Package 'bayesmodels'

June 28, 2021

Title The 'Tidymodels' Extension for Bayesian Models

Version 0.1.1

Description Bayesian framework for use with the 'tidymodels' ecosystem. Includes the following models: Sarima, Garch,
Random walk (naive), Additive Linear State Space Models, Stochastic Volatility Models from 'bayesforecast' package,
Adaptive Splines Surfaces from 'BASS' package and ETS from 'Rlgt' package.

URL https://github.com/AlbertoAlmuinha/bayesmodels

BugReports https://github.com/AlbertoAlmuinha/bayesmodels/issues

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 4.0.0), parsnip, bayesforecast, bsts

Imports rlang (>= 0.1.2), brms, BASS, Rlgt, rstan, magrittr, purrr,
dplyr, tibble, dials, workflows, modeltime, timetk, cli,
crayon, rstudioapi

Suggests tidymodels, tidyverse, lubridate, knitr, rmarkdown, roxygen2,
testthat (>= 3.0.0), covr

RoxygenNote 7.1.1

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Alberto Almuiña [aut, cre]

Maintainer Alberto Almuiña <albertogonzalezalmuinha@gmail.com>

Repository CRAN

Date/Publication 2021-06-28 21:30:02 UTC

R topics documented:

`adaptive_spline` *General Interface for Adaptive Spline Surface Models*

Description

`adaptive_spline()` is a way to generate a *specification* of an Adaptive Spline Surface model before fitting and allows the model to be created using different packages. Currently the only package is `BASS`.

Usage

```
adaptive_spline(
  mode = "regression",
  splines_degree = NULL,
  max_degree = NULL,
  max_categorical_degree = NULL,
  min_basis_points = NULL
)
```

Arguments

mode	A single character string for the type of model. The only possible value for this model is "regression".
splines_degree	degree of splines. Stability should be examined for anything other than 1.
max_degree	integer for maximum degree of interaction in spline basis functions. Defaults to the number of predictors, which could result in overfitting.
max_categorical_degree	(categorical input only) integer for maximum degree of interaction of categorical inputs.
min_basis_points	minimum number of non-zero points in a basis function. If the response is functional, this refers only to the portion of the basis function coming from the non-functional predictors. Defaults to 20 or 0.1 times the number of observations, whichever is smaller.

Details

The data given to the function are not saved and are only used to determine the *mode* of the model. For `adaptive_spline()`, the mode will always be "regression".

The model can be created using the `fit()` function using the following *engines*:

- "stan" (default) - Connects to `BASS::bass()`

Main Arguments

The main arguments (tuning parameters) for the model are:

- `splines_degree`
- `max_degree`
- `max_categorical_degree`
- `min_basis_points`

These arguments are converted to their specific names at the time that the model is fit.

Other options and argument can be set using `set_engine()` (See Engine Details below).

If parameters need to be modified, `update()` can be used in lieu of recreating the object from scratch.

Value

A model spec

Engine Details

Other options can be set using `set_engine()`.

stan (default engine)

The engine uses `BASS::bass()`.

Parameter Notes:

- `xreg` - This is supplied via the parsnip / bayesmodels `fit()` interface (so don't provide this manually). See Fit Details (below).

Fit Details

Date and Date-Time Variable

It's a requirement to have a date or date-time variable as a predictor. The `fit()` interface accepts date and date-time features and handles them internally.

- `fit(y ~ date)`

Univariate (No xregs, Exogenous Regressors):

This algorithm only accepts multivariate: you need to pass xregs (read next section).

Multivariate (xregs, Exogenous Regressors)

The `xreg` parameter is populated using the `fit()` function:

- Only `factor`, ordered factor, and `numeric` data will be used as xregs.
- Date and Date-time variables are not used as xregs
- `character` data should be converted to factor.

Xreg Example: Suppose you have 3 features:

1. `y` (target)
2. `date` (time stamp),
3. `month.lbl` (labeled month as a ordered factor).

The `month.lbl` is an exogenous regressor that can be passed to the `sarima_reg()` using `fit()`:

- `fit(y ~ date + month.lbl)` will pass `month.lbl` on as an exogenous regressor.

Note that date or date-time class values are excluded from `xreg`.

See Also

`fit.model_spec()`, `set_engine()`

Examples

```
## Not run:
library(dplyr)
library(parsnip)
library(rsample)
library(timetk)
library(modeltime)
library(bayesmodels)
library(lubridate)

# Data
m750 <- m4_monthly %>% filter(id == "M750")
m750

# Split Data 80/20
splits <- rsample::initial_time_split(m750, prop = 0.8)

# ---- Adaptive Spline ----

# Model Spec
model_spec <- adaptive_spline() %>%
    set_engine("stan")

# Fit Spec
model_fit <- model_spec %>%
    fit(log(value) ~ date + month(date), data = training(splits))
model_fit

## End(Not run)
```

---

adaptive_splines_params

Tuning Parameters for Adaptive Splines Surface Models

Description

Tuning Parameters for Adaptive Splines Surface Models

Usage

```
splines_degree(range = c(0L, 5L), trans = NULL)

max_degree(range = c(0L, 5L), trans = NULL)

max_categorical_degree(range = c(0L, 5L), trans = NULL)

min_basis_points(range = c(0L, 1000L), trans = NULL)
```

Arguments

range	A two-element vector holding the *defaults* for the smallest and largest possible values, respectively.
trans	A trans object from the scales package, such as scales::log10_trans() or scales::reciprocal_trans(). If not provided, the default is used which matches the units used in range. If no transformation, NULL.

Details

The main parameters for Adaptive Splines Surface models are:

- splines_degree: degree of splines. Stability should be examined for anything other than 1.
- max_degree: integer for maximum degree of interaction in spline basis functions.
- max_categorical_degree: (categorical input only) integer for maximum degree of interaction of categorical inputs.
- min_basis_points: minimum number of non-zero points in a basis function

Value

A parameter

A parameter

A parameter

A parameter

Examples

```
splines_degree()

max_degree()

min_basis_points()
```

---

adaptive_spline_stan_fit_impl

Low-Level ARIMA function for translating modeltime to forecast

---

Description

Low-Level ARIMA function for translating modeltime to forecast

Usage

```
adaptive_spline_stan_fit_impl(
  x,
  y,
  degree = 1,
  maxInt = 3,
  maxInt.cat = 3,
  npart = NULL,
  ...
)
```

Arguments

x	A dataframe of xreg (exogenous regressors)
y	A numeric vector of values to fit
degree	degree of splines
maxInt	integer for maximum degree of interaction in spline basis functions
maxInt.cat	(categorical input only) integer for maximum degree of interaction of categorical inputs
npart	minimum number of non-zero points in a basis function
...	Extra arguments

Value

A modeltime model

---

adaptive_spline_stan_predict_impl

Bridge prediction function for ARIMA models

Description

Bridge prediction function for ARIMA models

Usage

```
adaptive_spline_stan_predict_impl(object, new_data, ...)
```

Arguments

object	An object of class model_fit
new_data	A rectangular data object, such as a data frame.
...	Additional arguments passed to forecast::Arima()

Value

A prediction

`additive_state_space` *General Interface for Additive Linear State Space Regression Models*

Description

`additive_state_space()` is a way to generate a *specification* of a Additive Linear State Space Regression Model before fitting and allows the model to be created using different packages. Currently the only package is `bayesforecast`.

Usage

```
additive_state_space(
  mode = "regression",
  trend_model = NULL,
  damped_model = NULL,
  seasonal_model = NULL,
  seasonal_period = NULL,
  garch_t_student = NULL,
  markov_chains = NULL,
  chain_iter = NULL,
  warmup_iter = NULL,
  adapt_delta = NULL,
  tree_depth = NULL,
  pred_seed = NULL
)
```

Arguments

`mode`	A single character string for the type of model. The only possible value for this model is "regression".
`trend_model`	a boolean value to specify a trend local level model. By default is FALSE.
`damped_model`	a boolean value to specify a damped trend local level model. By default is FALSE.
`seasonal_model`	a boolean value to specify a seasonal local level model. By default is FALSE.
`seasonal_period`	an integer specifying the periodicity of the time series by default the value frequency(ts) is used
`garch_t_student`	a boolean value to specify for a generalized t-student SSM model.
`markov_chains`	An integer of the number of Markov Chains chains to be run, by default 4 chains are run.
`chain_iter`	An integer of total iterations per chain including the warm-up, by default the number of iterations are 2000.

warmup_iter	A positive integer specifying number of warm-up (aka burn-in) iterations. This also specifies the number of iterations used for step-size adaptation, so warm-up samples should not be used for inference. The number of warmup should not be larger than iter and the default is iter/2.
adapt_delta	An optional real value between 0 and 1, the thin of the jumps in a HMC method. By default is 0.9
tree_depth	An integer of the maximum depth of the trees evaluated during each iteration. By default is 10.
pred_seed	An integer with the seed for using when predicting with the model.

