Package: prophet (via r-universe)

October 29, 2026

Title Automatic Forecasting Procedure

Version 1.1.7

Date 2026-01-22

Description Implements a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. It works best with time series that have strong seasonal effects and several seasons of historical data. Prophet is robust to missing data and shifts in the trend, and typically handles outliers well.

URL https://github.com/facebook/prophet

BugReports https://github.com/facebook/prophet/issues

Depends R (>= 3.4.0), Rcpp (>= 0.12.0), rlang (>= 0.3.0.1)

Imports dplyr (>= 0.7.7), dygraphs (>= 1.1.1.4), extraDistr, ggplot2, grid, lubridate, methods, RcppParallel (>= 5.0.1), rstan (>= 2.18.1), rstantools (>= 2.0.0), scales, StanHeaders, stats, tidyr (>= 0.6.1), xts

Suggests cmdstanr, posterior, knitr, testthat, readr, rmarkdown

Additional_repositories https://stan-dev.r-universe.dev

SystemRequirements GNU make, C++17

Biarch true

License MIT + file LICENSE

LinkingTo BH (>= 1.66.0), Rcpp (>= 0.12.0), RcppParallel (>= 5.0.1), RcppEigen (>= 0.3.3.3.0), rstan (>= 2.18.1), StanHeaders (>= 2.18.0)

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.2.0

NeedsCompilation yes

Author Cuong Duong [cre, aut], Sean Taylor [aut], Ben Letham [aut]

Maintainer Cuong Duong <cuong.duong242@gmail.com>

Config/pak/sysreqs make

Repository https://cran.r-universe.dev

Date/Publication 2026-01-22 08:53:08 UTC

RemoteUrl https://github.com/cran/prophet

RemoteRef HEAD

RemoteSha 323cbc0f1dd442b5457316c14346e3652dea6758

Contents

add_changepoints_to_plot . . . 2
add_country_holidays . . . 3
add_regressor . . . 4
add_seasonality . . . 5
cross_validation . . . 6
dyplot.prophet . . . 7
fit.prophet . . . 8
generated_holidays . . . 8
make_future_dataframe . . . 9
performance_metrics . . . 9
plot.prophet . . . 10
plot_cross_validation_metric . . . 11
plot_forecast_component . . . 12
predict.prophet . . . 13
predictive_samples . . . 13
prophet . . . 14
prophet_plot_components . . . 16
regressor_coefficients . . . 17
rolling_median_by_h . . . 18

Index **19**

`add_changepoints_to_plot`

Get layers to overlay significant changepoints on prophet forecast plot.

Description

Get layers to overlay significant changepoints on prophet forecast plot.

Usage

```
add_changepoints_to_plot(
  m,
  threshold = 0.01,
  cp_color = "red",
  cp_linetype = "dashed",
  trend = TRUE,
  ...
)
```

Arguments

m	Prophet model object.
threshold	Numeric, changepoints where abs(delta) >= threshold are significant. (Default 0.01)
cp_color	Character, line color. (Default "red")
cp_linetype	Character or integer, line type. (Default "dashed")
trend	Logical, if FALSE, do not draw trend line. (Default TRUE)
...	Other arguments passed on to layers.

Value

A list of ggplot2 layers.

Examples

```
## Not run:
plot(m, fcst) + add_changepoints_to_plot(m)

## End(Not run)
```

---

add_country_holidays *Add in built-in holidays for the specified country.*

Description

These holidays will be included in addition to any specified on model initialization.

Usage

```
add_country_holidays(m, country_name)
```

Arguments

m	Prophet object.
country_name	Name of the country, like 'UnitedStates' or 'US'

Details

Holidays will be calculated for arbitrary date ranges in the history and future. See the online documentation for the list of countries with built-in holidays.

Built-in country holidays can only be set for a single country.

Value

The prophet model with the holidays country set.

