainer Max Kuhn <max@posit.co>

Repository CRAN

Date/Publication 2026-03-18 10:30:02 UTC

Contents

`control_tab_pfn` *Controlling TabPFN execution*

Description

Controlling TabPFN execution

Usage

```
control_tab_pfn(
  n_preprocessing_jobs = 1L,
  device = "auto",
  ignore_pretraining_limits = FALSE,
  inference_precision = "auto",
  fit_mode = "fit_preprocessors",
  memory_saving_mode = "auto",
  random_state = sample.int(10^6, 1)
)
```

Arguments

`n_preprocessing_jobs`
: An integer for the number of worker processes. A value of -1L indicates all possible resources.

`device`
: A character value for the device used for torch (e.g., `"cpu"`, `"cuda"`, `"mps"`, etc.). Th default is `"auto"`.

`ignore_pretraining_limits`
: A logical to bypass the default data limits on:the number of training set samples (10,000) and, the number of predictors (500). There is an unchangeable limit to the number of classes (10).

`inference_precision`
: A character value for the trade off between speed and reproducibility. This can be a torch `dtype`, `"autocast"` (for torch's mixed-precision autocast), or "auto".

`fit_mode`
: A character value to control how the are preprocessed and/or cached. Values are `"fit_preprocessors"` (the default), `"low_memory"`, `"fit_with_cache"`, and `"batched"`.

`memory_saving_mode`
: A character string to help with out-of-memory errors. Values are either a logical or `"auto"`.

`random_state`
: An integer to set the random number stream.

Value

A list with extra class "control_tab_pfn" that has named elements for each of the argument values.

References

https://github.com/PriorLabs/TabPFN/blob/main/src/tabpfn/classifier.py, https://github.com/PriorLabs/TabPFN/blob/main/src/tabpfn/regressor.py

Examples

```
control_tab_pfn()
```

is_tab_pfn_installed *Check the Python package installation*

Description

Attempts to import the Python package

Usage

```
is_tab_pfn_installed()
```

Value

A single logical

Examples

```
if (interactive()) {
 # This may take a minute
 is_tab_pfn_installed()
}
```

predict.tab_pfn *Predict using* TabPFN

Description

Predict using TabPFN

Usage

```
## S3 method for class 'tab_pfn'
predict(object, new_data, ...)

## S3 method for class 'tab_pfn'
augment(x, new_data, ...)
```

Arguments

object, x	A tab_pfn object.
new_data	A data frame or matrix of new predictors.
...	Not used, but required for extensibility.

Value

`predict()` returns a tibble of predictions and `augment()` appends the columns in `new_data`. In either case, the number of rows in the tibble is guaranteed to be the same as the number of rows in `new_data`.

For regression data, the prediction is in the column `.pred`. For classification, the class predictions are in `.pred_class` and the probability estimates are in columns with the pattern `.pred_{level}` where `level` is the levels of the outcome factor vector.

Examples

```
# Minimal example for quick execution
car_train <- mtcars[ 1:5,   ]
car_test  <- mtcars[6, -1]

## Not run:
# Fit
if (is_tab_pfn_installed() & interactive()) {
 mod <- tab_pfn(mpg ~ cyl + log(drat), car_train)

 # Predict
 predict(mod, car_test)
 augment(mod, car_test)
}

## End(Not run)
```

tab_pfn *Fit a TabPFN model.*

Description

`tab_pfn()` applies data to a pre-estimated deep learning model defined by Hollmann *et al* (2025). This model emulates Bayesian inference for regression and classification models.