Details

The data given to the function are not saved and are only used to determine the *mode* of the model. For `additive_state_space()`, the mode will always be "regression".

The model can be created using the `fit()` function using the following *engines*:

- "stan" (default) - Connects to `bayesforecast::stan_ssm()`

Main Arguments

The main arguments (tuning parameters) for the model are:

- `trend_model`: a boolean value to specify a trend local level model. By default is FALSE.
- `damped_model`: a boolean value to specify a damped trend local level model. By default is FALSE.
- `seasonal_model`: a boolean value to specify a seasonal local level model. By default is FALSE.
- `markov_chains`: An integer of the number of Markov Chains chains to be run.
- `adapt_delta`: The thin of the jumps in a HMC method.
- `tree_depth`: The maximum depth of the trees evaluated during each iteration.

These arguments are converted to their specific names at the time that the model is fit.

Other options and argument can be set using `set_engine()` (See Engine Details below).

If parameters need to be modified, `update()` can be used in lieu of recreating the object from scratch.

Value

A model spec

Engine Details

The standardized parameter names in `bayesmodels` can be mapped to their original names in each engine:

bayesmodels	bayesforecast::stan_ssm
trend_model	trend
damped_model	damped

seasonal_model	seasonal
seasonal_period	period
markov_chains	chains(4)
adapt_delta	adapt.delta(0.9)
tree_depth	tree.depth(10)

Other options can be set using `set_engine()`.

stan (default engine)

The engine uses `bayesforecast::stan_ssm()`.

Parameter Notes:

- `xreg` - This is supplied via the parsnip / modeltime `fit()` interface (so don't provide this manually). See Fit Details (below).

Fit Details

Date and Date-Time Variable

It's a requirement to have a date or date-time variable as a predictor. The `fit()` interface accepts date and date-time features and handles them internally.

- `fit(y ~ date)`

Seasonal Period Specification

The period can be non-seasonal (seasonal_period = 1 or "none") or yearly seasonal (e.g. For monthly time stamps, `seasonal_period = 12`, `seasonal_period = "12 months"`, or `seasonal_period = "yearly"`). There are 3 ways to specify:

1. `seasonal_period = "auto"`: A seasonal period is selected based on the periodicity of the data (e.g. 12 if monthly)
2. `seasonal_period = 12`: A numeric frequency. For example, 12 is common for monthly data
3. `seasonal_period = "1 year"`: A time-based phrase. For example, "1 year" would convert to 12 for monthly data.

Univariate (No xregs, Exogenous Regressors):

For univariate analysis, you must include a date or date-time feature. Simply use:

- Formula Interface (recommended): `fit(y ~ date)` will ignore xreg's.

Multivariate (xregs, Exogenous Regressors)

The `xreg` parameter is populated using the `fit()` or `fit_xy()` function:

- Only `factor`, ordered factor, and `numeric` data will be used as xregs.
- Date and Date-time variables are not used as xregs
- `character` data should be converted to factor.

Xreg Example: Suppose you have 3 features:

1. y (target)
2. date (time stamp),
3. month.lbl (labeled month as a ordered factor).

The month.lbl is an exogenous regressor that can be passed to the arima_reg() using fit():

- fit(y ~ date + month.lbl) will pass month.lbl on as an exogenous regressor.

Note that date or date-time class values are excluded from xreg.

See Also

fit.model_spec(), set_engine()

Examples

```
## Not run:
library(dplyr)
library(parsnip)
library(rsample)
library(timetk)
library(modeltime)
library(bayesmodels)

# Data
m750 <- m4_monthly %>% filter(id == "M750")
m750

# Split Data 80/20
splits <- rsample::initial_time_split(m750, prop = 0.8)

# ---- AUTO ARIMA ----

# Model Spec
model_spec <- additive_state_space() %>%
    set_engine("stan")

# Fit Spec
model_fit <- model_spec %>%
    fit(log(value) ~ date, data = training(splits))
model_fit

predict(model_fit, testing(splits))


## End(Not run)
```

`bayesian_structural_reg` *General Interface for Bayesian Structural Time Series Models*

Description

`bayesian_structural_reg()` is a way to generate a *specification* of a Bayesian Structural Time Series Model before fitting and allows the model to be created using different packages. Currently the only package is `bsts`.

Usage

```
bayesian_structural_reg(mode = "regression", distribution = NULL)
```

Arguments

`mode`	A single character string for the type of model. The only possible value for this model is "regression".
`distribution`	The model family for the observation equation. Non-Gaussian model families use data augmentation to recover a conditionally Gaussian model.

Details

The data given to the function are not saved and are only used to determine the *mode* of the model. For `bayesian_structural_reg()`, the mode will always be "regression".

The model can be created using the `fit()` function using the following *engines*:

- "stan" (default) - Connects to `bsts::bsts()`

Main Arguments

Other options and argument can be set using `set_engine()` (See Engine Details below).

If parameters need to be modified, `update()` can be used in lieu of recreating the object from scratch.

stan (default engine)

The engine uses `bsts::bsts()`.

Parameter Notes:

- `xreg` - This is supplied via the parsnip / modeltime `fit()` interface (so don't provide this manually). See Fit Details (below).

Value

A model spec

Fit Details

Date and Date-Time Variable

It's a requirement to have a date or date-time variable as a predictor. The `fit()` interface accepts date and date-time features and handles them internally.

- `fit(y ~ date)`

Univariate (No xregs, Exogenous Regressors):

For univariate analysis, you must include a date or date-time feature. Simply use:

- Formula Interface (recommended): `fit(y ~ date)` will ignore xreg's.

Multivariate (xregs, Exogenous Regressors)

The `xreg` parameter is populated using the `fit()` or `fit_xy()` function:

- Only `factor`, ordered factor, and `numeric` data will be used as xregs.
- Date and Date-time variables are not used as xregs
- `character` data should be converted to factor.

Xreg Example: Suppose you have 3 features:

1. `y` (target)
2. `date` (time stamp),
3. `month.lbl` (labeled month as a ordered factor).

The `month.lbl` is an exogenous regressor that can be passed to the `arima_reg()` using `fit()`:

- `fit(y ~ date + month.lbl)` will pass `month.lbl` on as an exogenous regressor.

Note that date or date-time class values are excluded from `xreg`.

See Also

`fit.model_spec()`, `set_engine()`

Examples

```
## Not run:
library(dplyr)
library(parsnip)
library(rsample)
library(timetk)
library(modeltime)
library(bayesmodels)

# Data
m750 <- m4_monthly %>% filter(id == "M750")
m750

# Split Data 80/20
splits <- initial_time_split(m750, prop = 0.8)
```

```
ss <- AddLocalLinearTrend(list(), training(splits)$value)

# Model Spec
model_spec <- bayesian_structural_reg() %>%
    set_engine("stan", state.specification = ss)

# Fit Spec
model_fit <- model_spec %>%
    fit(log(value) ~ date, data = training(splits))
model_fit

predict(model_fit, testing(splits))


## End(Not run)
```

`bayesian_structural_stan_fit_impl` *Low-Level ARIMA function for translating modeltime to forecast*

Description

Low-Level ARIMA function for translating modeltime to forecast

Usage

```
bayesian_structural_stan_fit_impl(formula, data, family = "gaussian", ...)
```

Arguments

`formula`	A dataframe of xreg (exogenous regressors)
`data`	A numeric vector of values to fit
`family`	The model family for the observation equation. Non-Gaussian model families use data augmentation to recover a conditionally Gaussian model.
`...`	Additional arguments passed to `forecast::Arima`

Value

A modeltime model

bayesian_structural_stan_predict_impl *Bridge prediction function for ARIMA models*

Description

Bridge prediction function for ARIMA models

Usage

```
bayesian_structural_stan_predict_impl(object, new_data, ...)
```

Arguments

object	An object of class model_fit
new_data	A rectangular data object, such as a data frame.
...	Additional arguments passed to forecast::Arima()

Value

A prediction

exponential_smoothing *General Interface for Exponential Smoothing Models*

Description

exponential_smoothing() is a way to generate a *specification* of an ETS model before fitting and allows the model to be created using different packages. Currently the only package is Rlgt.