`add_regressor` *Add an additional regressor to be used for fitting and predicting.*

Description

The dataframe passed to ‘fit‘ and ‘predict‘ will have a column with the specified name to be used as a regressor. When standardize='auto', the regressor will be standardized unless it is binary. The regression coefficient is given a prior with the specified scale parameter. Decreasing the prior scale will add additional regularization. If no prior scale is provided, holidays.prior.scale will be used. Mode can be specified as either 'additive' or 'multiplicative'. If not specified, m$seasonality.mode will be used. 'additive' means the effect of the regressor will be added to the trend, 'multiplicative' means it will multiply the trend.

Usage

```
add_regressor(m, name, prior.scale = NULL, standardize = "auto", mode = NULL)
```

Arguments

`m`	Prophet object.
`name`	String name of the regressor
`prior.scale`	Float scale for the normal prior. If not provided, holidays.prior.scale will be used.
`standardize`	Bool, specify whether this regressor will be standardized prior to fitting. Can be 'auto' (standardize if not binary), True, or False.
`mode`	Optional, 'additive' or 'multiplicative'. Defaults to m$seasonality.mode.

Value

The prophet model with the regressor added.

`add_seasonality` *Add a seasonal component with specified period, number of Fourier components, and prior scale.*

Description

Increasing the number of Fourier components allows the seasonality to change more quickly (at risk of overfitting). Default values for yearly and weekly seasonalities are 10 and 3 respectively.

Usage

```
add_seasonality(
  m,
  name,
  period,
  fourier.order,
  prior.scale = NULL,
  mode = NULL,
  condition.name = NULL
)
```

Arguments

`m`	Prophet object.
`name`	String name of the seasonality component.
`period`	Float number of days in one period.
`fourier.order`	Int number of Fourier components to use.
`prior.scale`	Optional float prior scale for this component.
`mode`	Optional 'additive' or 'multiplicative'.
`condition.name`	String name of the seasonality condition.

Details

Increasing prior scale will allow this seasonality component more flexibility, decreasing will dampen it. If not provided, will use the seasonality.prior.scale provided on Prophet initialization (defaults to 10).

Mode can be specified as either 'additive' or 'multiplicative'. If not specified, m$seasonality.mode will be used (defaults to 'additive'). Additive means the seasonality will be added to the trend, multiplicative means it will multiply the trend.

If condition.name is provided, the dataframe passed to 'fit' and 'predict' should have a column with the specified condition.name containing booleans which decides when to apply seasonality.

Value

The prophet model with the seasonality added.

`cross_validation` *Cross-validation for time series.*

Description

Computes forecasts from historical cutoff points which user can input.If not provided, these are computed beginning from (end - horizon), and working backwards making cutoffs with a spacing of period until initial is reached.

Usage

```
cross_validation(
  model,
  horizon,
  units,
  period = NULL,
  initial = NULL,
  cutoffs = NULL
)
```

Arguments

`model`	Fitted Prophet model.
`horizon`	Integer size of the horizon
`units`	String unit of the horizon, e.g., "days", "secs".
`period`	Integer amount of time between cutoff dates. Same units as horizon. If not provided, 0.5 * horizon is used.
`initial`	Integer size of the first training period. If not provided, 3 * horizon is used. Same units as horizon.
`cutoffs`	Vector of cutoff dates to be used during cross-validtation. If not provided works beginning from (end - horizon), works backwards making cutoffs with a spacing of period until initial is reached.

Details

When period is equal to the time interval of the data, this is the technique described in https://robjhyndman.com/hyndsight/tscv
.

Value

A dataframe with the forecast, actual value, and cutoff date.

`dyplot.prophet` *Plot the prophet forecast.*

Description

Plot the prophet forecast.