Usage

```
tab_pfn(x, ...)

## Default S3 method:
tab_pfn(x, ...)

## S3 method for class 'data.frame'
tab_pfn(
  x,
  y,
  num_estimators = 8L,
  softmax_temperature = 0.9,
  balance_probabilities = FALSE,
  average_before_softmax = FALSE,
  training_set_limit = 10000,
  control = control_tab_pfn(),
  ...
)

## S3 method for class 'matrix'
tab_pfn(
  x,
  y,
  num_estimators = 8L,
  softmax_temperature = 0.9,
  balance_probabilities = FALSE,
  average_before_softmax = FALSE,
  training_set_limit = 10000,
  control = control_tab_pfn(),
  ...
)

## S3 method for class 'formula'
tab_pfn(
  formula,
  data,
  num_estimators = 8L,
  softmax_temperature = 0.9,
```

```
  balance_probabilities = FALSE,
  average_before_softmax = FALSE,
  training_set_limit = 10000,
  control = control_tab_pfn(),
  ...
)

## S3 method for class 'recipe'
tab_pfn(
  x,
  data,
  num_estimators = 8L,
  softmax_temperature = 0.9,
  balance_probabilities = FALSE,
  average_before_softmax = FALSE,
  training_set_limit = 10000,
  control = control_tab_pfn(),
  ...
)
```

Arguments

`x` Depending on the context:

- A **data frame** of predictors.
- A **matrix** of predictors.
- A **recipe** specifying a set of preprocessing steps created from `recipes::recipe()`.

`...` Not currently used, but required for extensibility.

`y` When `x` is a **data frame** or **matrix**, `y` is the outcome specified as:

- A **data frame** with 1 numeric column.
- A **matrix** with 1 numeric column.
- A numeric **vector** for regression or a **factor** for classification.

`num_estimators` An integer for the ensemble size. Default is `8L`.

`softmax_temperature`
An adjustment factor that is a divisor in the exponents of the softmax function (see Details below). Defaults to 0.9.

`balance_probabilities`
A logical to adjust the prior probabilities in cases where there is a class imbalance. Default is `FALSE`. Classification only.

`average_before_softmax`
A logical. For cases where `num_estimators` > 1, should the average be done before using the softmax function or after? Default is `FALSE`.

`training_set_limit`
An integer greater than 2L (and possibly `Inf`) that can be used to keep the training data within the limits of the data constraints imposed by the Python library.

`control` A list of options produced by `control_tab_pfn()`.

formula	A formula specifying the outcome terms on the left-hand side, and the predictor terms on the right-hand side.
data	When a **recipe** or **formula** is used, `data` is specified as: • A **data frame** containing both the predictors and the outcome.

Details

Computing Requirements:

This model can be used with or without a graphics processing unit (GPU). However, it is fairly limited when used with a CPU (and no GPU). There might be additional data size limitation warnings with CPU computations, and, understandably, the execution time is much longer. CPU computations can also consume a significant amount of system memory, depending on the size of your data.

GPUs using CUDA (Compute Unified Device Architecture) are most effective. Limited testing with others has shown that GPUs with Metal Performance Shaders (MPS) instructions (e.g., Apple GPUs) have limited utility for these specific computations and might be slower than the CPU for some data sets.

License Requirements:

On November 6, 2025, PriorLabs released version 2.5 of the model, which contained several improvements. One other change is that accessing the model parameters required an API key. Without one, an error occurs:

"This model is gated and requires you to accept its terms. Please follow these steps: 1. Visit `https://huggingface.co/Prior-Labs/tabpfn_2_5` in your browser and accept the terms of use. 2. Log in to your Hugging Face account via the command line by running: hf auth login (Alternatively, you can set the HF_TOKEN environment variable with a read token)."

The license contains provisions for "Non-Commercial Use Only" usage if that is relevant for you.

To get an API key, use the `huggingface` link above, create an account, and then get an API key. Once you have that, put it in your `.Renviron` file in the form of:

```
HF_TOKEN=your_api_key_value
```

The **usethis** function `edit_r_environ()` can be very helpful here.

Python Installation:

You will need a working Python virtual environment with the correct packages to use these modeling functions.

There are at least two ways to proceed.