Usage

```
exponential_smoothing(
  mode = "regression",
  seasonality = NULL,
  second_seasonality = NULL,
  seasonality_type = NULL,
  method = NULL,
  error_method = NULL
)
```

Arguments

mode	A single character string for the type of model. The only possible value for this model is "regression".
seasonality	This specification of seasonality will be overridden by frequency of y, if y is of ts or msts class. 1 by default, i.e. no seasonality.
second_seasonality	Second seasonality.
seasonality_type	Either "multiplicative" (default) or "generalized". The latter seasonality generalizes additive and multiplicative seasonality types.
method	"HW", "seasAvg", "HW_sAvg". Here, "HW" follows Holt-Winters approach. "seasAvg" calculates level as a smoothed average of the last seasonality number of points (or seasonality2 of them for the dual seasonality model), and HW_sAvg is an weighted average of HW and seasAvg methods.
error_method	Function providing size of the error. Either "std" (monotonically, but slower than proportionally, growing with the series values) or "innov" (proportional to a smoothed abs size of innovations, i.e. surprises)

Details

The data given to the function are not saved and are only used to determine the *mode* of the model. For `exponential_smoothing()`, the mode will always be "regression".

The model can be created using the `fit()` function using the following *engines*:

- "stan" (default) - Connects to `Rlgt::rlgt()`

Main Arguments

The main arguments (tuning parameters) for the model are:

- `seasonality`: Seasonality.
- `second_seasonality`: Second seasonality.
- `seasonality_type`: Either "multiplicative" (default) or "generalized".
- `method`: "HW", "seasAvg", "HW_sAvg"
- `error_method`: Either "std" or "innov"

These arguments are converted to their specific names at the time that the model is fit.

Other options and argument can be set using `set_engine()`.

If parameters need to be modified, `update()` can be used in lieu of recreating the object from scratch.

stan (default engine)

The engine uses `Rlgt::rlgt()`.

Parameter Notes:

- `xreg` - This is supplied via the parsnip / bayesmodels `fit()` interface (so don't provide this manually). See Fit Details (below).

Value

A model spec

Fit Details

Date and Date-Time Variable

It's a requirement to have a date or date-time variable as a predictor. The `fit()` interface accepts date and date-time features and handles them internally.

- `fit(y ~ date)`

Univariate (No xregs, Exogenous Regressors):

For univariate analysis, you must include a date or date-time feature. Simply use:

- Formula Interface: `fit(y ~ date)` will ignore xreg's.

Multivariate (xregs, Exogenous Regressors)

The `xreg` parameter is populated using the `fit()` function:

- Only `factor`, ordered factor, and `numeric` data will be used as xregs.
- Date and Date-time variables are not used as xregs
- `character` data should be converted to factor.

Xreg Example: Suppose you have 3 features:

1. `y` (target)
2. `date` (time stamp),
3. `month.lbl` (labeled month as a ordered factor).

The `month.lbl` is an exogenous regressor that can be passed to the `expotential_smoothing()` using `fit()`:

- `fit(y ~ date + month.lbl)` will pass `month.lbl` on as an exogenous regressor.

Note that date or date-time class values are excluded from `xreg`.

See Also

`fit.model_spec()`, `set_engine()`

Examples

```
## Not run:
library(dplyr)
library(parsnip)
library(rsample)
library(timetk)
library(modeltime)
library(bayesmodels)

# Data
```

```
m750 <- m4_monthly %>% filter(id == "M750")
m750

# Split Data 80/20
splits <- rsample::initial_time_split(m750, prop = 0.8)

# ---- ARIMA ----

# Model Spec
model_spec <- exponential_smoothing() %>%
    set_engine("stan")

# Fit Spec
model_fit <- model_spec %>%
    fit(log(value) ~ date + month(date), data = training(splits))
model_fit

## End(Not run)
```

`exponential_smoothing_params` *Tuning Parameters for Exponential Smoothing Models*

Description

Tuning Parameters for Exponential Smoothing Models

Usage

```
seasonality_type()

method()

error_method()
```

Details

The main parameters for Exponential Smoothing models are:

- `garch_order`: Integer with the garch order.
- `arch_order`: Integer with the arch_order.
- `mgarch_order`: Integer with the mgarch order.
- `garch_t_student`: A boolean value to specify for a generalized t-student garch model.
- `asymmetry`: a string value for the asymmetric function for an asymmetric GARCH process. By default the value "none" for standard GARCH process. If "logit" a logistic function is used for asymmetry, and if "exp" an exponential function is used.
- `non_seasonal_ar`: The order of the non-seasonal auto-regressive (AR) terms.

- non_seasonal_ma: The order of the non-seasonal moving average (MA) terms.
- markov_chains: The number of markov chains.
- adapt_delta: The thin of the jumps in a HMC method
- tree_depth: Maximum depth of the trees

Value

A parameter

A parameter

A parameter

Examples

```
non_seasonal_ar()

non_seasonal_differences()

non_seasonal_ma()
```

---

exp_smoothing_stan_fit_impl

Low-Level ARIMA function for translating modeltime to forecast

---

Description

Low-Level ARIMA function for translating modeltime to forecast

Usage

```
exp_smoothing_stan_fit_impl(
  x,
  y,
  seasonality = 1,
  seasonality2 = 1,
  seasonality.type = "multiplicative",
  error.size.method = "std",
  level.method = "HW",
  ...
)
```

Arguments

x	A dataframe of xreg (exogenous regressors)
y	A numeric vector of values to fit
seasonality	Seasonality
seasonality2	Second seasonality
seasonality.type	Either "multiplicative" (default) or "generalized". The latter seasonality generalizes additive and multiplicative seasonality types.
error.size.method	Either "std" (monotonically, but slower than proportionally, growing with the series values) or "innov" (proportional to a smoothed abs size of innovations, i.e. surprises)
level.method	"HW", "seasAvg", "HW_sAvg"
...	Additional arguments passed to `forecast::Arima`

Value

A modeltime model

---

exp_smoothing_stan_predict_impl

Bridge prediction function for ARIMA models

Description

Bridge prediction function for ARIMA models

Usage

```
exp_smoothing_stan_predict_impl(object, new_data, ...)
```

Arguments

object	An object of class `model_fit`
new_data	A rectangular data object, such as a data frame.
...	Additional arguments passed to `forecast::Arima()`

Value

A prediction

garch_params *Tuning Parameters for GARCHA Models*

Description

Tuning Parameters for GARCHA Models

Usage

```
garch_order(range = c(0L, 3L), trans = NULL)

arch_order(range = c(0L, 3L), trans = NULL)

mgarch_order(range = c(0L, 3L), trans = NULL)

garch_t_student()

asymmetry()
```

Arguments

range	A two-element vector holding the *defaults* for the smallest and largest possible values, respectively.
trans	A trans object from the scales package, such as scales::log10_trans() or scales::reciprocal_trans(). If not provided, the default is used which matches the units used in range. If no transformation, NULL.

Details

The main parameters for GARCHA models are:

- garch_order: Integer with the garch order.
- arch_order: Integer with the arch_order.
- mgarch_order: Integer with the mgarch order.
- garch_t_student: A boolean value to specify for a generalized t-student garch model.
- asymmetry: a string value for the asymmetric function for an asymmetric GARCH process. By default the value "none" for standard GARCH process. If "logit" a logistic function is used for asymmetry, and if "exp" an exponential function is used.
- non_seasonal_ar: The order of the non-seasonal auto-regressive (AR) terms.
- non_seasonal_ma: The order of the non-seasonal moving average (MA) terms.
- markov_chains: The number of markov chains.
- adapt_delta: The thin of the jumps in a HMC method
- tree_depth: Maximum depth of the trees

Value

A parameter

A parameter

A parameter

A parameter

A parameter

Examples

```
non_seasonal_ar()

non_seasonal_differences()

non_seasonal_ma()
```

`garch_reg` *General Interface for GARCH Regression Models*

Description

`garch_reg()` is a way to generate a *specification* of a GARCH model before fitting and allows the model to be created using different packages. Currently the only package is `bayesforecast`.