Usage

```
dyplot.prophet(x, fcst, uncertainty = TRUE, ...)
```

Arguments

`x`	Prophet object.
`fcst`	Data frame returned by predict(m, df).
`uncertainty`	Optional boolean indicating if the uncertainty interval for yhat should be plotted, which will only be done if x$uncertainty.samples > 0. Must be present in fcst as yhat_lower and yhat_upper.
`...`	additional arguments passed to dygraphs::dygraph

Value

A dygraph plot.

Examples

```
## Not run:
history <- data.frame(
 ds = seq(as.Date('2015-01-01'), as.Date('2016-01-01'), by = 'd'),
 y = sin(1:366/200) + rnorm(366)/10)
m <- prophet(history)
future <- make_future_dataframe(m, periods = 365)
forecast <- predict(m, future)
dyplot.prophet(m, forecast)

## End(Not run)
```

fit.prophet *Fit the prophet model.*

Description

This sets m$params to contain the fitted model parameters. It is a list with the following elements: k (M array): M posterior samples of the initial slope. m (M array): The initial intercept. delta (MxN matrix): The slope change at each of N changepoints. beta (MxK matrix): Coefficients for K seasonality features. sigma_obs (M array): Noise level. Note that M=1 if MAP estimation.

Usage

```
fit.prophet(m, df, ...)
```

Arguments

m	Prophet object.
df	Data frame.
...	Additional arguments passed to the optimizing or sampling functions in Stan.

generated_holidays *Generated table of holiday dates at the country level from 1995 to 2045*

Description

The data is primarily based on the Python package [holidays](https://pypi.org/project/holidays/)

Usage

```
generated_holidays
```

Format

A data frame with four variables: ds, holiday, country, year

Source

https://github.com/facebook/prophet/blob/main/python/scripts/generate_holidays_file.py

make_future_dataframe *Make dataframe with future dates for forecasting.*

Description

Make dataframe with future dates for forecasting.

Usage

```
make_future_dataframe(m, periods, freq = "day", include_history = TRUE)
```

Arguments

m	Prophet model object.
periods	Int number of periods to forecast forward.
freq	'day', 'week', 'month', 'quarter', 'year', 1(1 sec), 60(1 minute) or 3600(1 hour).
include_history	Boolean to include the historical dates in the data frame for predictions.

Value

Dataframe that extends forward from the end of m$history for the requested number of periods.

performance_metrics *Compute performance metrics from cross-validation results.*

Description

Computes a suite of performance metrics on the output of cross-validation. By default the following metrics are included: 'mse': mean squared error, 'rmse': root mean squared error, 'mae': mean absolute error, 'mape': mean percent error, 'mdape': median percent error, 'smape': symmetric mean absolute percentage error, 'coverage': coverage of the upper and lower intervals

Usage

```
performance_metrics(df, metrics = NULL, rolling_window = 0.1)
```

Arguments

df	The dataframe returned by cross_validation.
metrics	An array of performance metrics to compute. If not provided, will use c('mse', 'rmse', 'mae', 'mape', 'mdape', 'smape', 'coverage').
rolling_window	Proportion of data to use in each rolling window for computing the metrics. Should be in [0, 1] to average.

Details

A subset of these can be specified by passing a list of names as the ‘metrics‘ argument.

Metrics are calculated over a rolling window of cross validation predictions, after sorting by horizon. Averaging is first done within each value of the horizon, and then across horizons as needed to reach the window size. The size of that window (number of simulated forecast points) is determined by the rolling_window argument, which specifies a proportion of simulated forecast points to include in each window. rolling_window=0 will compute it separately for each horizon. The default of rolling_window=0.1 will use 10 rolling_window=1 will compute the metric across all simulated forecast points. The results are set to the right edge of the window.

If rolling_window < 0, then metrics are computed at each datapoint with no averaging (i.e., 'mse' will actually be squared error with no mean).

The output is a dataframe containing column 'horizon' along with columns for each of the metrics computed.

Value

A dataframe with a column for each metric, and column 'horizon'.

