

# Package ‘prophet’

April 29, 2020

**Title** Automatic Forecasting Procedure

**Version** 0.6.1

**Date** 2020-04-28

**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, methods, rstan (>= 2.18.1), rstantools (>= 2.0.0), scales, stats, tidyr (>= 0.6.1), xts

**Suggests** knitr, testthat, readr, rmarkdown

**SystemRequirements** GNU make, C++11

**Biarch** true

**License** MIT + file LICENSE

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

**LazyData** true

**VignetteBuilder** knitr

**Encoding** UTF-8

**RoxygenNote** 7.1.0

**NeedsCompilation** yes

**Author** Sean Taylor [cre, aut],  
Ben Letham [aut]

**Maintainer** Sean Taylor <sjtz@pm.me>

**Repository** CRAN

**Date/Publication** 2020-04-29 17:30:02 UTC

**R topics documented:**

add_changepoints_to_plot . . . . .	2
add_country_holidays . . . . .	3
add_regressor . . . . .	4
add_seasonality . . . . .	4
cross_validation . . . . .	5
dyplot.prophet . . . . .	6
fit.prophet . . . . .	7
generated_holidays . . . . .	7
make_future_dataframe . . . . .	8
performance_metrics . . . . .	8
plot.prophet . . . . .	9
plot_cross_validation_metric . . . . .	10
plot_forecast_component . . . . .	11
predict.prophet . . . . .	12
predictive_samples . . . . .	12
prophet . . . . .	13
prophet_plot_components . . . . .	15
<b>Index</b>	<b>17</b>

---

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 $\text{abs}(\text{delta}) \geq \text{threshold}$ 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	<i>Add an additional regressor to be used for fitting and predicting.</i>
---------------	---

---

### 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	<i>Add a seasonal component with specified period, number of Fourier components, and prior scale.</i>
-----------------	---

---

### 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**

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

---

<code>cross_validation</code>	<i>Cross-validation for time series.</i>
-------------------------------	--

---

**Description**

Computes forecasts from historical cutoff points. Beginning from (end - horizon), works backwards making cutoffs with a spacing of period until initial is reached.

**Usage**

```
cross_validation(model, horizon, units, period = NULL, initial = 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.

**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	<i>Plot the prophet forecast.</i>
----------------	-----------------------------------

---

**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 dygraph::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	<i>Fit the prophet model.</i>
-------------	-------------------------------

---

**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**

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

---

generated_holidays	<i>holidays table</i>
--------------------	-----------------------

---

**Description**

holidays table

**Usage**

```
generated_holidays
```

**Format**

A data frame with five variables: `ds`, `holiday`, `country`, `year`

---

`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

<code>m</code>	Prophet model object.
<code>periods</code>	Int number of periods to forecast forward.
<code>freq</code>	'day', 'week', 'month', 'quarter', 'year', 1(1 sec), 60(1 minute) or 3600(1 hour).
<code>include_history</code>	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 'coverage': coverage of the upper and lower intervals

### Usage

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

### Arguments

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

---

<code>plot.prophet</code>	<i>Plot the prophet forecast.</i>
---------------------------	-----------------------------------

---

## Description

Plot the prophet forecast.

## Usage

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

## Arguments

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

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 <code>fbprophet.diagnostics.cross_validation</code> .
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	<i>Predict using the prophet model.</i>
-----------------	---

---

**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	<i>Sample from the posterior predictive distribution.</i>
--------------------	---

---

**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	<i>Prophet forecaster.</i>
---------	----------------------------

---

**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,
  ...
)
```

**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 <code>fit.prophet(m, df)</code> to fit the model.
growth	String 'linear' or 'logistic' to specify a linear or logistic trend.

<code>changepoints</code>	Vector of dates at which to include potential changepoints. If not specified, potential changepoints are selected automatically.
<code>n.changepoints</code>	Number of potential changepoints to include. Not used if input 'changepoints' is supplied. If 'changepoints' is not supplied, then <code>n.changepoints</code> potential changepoints are selected uniformly from the first 'changepoint.range' proportion of <code>df\$ds</code> .
<code>changepoint.range</code>	Proportion of history in which trend changepoints will be estimated. Defaults to 0.8 for the first 80 'changepoints' is specified.
<code>yearly.seasonality</code>	Fit yearly seasonality. Can be 'auto', TRUE, FALSE, or a number of Fourier terms to generate.
<code>weekly.seasonality</code>	Fit weekly seasonality. Can be 'auto', TRUE, FALSE, or a number of Fourier terms to generate.
<code>daily.seasonality</code>	Fit daily seasonality. Can be 'auto', TRUE, FALSE, or a number of Fourier terms to generate.
<code>holidays</code>	data frame with columns <code>holiday</code> (character) and <code>ds</code> (date type) and optionally columns <code>lower_window</code> and <code>upper_window</code> which specify a range of days around the date to be included as holidays. <code>lower_window=-2</code> will include 2 days prior to the date as holidays. Also optionally can have a column <code>prior_scale</code> specifying the prior scale for each holiday.
<code>seasonality.mode</code>	'additive' (default) or 'multiplicative'.
<code>seasonality.prior.scale</code>	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 <code>add_seasonality</code> .
<code>holidays.prior.scale</code>	Parameter modulating the strength of the holiday components model, unless overridden in the <code>holidays</code> input.
<code>changepoint.prior.scale</code>	Parameter modulating the flexibility of the automatic changepoint selection. Large values will allow many changepoints, small values will allow few changepoints.
<code>mcmc.samples</code>	Integer, if greater than 0, will do full Bayesian inference with the specified number of MCMC samples. If 0, will do MAP estimation.
<code>interval.width</code>	Numeric, width of the uncertainty intervals provided for the forecast. If <code>mcmc.samples=0</code> , this will be only the uncertainty in the trend using the MAP estimate of the extrapolated generative model. If <code>mcmc.samples&gt;0</code> , this will be integrated over all model parameters, which will include uncertainty in seasonality.
<code>uncertainty.samples</code>	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.  
`...` 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`.

<code>plot_cap</code>	Boolean indicating if the capacity should be shown in the figure, if available.
<code>weekly_start</code>	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.
<code>yearly_start</code>	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.
<code>render_plot</code>	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

# Index

## \*Topic **datasets**

- [generated\\_holidays](#), 7
- [add\\_changepoints\\_to\\_plot](#), 2
- [add\\_country\\_holidays](#), 3
- [add\\_regressor](#), 4
- [add\\_seasonality](#), 4
  
- [cross\\_validation](#), 5
  
- [dyplot.prophet](#), 6
  
- [fit.prophet](#), 7, 15
  
- [generated\\_holidays](#), 7
  
- [make\\_future\\_dataframe](#), 8
  
- [performance\\_metrics](#), 8
- [plot.prophet](#), 9
- [plot\\_cross\\_validation\\_metric](#), 10
- [plot\\_forecast\\_component](#), 11
- [predict.prophet](#), 12
- [predictive\\_samples](#), 12
- [prophet](#), 13
- [prophet\\_plot\\_components](#), 15