Usage

```
garch_reg(
  mode = "regression",
  garch_order = NULL,
  arch_order = NULL,
  mgarch_order = NULL,
  non_seasonal_ar = NULL,
  non_seasonal_ma = NULL,
  garch_t_student = NULL,
  asymmetry = NULL,
  markov_chains = NULL,
  chain_iter = NULL,
  warmup_iter = NULL,
  adapt_delta = NULL,
  tree_depth = NULL,
  pred_seed = NULL
)
```

Arguments

mode	A single character string for the type of model. The only possible value for this model is "regression".
garch_order	Integer with the garch order.
arch_order	Integer with the arch_order.
mgarch_order	Integer with the mgarch order.
non_seasonal_ar	The order of the non-seasonal auto-regressive (AR) terms. Often denoted "p" in pdq-notation.
non_seasonal_ma	The order of the non-seasonal moving average (MA) terms. Often denoted "q" in pdq-notation
garch_t_student	A boolean value to specify for a generalized t-student garch model.
asymmetry	a string value for the asymmetric function for an asymmetric GARCH process. By default the value "none" for standard GARCH process. If "logit" a logistic function is used for asymmetry, and if "exp" an exponential function is used.
markov_chains	An integer of the number of Markov Chains chains to be run, by default 4 chains are run.
chain_iter	An integer of total iterations per chain including the warm-up, by default the number of iterations are 2000.
warmup_iter	A positive integer specifying number of warm-up (aka burn-in) iterations. This also specifies the number of iterations used for step-size adaptation, so warm-up samples should not be used for inference. The number of warmup should not be larger than iter and the default is iter/2.
adapt_delta	An optional real value between 0 and 1, the thin of the jumps in a HMC method. By default is 0.9
tree_depth	An integer of the maximum depth of the trees evaluated during each iteration. By default is 10.
pred_seed	An integer with the seed for using when predicting with the model.

Details

The data given to the function are not saved and are only used to determine the *mode* of the model. For `garch_reg()`, the mode will always be "regression".

The model can be created using the `fit()` function using the following *engines*:

- "stan" (default) - Connects to `bayesforecast::stan_garch()`

Main Arguments

The main arguments (tuning parameters) for the model are:

- `arch_order`: Integer with the arch_order.
- `garch_order`: Integer with the garch_order.

- `mgarch_order`: Integer with the mgarch_order.
- `garch_t_student`: A boolean value to specify for a generalized t-student garch model.
- `asymmetry`: a string value for the asymmetric function for an asymmetric GARCH process.
- `non_seasonal_ar`: The order of the non-seasonal auto-regressive (AR) terms.
- `non_seasonal_ma`: The order of the non-seasonal moving average (MA)
- `markov_chains`: An integer of the number of Markov Chains chains to be run.
- `adapt_delta`: The thin of the jumps in a HMC method.
- `tree_depth`: The maximum depth of the trees evaluated during each iteration.

These arguments are converted to their specific names at the time that the model is fit.

Other options and argument can be set using `set_engine()` (See Engine Details below).

If parameters need to be modified, `update()` can be used in lieu of recreating the object from scratch.

Value

A model spec

Engine Details

The standardized parameter names in `bayesforecast` can be mapped to their original names in each engine:

bayesmodels	bayesforecast::stan_garch
arch_order, garch_order, mgarch_order	order = c(s(1), k(1), h(0))
non_seasonal_ar, non_seasonal_ma	arma = c(p(1), q(0))
garch_t_student	genT(FALSE)
assymetry	asym('none')
markov_chains	chains(4)
adapt_delta	adapt.delta(0.9)
tree_depth	tree.depth(10)

Other options can be set using `set_engine()`.

stan (default engine)

The engine uses `bayesforecast::stan_garch()`.

Parameter Notes:

- `xreg` - This is supplied via the parsnip / modeltime `fit()` interface (so don't provide this manually). See Fit Details (below).

Fit Details

Date and Date-Time Variable

It's a requirement to have a date or date-time variable as a predictor. The `fit()` interface accepts date and date-time features and handles them internally.

- `fit(y ~ date)`

Seasonal Period Specification

The period can be non-seasonal (seasonal_period = 1 or "none") or yearly seasonal (e.g. For monthly time stamps, `seasonal_period = 12`, `seasonal_period = "12 months"`, or `seasonal_period = "yearly"`). There are 3 ways to specify:

1. `seasonal_period = "auto"`: A seasonal period is selected based on the periodicity of the data (e.g. 12 if monthly)
2. `seasonal_period = 12`: A numeric frequency. For example, 12 is common for monthly data
3. `seasonal_period = "1 year"`: A time-based phrase. For example, "1 year" would convert to 12 for monthly data.

Univariate (No xregs, Exogenous Regressors):

For univariate analysis, you must include a date or date-time feature. Simply use:

- Formula Interface (recommended): `fit(y ~ date)` will ignore xreg's.

Multivariate (xregs, Exogenous Regressors)

The `xreg` parameter is populated using the `fit()` or `fit_xy()` function:

- Only `factor`, ordered factor, and `numeric` data will be used as xregs.
- Date and Date-time variables are not used as xregs
- `character` data should be converted to factor.

Xreg Example: Suppose you have 3 features:

1. `y` (target)
2. `date` (time stamp),
3. `month.lbl` (labeled month as a ordered factor).

The `month.lbl` is an exogenous regressor that can be passed to the `garch_reg()` using `fit()`:

- `fit(y ~ date + month.lbl)` will pass `month.lbl` on as an exogenous regressor.

Note that date or date-time class values are excluded from `xreg`.

See Also

`fit.model_spec()`, `set_engine()`

Examples

```
## Not run:
library(dplyr)
library(parsnip)
library(rsample)
library(timetk)
library(modeltime)
library(bayesmodels)
```

```
# Data
m750 <- m4_monthly %>% filter(id == "M750")
m750

# Split Data 80/20
splits <- rsample::initial_time_split(m750, prop = 0.8)

# ---- AUTO ARIMA ----

# Model Spec
model_spec <- garch_reg() %>%
    set_engine("stan")

# Fit Spec
model_fit <- model_spec %>%
    fit(log(value) ~ date, data = training(splits))
model_fit


# Model Spec
model_spec <- garch_reg(
        arch_order               = 2,
        garch_order              = 2,
        mgarch_order             = 1,
        non_seasonal_ar          = 1,
        non_seasonal_ma          = 1
    ) %>%
    set_engine("stan")

# Fit Spec
model_fit <- model_spec %>%
    fit(log(value) ~ date, data = training(splits))
model_fit

## End(Not run)
```

`garch_stan_fit_impl` *Low-Level ARIMA function for translating modeltime to forecast*

Description

Low-Level ARIMA function for translating modeltime to forecast

Usage

```
garch_stan_fit_impl(
  x,
  y,
  s = 1,
  k = 1,
```

```
  h = 1,
  p = 0,
  q = 0,
  genT = FALSE,
  asym = "none",
  chains = 4,
  iter = 2000,
  warmup = iter/2,
  adapt.delta = 0.9,
  tree.depth = 10,
  seed = NULL,
  ...
)
```

Arguments

x	A dataframe of xreg (exogenous regressors)
y	A numeric vector of values to fit
s	garch_order
k	arch_order
h	mgarch_order
p	The order of the non-seasonal auto-regressive (AR) terms. Often denoted "p" in pdq-notation.
q	The order of the non-seasonal moving average (MA) terms. Often denoted "q" in pdq-notation.
genT	a boolean value to specify for a generalized t-student garch model.
asym	a string value for the asymmetric function for an asymmetric GARCH process.
chains	An integer of the number of Markov Chains chains to be run, by default 4 chains are run.
iter	An integer of total iterations per chain including the warm-up, by default the number of iterations are 2000.
warmup	A positive integer specifying number of warm-up (aka burn-in) iterations. This also specifies the number of iterations used for step-size adaptation, so warm-up samples should not be used for inference. The number of warmup should not be larger than iter and the default is iter/2.
adapt.delta	An optional real value between 0 and 1, the thin of the jumps in a HMC method. By default is 0.9
tree.depth	An integer of the maximum depth of the trees evaluated during each iteration. By default is 10.
seed	An integer with the seed for using when predicting with the model.
...	Additional arguments passed to `forecast::Arima`