`plot.prophet` *Plot the prophet forecast.*

Description

Plot the prophet forecast.

Usage

```
## S3 method for class 'prophet'
plot(
  x,
  fcst,
  uncertainty = TRUE,
  plot_cap = TRUE,
  xlabel = "ds",
  ylabel = "y",
  ...
)
```

Arguments

`x`	Prophet object.
`fcst`	Data frame returned by predict(m, df).
`uncertainty`	Optional boolean indicating if the uncertainty interval for yhat should be plotted, which will only be done if x$uncertainty.samples > 0. Must be present in fcst as yhat_lower and yhat_upper.

plot_cap	Boolean indicating if the capacity should be shown in the figure, if available.
xlabel	Optional label for x-axis
ylabel	Optional label for y-axis
...	additional arguments

Value

A ggplot2 plot.

Examples

```
## Not run:
history <- data.frame(ds = seq(as.Date('2015-01-01'), as.Date('2016-01-01'), by = 'd'),
                      y = sin(1:366/200) + rnorm(366)/10)
m <- prophet(history)
future <- make_future_dataframe(m, periods = 365)
forecast <- predict(m, future)
plot(m, forecast)

## End(Not run)
```

---

plot_cross_validation_metric

Plot a performance metric vs. forecast horizon from cross validation. Cross validation produces a collection of out-of-sample model predictions that can be compared to actual values, at a range of different horizons (distance from the cutoff). This computes a specified performance metric for each prediction, and aggregated over a rolling window with horizon.

Description

This uses fbprophet.diagnostics.performance_metrics to compute the metrics. Valid values of metric are 'mse', 'rmse', 'mae', 'mape', and 'coverage'.

Usage

```
plot_cross_validation_metric(df_cv, metric, rolling_window = 0.1)
```

Arguments

df_cv	The output from fbprophet.diagnostics.cross_validation.
metric	Metric name, one of 'mse', 'rmse', 'mae', 'mape', 'coverage'.
rolling_window	Proportion of data to use for rolling average of metric. In [0, 1]. Defaults to 0.1.

Details

rolling_window is the proportion of data included in the rolling window of aggregation. The default value of 0.1 means 10 aggregation for computing the metric.

As a concrete example, if metric='mse', then this plot will show the squared error for each cross validation prediction, along with the MSE averaged over rolling windows of 10

Value

A ggplot2 plot.

`plot_forecast_component`

Plot a particular component of the forecast.

Description

Plot a particular component of the forecast.

Usage

```
plot_forecast_component(m, fcst, name, uncertainty = TRUE, plot_cap = FALSE)
```

Arguments

`m`	Prophet model
`fcst`	Dataframe output of 'predict'.
`name`	String name of the component to plot (column of fcst).
`uncertainty`	Optional boolean to plot uncertainty intervals, which will only be done if m$uncertainty.samples > 0.
`plot_cap`	Boolean indicating if the capacity should be shown in the figure, if available.

Value

A ggplot2 plot.

predict.prophet *Predict using the prophet model.*

Description

Predict using the prophet model.

Usage

```
## S3 method for class 'prophet'
predict(object, df = NULL, ...)
```

Arguments

object	Prophet object.
df	Dataframe with dates for predictions (column ds), and capacity (column cap) if logistic growth. If not provided, predictions are made on the history.
...	additional arguments.

Value

A dataframe with the forecast components.

Examples

```
## Not run:
history <- data.frame(ds = seq(as.Date('2015-01-01'), as.Date('2016-01-01'), by = 'd'),
                      y = sin(1:366/200) + rnorm(366)/10)
m <- prophet(history)
future <- make_future_dataframe(m, periods = 365)
forecast <- predict(m, future)
plot(m, forecast)

## End(Not run)
```

predictive_samples *Sample from the posterior predictive distribution.*

Description

Sample from the posterior predictive distribution.