Ephemeral uv *Install:*

The first approach, which we *strongly suggest*, is to simply load this package and attempt to run a model. This will prompt **reticulate** to create an ephemeral environment and automatically install the required packages. That process would look like this:

```
> library(tabpfn)
>
> predictors <- mtcars[, -1]
> outcome <- mtcars[, 1]
>
```

```
> # XY interface
> mod <- tab_pfn(predictors, outcome)
Downloading uv...Done!
Downloading cpython-3.12.12 (download) (15.9MiB)
 Downloading cpython-3.12.12 (download)
Downloading setuptools (1.1MiB)
Downloading scikit-learn (8.2MiB)
Downloading numpy (4.9MiB)

<downloading and installing more packages>

 Downloading llvmlite
 Downloading torch
Installed 58 packages in 350ms
> mod
TabPFN Regression Model

Training set
i 32 data points
i 10 predictors
```

The location of the environment can be found at `tools::R_user_dir("reticulate", "cache")`. See the documentation for `reticulate::py_require()` to learn more about this method.

Manually created `venv` *Virtual Environment:*
Alternatively, you can use the functions in the **reticulate** package to create a virtual environment and install the required Python packages there. An example pattern is:

```
library(reticulate)

venv_name <- "r-tabpfn"    # exact name can be different
venv_seed_python <-
  virtualenv_starter(">=3.11,<3.14")

virtualenv_create(
  envname = venv_name,
  python = venv_seed_python,
  packages = c("numpy", "tabpfn")
)
```

Once you have that virtual environment installed, you can declare it as your preferred Python installation with `use_virtualenv()`. (You must do this before reticulate has initialized Python, i.e., before attempting to use **tabpfn**):

```
reticulate::use_virtualenv("r-tabpfn")
```

Data:
Be default, there are limits to the training data dimensions:

- Version 2.0: number of training set samples (10,000) and, the number of predictors (500). There is an unchangeable limit to the number of classes (10).
- Version 2.5: number of training set samples (50,000) and, the number of predictors (2,000). There is an unchangeable limit to the number of classes (10).

Predictors do not require preprocessing; missing values and factor vectors are allowed.

Calculations:

For the `softmax_temperature` value, the softmax terms are:

```
exp(value / softmax_temperature)
```

A value of `softmax_temperature = 1` results in a plain softmax value.

Value

A `tab_pfn` object with elements:

- `fit`: the python object containing the model.
- `levels`: a character string of class levels (or NULL for regression)
- `training`: a vector with the training set dimensions.
- `logging`: any R or python messages produced by the computations.
- `blueprint`: am object produced by `hardhat::mold()` used to process new data during prediction.

References

Hollmann, Noah, Samuel Müller, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmeister, and Frank Hutter. "Accurate predictions on small data with a tabular foundation model." *Nature* 637, no. 8045 (2025): 319-326.

Hollmann, Noah, Samuel Müller, Katharina Eggensperger, and Frank Hutter. "Tabpfn: A transformer that solves small tabular classification problems in a second." *arXiv preprint* arXiv:2207.01848 (2022).

Müller, Samuel, Noah Hollmann, Sebastian Pineda Arango, Josif Grabocka, and Frank Hutter. "Transformers can do Bayesian inference." *arXiv preprint* arXiv:2112.10510 (2021).

See Also

`control_tab_pfn()`, `predict.tab_pfn()`

Examples

```
predictors <- mtcars[, -1]
outcome <- mtcars[, 1]

## Not run:
if (is_tab_pfn_installed() & interactive()) {
 # XY interface
 mod <- tab_pfn(predictors, outcome)

 # Formula interface
 mod2 <- tab_pfn(mpg ~ ., mtcars)

 # Recipes interface
 if (rlang::is_installed("recipes")) {
```

```
  suppressPackageStartupMessages(library(recipes))
  rec <-
   recipe(mpg ~ ., mtcars) %>%
   step_log(disp)

  mod3 <- tab_pfn(rec, mtcars)
  mod3
 }
}

## End(Not run)
```

Index