Value

A modeltime model

garch_stan_predict_impl *Bridge prediction function for ARIMA models*

Description

Bridge prediction function for ARIMA models

Usage

```
garch_stan_predict_impl(object, new_data, ...)
```

Arguments

object	An object of class model_fit
new_data	A rectangular data object, such as a data frame.
...	Additional arguments passed to forecast::Arima()

Value

A prediction

gen_additive_reg *Interface for Generalized Additive Models (GAM)*

Description

Interface for Generalized Additive Models (GAM)

Usage

```
gen_additive_reg(
  mode = "regression",
  markov_chains = NULL,
  chain_iter = NULL,
  warmup_iter = NULL,
  adapt_delta = NULL
)
```

Arguments

mode	A single character string for the type of model.
markov_chains	Number of Markov chains (defaults to 4).
chain_iter	Number of total iterations per chain (including warmup; defaults to 2000).
warmup_iter	A positive integer specifying number of warmup (aka burnin) iterations. This also specifies the number of iterations used for stepsize adaptation, so warmup samples should not be used for inference. The number of warmup should not be larger than iter and the default is iter/2.
adapt_delta	The thin of the jumps in a HMC method.

Details

Available Engines:

- **stan**: Connects to `brms::brm()`

Value

A `parsnip` model specification

A model spec

Engine Details

stan

This engine uses `brms::brm()` and has the following parameters, which can be modified through the `parsnip::set_engine()` function.

```
## function (formula, data, family = gaussian(), prior = NULL, autocor = NULL,
##     data2 = NULL, cov_ranef = NULL, sample_prior = "no", sparse = NULL,
##    knots = NULL, stanvars = NULL, stan_funs = NULL, fit = NA, save_pars = NULL,
##    save_ranef = NULL, save_mevars = NULL, save_all_pars = NULL, inits = "random",
##    chains = 4, iter = 2000, warmup = floor(iter/2), thin = 1, cores = getOption("mc.cores",
##         1), threads = NULL, normalize = getOption("brms.normalize", TRUE),
##     control = NULL, algorithm = getOption("brms.algorithm", "sampling"),
##    backend = getOption("brms.backend", "rstan"), future = getOption("future",
##      FALSE), silent = 1, seed = NA, save_model = NULL, stan_model_args = list(),
##     file = NULL, file_refit = "never", empty = FALSE, rename = TRUE, ...)
```

Fit Details

BRMS Formula Interface

Fitting GAMs is accomplished using parameters including:

- `brms::s()`: GAM spline smooths
- `brms::t2()`: GAM tensor product smooths

These are applied in the `fit()` function:

```
fit(value ~ s(date_mon, k = 12) + s(date_num), data = df)
```

Examples

```
## Not run:
library(tidymodels)
library(bayesmodels)
library(modeltime)
library(tidyverse)
library(timetk)
library(lubridate)

m750_extended <- m750 %>%
    group_by(id) %>%
    future_frame(.length_out = 24, .bind_data = TRUE) %>%
    mutate(lag_24 = lag(value, 24)) %>%
    ungroup() %>%
    mutate(date_num = as.numeric(date)) %>%
    mutate(date_month = month(date))

m750_train  <- m750_extended %>% drop_na()
m750_future <- m750_extended %>% filter(is.na(value))

model_fit_gam <- gen_additive_reg(mode = "regression", markov_chains = 2) %>%
    set_engine("stan", family=Gamma(link="log")) %>%
    fit(value ~  date + s(date_month, k = 12)
        + s(lag_24),
        data = m750_train)

## End(Not run)
```

---

gen_additive_stan_fit_impl

Low-Level ARIMA function for translating modeltime to forecast

---

Description

Low-Level ARIMA function for translating modeltime to forecast

Usage

```
gen_additive_stan_fit_impl(
  formula,
  data,
  chains = 4,
  iter = 2000,
  warmup = 1000,
  ...
)
```

Arguments

formula	A dataframe of xreg (exogenous regressors)
data	A numeric vector of values to fit
chains	An integer of the number of Markov Chains chains to be run, by default 4 chains are run.
iter	An integer of total iterations per chain including the warm-up, by default the number of iterations are 2000.
warmup	A positive integer specifying number of warm-up (aka burn-in) iterations. This also specifies the number of iterations used for step-size adaptation, so warm-up samples should not be used for inference. The number of warmup should not be larger than iter and the default is iter/2.
...	Additional arguments passed to forecast::Arima

Value

A modeltime model

---

gen_additive_stan_predict_impl

Bridge prediction function for ARIMA models

Description

Bridge prediction function for ARIMA models

Usage

```
gen_additive_stan_predict_impl(object, new_data, ...)
```

Arguments

object	An object of class model_fit
new_data	A rectangular data object, such as a data frame.
...	Additional arguments passed to forecast::Arima()

Value

A prediction

naive_params *Tuning Parameters for Random Walk Models*

Description

Tuning Parameters for Random Walk Models

Usage

```
seasonal_random_walk()
```

Details

The main parameters for Random Walk Models are:

- `seasonal_random_walk`: A boolean value for select a seasonal random walk instead.
- `markov_chains`: The number of markov chains.
- `adapt_delta`: The thin of the jumps in a HMC method
- `tree_depth`: Maximum depth of the trees

Value

A parameter

random_walk_reg *General Interface for Naive and Random Walk models Regression Models*

Description

`random_walk_reg()` is a way to generate a *specification* of Naive and Random Walk models before fitting and allows the model to be created using different packages. Currently the only package is `bayesforecast`.

Usage

```
random_walk_reg(
  mode = "regression",
  seasonal_random_walk = NULL,
  seasonal_period = NULL,
  markov_chains = NULL,
  chain_iter = NULL,
  warmup_iter = NULL,
  adapt_delta = NULL,
  tree_depth = NULL,
  pred_seed = NULL
)
```

Arguments

mode	A single character string for the type of model. The only possible value for this model is "regression".
seasonal_random_walk	a Boolean value for select a seasonal random walk instead.
seasonal_period	an optional integer value for the seasonal period.
markov_chains	An integer of the number of Markov Chains chains to be run, by default 4 chains are run.
chain_iter	An integer of total iterations per chain including the warm-up, by default the number of iterations are 2000.
warmup_iter	A positive integer specifying number of warm-up (aka burn-in) iterations. This also specifies the number of iterations used for step-size adaptation, so warm-up samples should not be used for inference. The number of warmup should not be larger than iter and the default is iter/2.
adapt_delta	An optional real value between 0 and 1, the thin of the jumps in a HMC method. By default is 0.9
tree_depth	An integer of the maximum depth of the trees evaluated during each iteration. By default is 10.
pred_seed	An integer with the seed for using when predicting with the model.

Details

The data given to the function are not saved and are only used to determine the *mode* of the model. For `random_walk_reg()`, the mode will always be "regression".

The model can be created using the `fit()` function using the following *engines*:

- "stan" (default) - Connects to `bayesforecast::stan_naive()`

Main Arguments

The main arguments (tuning parameters) for the model are:

- `seasonal_random_walk`: a Boolean value for select a seasonal random walk instead.
- `markov_chains`: An integer of the number of Markov Chains chains to be run.
- `adapt_delta`: The thin of the jumps in a HMC method.
- `tree_depth`: The maximum depth of the trees evaluated during each iteration.

These arguments are converted to their specific names at the time that the model is fit.

Other options and argument can be set using `set_engine()` (See Engine Details below).

If parameters need to be modified, `update()` can be used in lieu of recreating the object from scratch.

Value

A model spec

Engine Details

The standardized parameter names in `bayesmodels` can be mapped to their original names in each engine:

bayesmodels	bayesforecast::stan_naive
seasonal_random_walk	seasonal
markov_chains	chains(4)
adapt_delta	adapt.delta(0.9)
tree_depth	tree.depth(10)

Other options can be set using `set_engine()`.

stam (default engine)

The engine uses `bayesforecast::stan_naive()`.