Usage

```
predictive_samples(m, df)
```

Arguments

m	Prophet object.
df	Dataframe with dates for predictions (column ds), and capacity (column cap) if logistic growth.

Value

A list with items "trend" and "yhat" containing posterior predictive samples for that component.

---

prophet *Prophet forecaster.*

Description

Prophet forecaster.

Usage

```
prophet(
  df = NULL,
  growth = "linear",
  changepoints = NULL,
  n.changepoints = 25,
  changepoint.range = 0.8,
  yearly.seasonality = "auto",
  weekly.seasonality = "auto",
  daily.seasonality = "auto",
  holidays = NULL,
  seasonality.mode = "additive",
  seasonality.prior.scale = 10,
  holidays.prior.scale = 10,
  changepoint.prior.scale = 0.05,
  mcmc.samples = 0,
  interval.width = 0.8,
  uncertainty.samples = 1000,
  fit = TRUE,
  backend = NULL,
  ...
)
```

Arguments

df	(optional) Dataframe containing the history. Must have columns ds (date type) and y, the time series. If growth is logistic, then df must also have a column cap that specifies the capacity at each ds. If not provided, then the model object will be instantiated but not fit; use fit.prophet(m, df) to fit the model.

`growth` String 'linear', 'logistic', or 'flat' to specify a linear, logistic or flat trend.

`changepoints` Vector of dates at which to include potential changepoints. If not specified, potential changepoints are selected automatically.

`n.changepoints` Number of potential changepoints to include. Not used if input 'changepoints' is supplied. If 'changepoints' is not supplied, then n.changepoints potential changepoints are selected uniformly from the first 'changepoint.range' proportion of df$ds.

`changepoint.range`
Proportion of history in which trend changepoints will be estimated. Defaults to 0.8 for the first 80 'changepoints' is specified.

`yearly.seasonality`
Fit yearly seasonality. Can be 'auto', TRUE, FALSE, or a number of Fourier terms to generate.

`weekly.seasonality`
Fit weekly seasonality. Can be 'auto', TRUE, FALSE, or a number of Fourier terms to generate.

`daily.seasonality`
Fit daily seasonality. Can be 'auto', TRUE, FALSE, or a number of Fourier terms to generate.

`holidays` data frame with columns holiday (character) and ds (date type)and optionally columns lower_window and upper_window which specify a range of days around the date to be included as holidays. lower_window=-2 will include 2 days prior to the date as holidays. Also optionally can have a column prior_scale specifying the prior scale for each holiday.

`seasonality.mode`
'additive' (default) or 'multiplicative'.

`seasonality.prior.scale`
Parameter modulating the strength of the seasonality model. Larger values allow the model to fit larger seasonal fluctuations, smaller values dampen the seasonality. Can be specified for individual seasonalities using add_seasonality.

`holidays.prior.scale`
Parameter modulating the strength of the holiday components model, unless overridden in the holidays input.

`changepoint.prior.scale`
Parameter modulating the flexibility of the automatic changepoint selection. Large values will allow many changepoints, small values will allow few changepoints.

`mcmc.samples` Integer, if greater than 0, will do full Bayesian inference with the specified number of MCMC samples. If 0, will do MAP estimation.

`interval.width` Numeric, width of the uncertainty intervals provided for the forecast. If mcmc.samples=0, this will be only the uncertainty in the trend using the MAP estimate of the extrapolated generative model. If mcmc.samples>0, this will be integrated over all model parameters, which will include uncertainty in seasonality.

`uncertainty.samples`
Number of simulated draws used to estimate uncertainty intervals. Settings this value to 0 or False will disable uncertainty estimation and speed up the calculation.

fit	Boolean, if FALSE the model is initialized but not fit.
backend	Whether to use the "rstan" or "cmdstanr" backend to fit the model. If not provided, uses the R_STAN_BACKEND environment variable.
...	Additional arguments, passed to fit.prophet

Value

A prophet model.