Fit Details

Date and Date-Time Variable

It's a requirement to have a date or date-time variable as a predictor. The `fit()` interface accepts date and date-time features and handles them internally.

- `fit(y ~ date)`

Seasonal Period Specification

The period can be non-seasonal (seasonal_period = 1 or "none") or yearly seasonal (e.g. For monthly time stamps, `seasonal_period = 12`, `seasonal_period = "12 months"`, or `seasonal_period = "yearly"`). There are 3 ways to specify:

1. `seasonal_period = "auto"`: A seasonal period is selected based on the periodicity of the data (e.g. 12 if monthly)
2. `seasonal_period = 12`: A numeric frequency. For example, 12 is common for monthly data
3. `seasonal_period = "1 year"`: A time-based phrase. For example, "1 year" would convert to 12 for monthly data.

Univariate (No xregs, Exogenous Regressors):

For univariate analysis, you must include a date or date-time feature. Simply use:

- Formula Interface (recommended): `fit(y ~ date)` will ignore xreg's.

See Also

`fit.model_spec()`, `set_engine()`

Examples

```
## Not run:
library(dplyr)
library(parsnip)
library(rsample)
library(timetk)
library(modeltime)
library(bayesmodels)

# Data
m750 <- m4_monthly %>% filter(id == "M750")
m750

# Split Data 80/20
splits <- rsample::initial_time_split(m750, prop = 0.8)


# Model Spec
model_spec <- random_walk_reg() %>%
    set_engine("stan")

# Fit Spec
model_fit <- model_spec %>%
    fit(log(value) ~ date, data = training(splits))
model_fit

## End(Not run)
```

---

random_walk_stan_fit_impl

Low-Level ARIMA function for translating modeltime to forecast

---

Description

Low-Level ARIMA function for translating modeltime to forecast

Usage

```
random_walk_stan_fit_impl(
  x,
  y,
  seasonal = FALSE,
  m = 0,
  chains = 4,
  iter = 2000,
  warmup = iter/2,
  adapt.delta = 0.9,
  tree.depth = 10,
```

```
  seed = NULL,
  ...
)
```

Arguments

x	A dataframe of xreg (exogenous regressors)
y	A numeric vector of values to fit
seasonal	a Boolean value for select a seasonal random walk instead
m	an optional integer value for the seasonal period.
chains	An integer of the number of Markov Chains chains to be run, by default 4 chains are run.
iter	An integer of total iterations per chain including the warm-up, by default the number of iterations are 2000.
warmup	A positive integer specifying number of warm-up (aka burn-in) iterations. This also specifies the number of iterations used for step-size adaptation, so warm-up samples should not be used for inference. The number of warmup should not be larger than iter and the default is iter/2.
adapt.delta	An optional real value between 0 and 1, the thin of the jumps in a HMC method. By default is 0.9
tree.depth	An integer of the maximum depth of the trees evaluated during each iteration. By default is 10.
seed	An integer with the seed for using when predicting with the model.
...	Additional arguments passed to `forecast::Arima`

Value

A modeltime model

`random_walk_stan_predict_impl`

Bridge prediction function for ARIMA models

Description

Bridge prediction function for ARIMA models

Usage

```
random_walk_stan_predict_impl(object, new_data, ...)
```

Arguments

object	An object of class `model_fit`
new_data	A rectangular data object, such as a data frame.
...	Additional arguments passed to `forecast::Arima()`

Value

A prediction

`sarima_params` *Tuning Parameters for SARIMA Models*

Description

Tuning Parameters for SARIMA Models

Usage

```
non_seasonal_ar(range = c(0L, 5L), trans = NULL)

non_seasonal_differences(range = c(0L, 2L), trans = NULL)

non_seasonal_ma(range = c(0L, 5L), trans = NULL)

seasonal_ar(range = c(0L, 2L), trans = NULL)

seasonal_differences(range = c(0L, 1L), trans = NULL)

seasonal_ma(range = c(0L, 2L), trans = NULL)

markov_chains(range = c(0L, 8L), trans = NULL)

adapt_delta(range = c(0, 1), trans = NULL)

tree_depth(range = c(0L, 100L), trans = NULL)
```

Arguments

`range`	A two-element vector holding the *defaults* for the smallest and largest possible values, respectively.
`trans`	A `trans` object from the `scales` package, such as `scales::log10_trans()` or `scales::reciprocal_trans()`. If not provided, the default is used which matches the units used in `range`. If no transformation, `NULL`.

Details

The main parameters for SARIMA models are:

- `non_seasonal_ar`: The order of the non-seasonal auto-regressive (AR) terms.
- `non_seasonal_differences`: The order of integration for non-seasonal differencing.
- `non_seasonal_ma`: The order of the non-seasonal moving average (MA) terms.
- `seasonal_ar`: The order of the seasonal auto-regressive (SAR) terms.

- `seasonal_differences`: The order of integration for seasonal differencing.
- `seasonal_ma`: The order of the seasonal moving average (SMA) terms.
- `markov_chains`: The number of markov chains.
- `adapt_delta`: The thin of the jumps in a HMC method
- `tree_depth`: Maximum depth of the trees

Value

A parameter

A parameter

A parameter

A parameter

A parameter

A parameter

A parameter

A parameter

A parameter

Examples

```
non_seasonal_ar()

non_seasonal_differences()

non_seasonal_ma()
```

`sarima_reg` *General Interface for ARIMA Regression Models*

Description

`sarima_reg()` is a way to generate a *specification* of an ARIMA model before fitting and allows the model to be created using different packages. Currently the only package is `bayesforecast`.

Usage

```
sarima_reg(
  mode = "regression",
  seasonal_period = NULL,
  non_seasonal_ar = NULL,
  non_seasonal_differences = NULL,
  non_seasonal_ma = NULL,
```

```
  seasonal_ar = NULL,
  seasonal_differences = NULL,
  seasonal_ma = NULL,
  markov_chains = NULL,
  chain_iter = NULL,
  warmup_iter = NULL,
  adapt_delta = NULL,
  tree_depth = NULL,
  pred_seed = NULL
)
```

Arguments

mode	A single character string for the type of model. The only possible value for this model is "regression".
seasonal_period	A seasonal frequency. Uses "auto" by default. A character phrase of "auto" or time-based phrase of "2 weeks" can be used if a date or date-time variable is provided. See Fit Details below.
non_seasonal_ar	The order of the non-seasonal auto-regressive (AR) terms. Often denoted "p" in pdq-notation.
non_seasonal_differences	The order of integration for non-seasonal differencing. Often denoted "d" in pdq-notation.
non_seasonal_ma	The order of the non-seasonal moving average (MA) terms. Often denoted "q" in pdq-notation.
seasonal_ar	The order of the seasonal auto-regressive (SAR) terms. Often denoted "P" in PDQ-notation.
seasonal_differences	The order of integration for seasonal differencing. Often denoted "D" in PDQ-notation.
seasonal_ma	The order of the seasonal moving average (SMA) terms. Often denoted "Q" in PDQ-notation.
markov_chains	An integer of the number of Markov Chains chains to be run, by default 4 chains are run.
chain_iter	An integer of total iterations per chain including the warm-up, by default the number of iterations are 2000.
warmup_iter	A positive integer specifying number of warm-up (aka burn-in) iterations. This also specifies the number of iterations used for step-size adaptation, so warm-up samples should not be used for inference. The number of warmup should not be larger than iter and the default is iter/2.
adapt_delta	An optional real value between 0 and 1, the thin of the jumps in a HMC method. By default is 0.9

tree_depth	An integer of the maximum depth of the trees evaluated during each iteration. By default is 10.
pred_seed	An integer with the seed for using when predicting with the model.

Details

The data given to the function are not saved and are only used to determine the *mode* of the model. For sarima_reg(), the mode will always be "regression".