Examples

```
## Not run:
history <- data.frame(ds = seq(as.Date('2015-01-01'), as.Date('2016-01-01'), by = 'd'),
                      y = sin(1:366/200) + rnorm(366)/10)
m <- prophet(history)

## End(Not run)
```

---

prophet_plot_components

Plot the components of a prophet forecast. Prints a ggplot2 with whichever are available of: trend, holidays, weekly seasonality, yearly seasonality, and additive and multiplicative extra regressors.

Description

Plot the components of a prophet forecast. Prints a ggplot2 with whichever are available of: trend, holidays, weekly seasonality, yearly seasonality, and additive and multiplicative extra regressors.

Usage

```
prophet_plot_components(
  m,
  fcst,
  uncertainty = TRUE,
  plot_cap = TRUE,
  weekly_start = 0,
  yearly_start = 0,
  render_plot = TRUE
)
```

Arguments

m	Prophet object.
fcst	Data frame returned by predict(m, df).

uncertainty	Optional boolean indicating if the uncertainty interval should be plotted for the trend, from fcst columns trend_lower and trend_upper.This will only be done if m$uncertainty.samples > 0.
plot_cap	Boolean indicating if the capacity should be shown in the figure, if available.
weekly_start	Integer specifying the start day of the weekly seasonality plot. 0 (default) starts the week on Sunday. 1 shifts by 1 day to Monday, and so on.
yearly_start	Integer specifying the start day of the yearly seasonality plot. 0 (default) starts the year on Jan 1. 1 shifts by 1 day to Jan 2, and so on.
render_plot	Boolean indicating if the plots should be rendered. Set to FALSE if you want the function to only return the list of panels.

Value

Invisibly return a list containing the plotted ggplot objects

---

`regressor_coefficients`

Summarise the coefficients of the extra regressors used in the model. For additive regressors, the coefficient represents the incremental impact on `y` *of a unit increase in the regressor. For multiplicative regressors, the incremental impact is equal to* `trend(t)` *multiplied by the coefficient.*

Description

Coefficients are measured on the original scale of the training data.

Usage

```
regressor_coefficients(m)
```

Arguments

m	Prophet model object, after fitting.

Details

Output dataframe columns:

- regressor: Name of the regressor
- regressor_mode: Whether the regressor has an additive or multiplicative effect on `y`.
- center: The mean of the regressor if it was standardized. Otherwise 0.
- coef_lower: Lower bound for the coefficient, estimated from the MCMC samples. Only different to `coef` if `mcmc_samples > 0`.
- coef: Expected value of the coefficient.
- coef_upper: Upper bound for the coefficient, estimated from MCMC samples. Only to different to `coef` if `mcmc_samples > 0`.

Value

Dataframe with one row per regressor.

`rolling_median_by_h` *Compute a rolling median of x, after first aggregating by h*

Description

Right-aligned. Computes a single median for each unique value of h. Each median is over at least w samples.

Usage

```
rolling_median_by_h(x, h, w, name)
```

Arguments

`x`	Array.
`h`	Array of horizon for each value in x.
`w`	Integer window size (number of elements).
`name`	String name for metric in result dataframe.

Details

For each h where there are fewer than w samples, we take samples from the previous h,

Value

Dataframe with columns horizon and name, the rolling median of x.

Index

∗ **datasets**
generated_holidays, 8

add_changepoints_to_plot, 2
add_country_holidays, 3
add_regressor, 4
add_seasonality, 5

cross_validation, 6

dyplot.prophet, 7

fit.prophet, 8, *16*

generated_holidays, 8

make_future_dataframe, 9

performance_metrics, 9
plot.prophet, 10
plot_cross_validation_metric, 11
plot_forecast_component, 12
predict.prophet, 13
predictive_samples, 13
prophet, 14
prophet_plot_components, 16

regressor_coefficients, 17
rolling_median_by_h, 18