The model can be created using the fit() function using the following *engines*:

- "stan" (default) - Connects to bayesforecast::stan_sarima()

Main Arguments

The main arguments (tuning parameters) for the model are:

- non_seasonal_ar: The order of the non-seasonal auto-regressive (AR) terms.
- non_seasonal_differences: The order of integration for non-seasonal differencing.
- non_seasonal_ma: The order of the non-seasonal moving average (MA) terms.
- seasonal_ar: The order of the seasonal auto-regressive (SAR) terms.
- seasonal_differences: The order of integration for seasonal differencing.
- seasonal_ma: The order of the seasonal moving average (SMA) terms.
- markov_chains: An integer of the number of Markov Chains chains to be run.
- adapt_delta: The thin of the jumps in a HMC method.
- tree_depth: The maximum depth of the trees evaluated during each iteration

These arguments are converted to their specific names at the time that the model is fit.

Other options and argument can be set using set_engine() (See Engine Details below).

If parameters need to be modified, update() can be used in lieu of recreating the object from scratch.

Value

A model spec

Engine Details

The standardized parameter names in bayesmodels can be mapped to their original names in the engine:

bayesmodels	bayesforecast::stan_sarima
non_seasonal_ar, non_seasonal_differences, non_seasonal_ma	order = c(p(1), d(0), q(0))
seasonal_ar, seasonal_differences, seasonal_ma	seasonal = c(P(0), D(0), Q(0))
markov_chains	chains(4)
adapt_delta	adapt.delta(0.9)
tree_depth	tree.depth(10)

Other options can be set using `set_engine()`.

stan (default engine)

The engine uses `bayesforecast::stan_sarima()`.

Parameter Notes:

- `xreg` - This is supplied via the parsnip / bayesmodels `fit()` interface (so don't provide this manually). See Fit Details (below).

Fit Details

Date and Date-Time Variable

It's a requirement to have a date or date-time variable as a predictor. The `fit()` interface accepts date and date-time features and handles them internally.

- `fit(y ~ date)`

Seasonal Period Specification

The period can be non-seasonal (seasonal_period = 1 or "none") or yearly seasonal (e.g. For monthly time stamps, `seasonal_period = 12`, `seasonal_period = "12 months"`, or `seasonal_period = "yearly"`). There are 3 ways to specify:

1. `seasonal_period = "auto"`: A seasonal period is selected based on the periodicity of the data (e.g. 12 if monthly)
2. `seasonal_period = 12`: A numeric frequency. For example, 12 is common for monthly data
3. `seasonal_period = "1 year"`: A time-based phrase. For example, "1 year" would convert to 12 for monthly data.

Univariate (No xregs, Exogenous Regressors):

For univariate analysis, you must include a date or date-time feature. Simply use:

- Formula Interface: `fit(y ~ date)` will ignore xreg's.

Multivariate (xregs, Exogenous Regressors)

The `xreg` parameter is populated using the `fit()` function:

- Only `factor`, ordered factor, and `numeric` data will be used as xregs.
- Date and Date-time variables are not used as xregs
- `character` data should be converted to factor.

Xreg Example: Suppose you have 3 features:

1. `y` (target)
2. `date` (time stamp),
3. `month.lbl` (labeled month as a ordered factor).

The `month.lbl` is an exogenous regressor that can be passed to the `sarima_reg()` using `fit()`:

- `fit(y ~ date + month.lbl)` will pass `month.lbl` on as an exogenous regressor.

Note that date or date-time class values are excluded from `xreg`.

See Also

fit.model_spec(), set_engine()

Examples

```
## Not run:
library(dplyr)
library(parsnip)
library(rsample)
library(timetk)
library(modeltime)
library(bayesmodels)

# Data
m750 <- m4_monthly %>% filter(id == "M750")
m750

# Split Data 80/20
splits <- rsample::initial_time_split(m750, prop = 0.8)

# ---- ARIMA ----

# Model Spec
model_spec <- sarima_reg() %>%
    set_engine("stan")

# Fit Spec
model_fit <- model_spec %>%
    fit(log(value) ~ date, data = training(splits))
model_fit



# Model Spec
model_spec <- sarima_reg(
        seasonal_period          = 12,
        non_seasonal_ar          = 3,
        non_seasonal_differences = 1,
        non_seasonal_ma          = 3,
        seasonal_ar              = 1,
        seasonal_differences     = 0,
        seasonal_ma              = 1
    ) %>%
    set_engine("stan")

# Fit Spec
model_fit <- model_spec %>%
    fit(log(value) ~ date, data = training(splits))
model_fit

## End(Not run)
```

`Sarima_stan_fit_impl` *Low-Level ARIMA function for translating modeltime to forecast*

Description

Low-Level ARIMA function for translating modeltime to forecast

Usage

```
Sarima_stan_fit_impl(
  x,
  y,
  period = "auto",
  p = 0,
  d = 0,
  q = 0,
  P = 0,
  D = 0,
  Q = 0,
  chains = 4,
  iter = 2000,
  warmup = iter/2,
  adapt.delta = 0.9,
  tree.depth = 10,
  seed = NULL,
  ...
)
```

Arguments

`x`	A dataframe of xreg (exogenous regressors)
`y`	A numeric vector of values to fit
`period`	A seasonal frequency. Uses "auto" by default. A character phrase of "auto" or time-based phrase of "2 weeks" can be used if a date or date-time variable is provided.
`p`	The order of the non-seasonal auto-regressive (AR) terms. Often denoted "p" in pdq-notation.
`d`	The order of integration for non-seasonal differencing. Often denoted "d" in pdq-notation.
`q`	The order of the non-seasonal moving average (MA) terms. Often denoted "q" in pdq-notation.
`P`	The order of the seasonal auto-regressive (SAR) terms. Often denoted "P" in PDQ-notation.
`D`	The order of integration for seasonal differencing. Often denoted "D" in PDQ-notation.

Q	The order of the seasonal moving average (SMA) terms. Often denoted "Q" in PDQ-notation.
chains	An integer of the number of Markov Chains chains to be run, by default 4 chains are run.
iter	An integer of total iterations per chain including the warm-up, by default the number of iterations are 2000.
warmup	A positive integer specifying number of warm-up (aka burn-in) iterations. This also specifies the number of iterations used for step-size adaptation, so warm-up samples should not be used for inference. The number of warmup should not be larger than iter and the default is iter/2.
adapt.delta	An optional real value between 0 and 1, the thin of the jumps in a HMC method. By default is 0.9
tree.depth	An integer of the maximum depth of the trees evaluated during each iteration. By default is 10.
seed	An integer with the seed for using when predicting with the model.
...	Additional arguments passed to forecast::Arima

Value

A modeltime model

`Sarima_stan_predict_impl`

Bridge prediction function for ARIMA models

Description

Bridge prediction function for ARIMA models

Usage

```
Sarima_stan_predict_impl(object, new_data, ...)
```

Arguments

object	An object of class model_fit
new_data	A rectangular data object, such as a data frame.
...	Additional arguments passed to forecast::Arima()

Value

A prediction

ssm_params *Tuning Parameters for Additive Linear State Space Regression Models*

Description

Tuning Parameters for Additive Linear State Space Regression Models

Usage

```
trend_model()

damped_model()

seasonal_model()
```

Details

The main parameters for Additive Linear State Space Regression Models are:

- `trend_model`: A boolean value to specify a trend local level model.
- `damped_model`: A boolean value to specify a damped trend local level model.
- `seasonal_model`: A boolean value to specify a seasonal trend local level model.
- `markov_chains`: The number of markov chains.
- `adapt_delta`: The thin of the jumps in a HMC method
- `tree_depth`: Maximum depth of the trees

Value

A parameter

A parameter

A parameter

Examples

```
damped_model()

seasonal_model()
```

`ssm_stan_fit_impl` *Low-Level ARIMA function for translating modeltime to forecast*

Description

Low-Level ARIMA function for translating modeltime to forecast

Usage

```
ssm_stan_fit_impl(
  x,
  y,
  trend = FALSE,
  damped = FALSE,
  seasonal = FALSE,
  period = 0,
  genT = FALSE,
  chains = 4,
  iter = 2000,
  warmup = iter/2,
  adapt.delta = 0.9,
  tree.depth = 10,
  seed = NULL,
  ...
)
```

Arguments

`x`	A dataframe of xreg (exogenous regressors)
`y`	A numeric vector of values to fit
`trend`	a boolean value to specify a trend local level model. By default is FALSE.
`damped`	a boolean value to specify a damped trend local level model. By default is FALSE.
`seasonal`	a boolean value to specify a seasonal local level model.
`period`	an integer specifying the periodicity of the time series.
`genT`	a boolean value to specify for a generalized t-student SSM model.
`chains`	An integer of the number of Markov Chains chains to be run, by default 4 chains are run.
`iter`	An integer of total iterations per chain including the warm-up, by default the number of iterations are 2000.
`warmup`	A positive integer specifying number of warm-up (aka burn-in) iterations. This also specifies the number of iterations used for step-size adaptation, so warm-up samples should not be used for inference. The number of warmup should not be larger than iter and the default is iter/2.

adapt.delta	An optional real value between 0 and 1, the thin of the jumps in a HMC method. By default is 0.9
tree.depth	An integer of the maximum depth of the trees evaluated during each iteration. By default is 10.
seed	An integer with the seed for using when predicting with the model.
...	Additional arguments passed to forecast::Arima

Value

A modeltime model

---

ssm_stan_predict_impl *Bridge prediction function for ARIMA models*

Description

Bridge prediction function for ARIMA models

Usage

```
ssm_stan_predict_impl(object, new_data, ...)
```

Arguments

object	An object of class model_fit
new_data	A rectangular data object, such as a data frame.
...	Additional arguments passed to forecast::Arima()

Value

A prediction

---

svm_reg *General Interface for Stochastic Volatility Regression Models*

Description

svm_reg() is a way to generate a *specification* of a Stochastic volatility model before fitting and allows the model to be created using different packages. Currently the only package is bayesforecast.

Usage

```
svm_reg(
  mode = "regression",
  non_seasonal_ar = NULL,
  non_seasonal_ma = NULL,
  markov_chains = NULL,
  chain_iter = NULL,
  warmup_iter = NULL,
  adapt_delta = NULL,
  tree_depth = NULL,
  pred_seed = NULL
)
```

Arguments

mode	A single character string for the type of model. The only possible value for this model is "regression".
non_seasonal_ar	The order of the non-seasonal auto-regressive (AR) terms. Often denoted "p" in pdq-notation.
non_seasonal_ma	The order of the non-seasonal moving average (MA) terms. Often denoted "q" in pdq-notation
markov_chains	An integer of the number of Markov Chains chains to be run, by default 4 chains are run.
chain_iter	An integer of total iterations per chain including the warm-up, by default the number of iterations are 2000.
warmup_iter	A positive integer specifying number of warm-up (aka burn-in) iterations. This also specifies the number of iterations used for step-size adaptation, so warm-up samples should not be used for inference. The number of warmup should not be larger than iter and the default is iter/2.
adapt_delta	An optional real value between 0 and 1, the thin of the jumps in a HMC method. By default is 0.9
tree_depth	An integer of the maximum depth of the trees evaluated during each iteration. By default is 10.
pred_seed	An integer with the seed for using when predicting with the model.

Details

The data given to the function are not saved and are only used to determine the *mode* of the model. For `svm_reg()`, the mode will always be "regression".

The model can be created using the `fit()` function using the following *engines*:

- "stan" (default) - Connects to `bayesforecast::stan_SVM()`

Main Arguments

The main arguments (tuning parameters) for the model are:

- non_seasonal_ar: The order of the non-seasonal auto-regressive (AR) terms.
- non_seasonal_ma: The order of the non-seasonal moving average (MA) terms.
- markov_chains: An integer of the number of Markov Chains chains to be run.
- adapt_delta: The thin of the jumps in a HMC method.
- tree_depth: The maximum depth of the trees evaluated during each iteration.

These arguments are converted to their specific names at the time that the model is fit.

Other options and argument can be set using set_engine() (See Engine Details below).

If parameters need to be modified, update() can be used in lieu of recreating the object from scratch.

Value

A model spec

Engine Details

The standardized parameter names in bayesmodels can be mapped to their original names in each engine:

bayesmodels	bayesforecast::stan_SVM
non_seasonal_ar, non_seasonal_ma	arma(0, 0)
markov_chains	chains(4)
adapt_delta	adapt.delta(0.9)
tree_depth	tree.depth(10)

Other options can be set using set_engine().

stan (default engine)

The engine uses bayesforecast::stan_SVM().

Parameter Notes:

- xreg - This is supplied via the parsnip / modeltime fit() interface (so don't provide this manually). See Fit Details (below).

Fit Details

Date and Date-Time Variable

It's a requirement to have a date or date-time variable as a predictor. The fit() interface accepts date and date-time features and handles them internally.

- fit(y ~ date)

Seasonal Period Specification

The period can be non-seasonal (seasonal_period = 1 or "none") or yearly seasonal (e.g. For monthly time stamps, seasonal_period = 12, seasonal_period = "12 months", or seasonal_period = "yearly"). There are 3 ways to specify:

1. `seasonal_period = "auto"`: A seasonal period is selected based on the periodicity of the data (e.g. 12 if monthly)
2. `seasonal_period = 12`: A numeric frequency. For example, 12 is common for monthly data
3. `seasonal_period = "1 year"`: A time-based phrase. For example, "1 year" would convert to 12 for monthly data.

Univariate (No xregs, Exogenous Regressors):

For univariate analysis, you must include a date or date-time feature. Simply use:

- Formula Interface (recommended): `fit(y ~ date)` will ignore xreg's.

Multivariate (xregs, Exogenous Regressors)

The `xreg` parameter is populated using the `fit()` or `fit_xy()` function:

- Only `factor`, ordered factor, and `numeric` data will be used as xregs.
- Date and Date-time variables are not used as xregs
- `character` data should be converted to factor.

Xreg Example: Suppose you have 3 features:

1. `y` (target)
2. `date` (time stamp),
3. `month.lbl` (labeled month as a ordered factor).

The `month.lbl` is an exogenous regressor that can be passed to the `arima_reg()` using `fit()`:

- `fit(y ~ date + month.lbl)` will pass `month.lbl` on as an exogenous regressor.

Note that date or date-time class values are excluded from `xreg`.

See Also

`fit.model_spec()`, `set_engine()`

Examples

```
## Not run:
library(dplyr)
library(parsnip)
library(rsample)
library(timetk)
library(modeltime)
library(bayesmodels)

# Data
m750 <- m4_monthly %>% filter(id == "M750")
m750

# Split Data 80/20
splits <- rsample::initial_time_split(m750, prop = 0.8)
```

```
# Model Spec
model_spec <- svm_reg() %>%
    set_engine("stan")

# Fit Spec
model_fit <- model_spec %>%
    fit(log(value) ~ date, data = training(splits))
model_fit


## End(Not run)
```

`svm_stan_fit_impl` *Low-Level ARIMA function for translating modeltime to forecast*

Description

Low-Level ARIMA function for translating modeltime to forecast

Usage

```
svm_stan_fit_impl(
  x,
  y,
  p = 0,
  q = 0,
  chains = 4,
  iter = 2000,
  warmup = iter/2,
  adapt.delta = 0.9,
  tree.depth = 10,
  seed = NULL,
  ...
)
```

Arguments

`x`	A dataframe of xreg (exogenous regressors)
`y`	A numeric vector of values to fit
`p`	The order of the non-seasonal auto-regressive (AR) terms. Often denoted "p" in pdq-notation.
`q`	The order of the non-seasonal moving average (MA) terms. Often denoted "q" in pdq-notation.
`chains`	An integer of the number of Markov Chains chains to be run, by default 4 chains are run.

iter	An integer of total iterations per chain including the warm-up, by default the number of iterations are 2000.
warmup	A positive integer specifying number of warm-up (aka burn-in) iterations. This also specifies the number of iterations used for step-size adaptation, so warm-up samples should not be used for inference. The number of warmup should not be larger than iter and the default is iter/2.
adapt.delta	An optional real value between 0 and 1, the thin of the jumps in a HMC method. By default is 0.9
tree.depth	An integer of the maximum depth of the trees evaluated during each iteration. By default is 10.
seed	An integer with the seed for using when predicting with the model.
...	Additional arguments passed to `forecast::Arima`

Value

A modeltime model

`svm_stan_predict_impl` *Bridge prediction function for ARIMA models*

Description

Bridge prediction function for ARIMA models

Usage

```
svm_stan_predict_impl(object, new_data, ...)
```

Arguments

object	An object of class `model_fit`
new_data	A rectangular data object, such as a data frame.
...	Additional arguments passed to `forecast::Arima()`

Value

A prediction

Index