Package: forecastML (via r-universe)

August 27, 2024

Type Package

Title Time Series Forecasting with Machine Learning Methods

Version 0.9.1

Author Nickalus Redell

Maintainer Nickalus Redell <nickalusredell@gmail.com>

Description The purpose of 'forecastML' is to simplify the process of multi-step-ahead forecasting with standard machine learning algorithms. 'forecastML' supports lagged, dynamic, static, and grouping features for modeling single and grouped numeric or factor/sequence time series. In addition, simple wrapper functions are used to support model-building with most R packages. This approach to forecasting is inspired by Bergmeir, Hyndman, and Koo's (2018) paper ``A note on the validity of cross-validation for evaluating autoregressive time series prediction'' <doi:10.1016/j.csda.2017.11.003>.

License MIT + file LICENSE

URL https://github.com/nredell/forecastML/

Encoding UTF-8

LazyData true

Imports tidyr (>= 0.8.1), rlang (>= 0.4.0), magrittr (>= 1.5), lubridate (>= 1.7.4), ggplot2 (>= 3.1.0), future.apply (>= 1.3.0), methods, purrr (>= 0.3.2), data.table (>= 1.12.6), dtplyr (>= 1.0.0), tibble (>= 2.1.3)

RoxygenNote 7.1.0

Collate 'fill_gaps.R' 'create_windows.R' 'create_skeleton.R' 'combine_forecasts.R' 'lagged_df.R' 'return_error.R' 'return_hyper.R' 'train_model.R' 'reconcile_forecasts.R' 'calculate_intervals.R' 'data_seatbelts.R' 'data_buoy.R' 'data_buoy_gaps.R' 'zzz.R'

Depends R (>= 3.5.0), dplyr (>= 0.8.3)

Suggests glmnet (>= 2.0.16), DT (>= 0.5), knitr (>= 1.22), rmarkdown (>= 1.12.6), xgboost (>= 0.82.1), randomForest (>= 4.6.14), testthat (>= 2.2.1), covr (>= 3.3.1), reticulate (>= 1.15)

VignetteBuilder knitr

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

RemoteUrl https://github.com/nredell/forecastml

RemoteRef HEAD

RemoteSha 282ebe0b7ee62375cf1e71a1fb002be0e57a0a9c

Contents

`calculate_intervals` *Calculate bootstrap prediction intervals for forecasts*

Description

The residuals from model training/fit are sampled i.i.d. for (a) each direct forecast horizon for a single time series and (b) each combination of direct forecast horizon and group for multiple time series.

Usage

```
calculate_intervals(
  forecasts,
  residuals,
  index = NULL,
  outcome = NULL,
  keys = NULL,
  levels = c(0.95),
  times = 100L,
  weights = NULL,
  keep_samples = FALSE
)
```

Arguments

`forecasts`	A data.frame of forecasts.
`residuals`	A data.frame of residuals (e.g., `residuals(data_fit)`)
`index`	Optional for forecasts from `combine_forecasts()`. A string giving the name of the date column in `forecasts`.
`outcome`	Optional for forecasts from `combine_forecasts()`. A string giving the name of the forecast column in `forecasts`.
`keys`	Optional. For grouped time series, a character vector giving the column name(s) of the group columns. The key identifies unique time series of residuals for bootstrap sampling. For direct forecasting, a single time series will have one group per direct forecast horizon.
`levels`	A numeric vector with 1 or more forecast prediction intervals. A level of .95, for example, will return the 0.25 and .975 quantiles of the bootstrapped forecast distribution at each forecast horizon.
`times`	Integer. The number of bootstrap samples.
`weights`	Not implemented.
`keep_samples`	Boolean. If `TRUE`, a data.frame of `times` bootstrapped forecasts is returned in addition to the calculated forecast prediction intervals. The samples are in the list slot named 'samples'.

Value

If `forecasts` is an object of class 'forecast_results', a `forecast_results` object with a new column for each lower- and upper-bound forecast in `levels`. If `forecasts` is a data.frame, the function return will be the same but without `forecastML` attributes. If, `keep_samples` is `TRUE`, a named list of length 2 is returned with 'forecasts' and 'samples'.

Examples

```
## Not run:
  data("data_seatbelts", package = "forecastML")
  data_train <- create_lagged_df(data_seatbelts, type = "train", method = "direct",
                                 outcome_col = 1, lookback = 1:15,
                                 horizons = c(1, 6, 12))

  windows <- create_windows(data_train, window_length = 0)

  model_fn <- function(data) {
    model <- lm(DriversKilled ~ ., data)
  }

  model_results <- train_model(data_train, windows, model_name = "OLS",
                               model_function = model_fn)

  predict_fn <- function(model, data) {
    data_pred <- as.data.frame(predict(model, data))
  }

  data_fit <- predict(model_results, prediction_function = list(predict_fn), data = data_train)

  residuals <- residuals(data_fit)

  data_forecast <- create_lagged_df(data_seatbelts, type = "forecast",
                                    method = "direct", outcome_col = 1,
                                    lookback = 1:15, horizons = c(1, 6, 12))

  data_forecasts <- predict(model_results, prediction_function = list(predict_fn),
                            data = data_forecast)

  data_forecasts <- combine_forecasts(data_forecasts)

  data_forecasts <- calculate_intervals(data_forecasts, residuals, times = 30)

  plot(data_forecasts)

## End(Not run)
```

combine_forecasts *Combine multiple horizon-specific forecast models to produce one forecast*

Description

The horizon-specific models can either be combined to (a) produce final forecasts for only those horizons at which they were trained (i.e., shorter-horizon models override longer-horizon models when producing final short-horizon h-step-ahead forecasts) or (b) produce final forecasts using any combination of horizon-specific models that minimized error over the validation/training dataset.

Usage

```
combine_forecasts(
  ...,
  type = c("horizon", "error"),
  aggregate = stats::median,
  data_error = list(NULL),
  metric = NULL
)
```

Arguments

`...` One or more objects of class 'forecast_results' from running `predict.forecast_model()` on an input forward-looking forecast dataset. These are the forecasts from the horizon-specific direct forecasting models trained over the entire training dataset by setting `create_windows(..., window_length = 0)`. If multiple models are passed in `...` with the same direct forecast horizon, for `type = 'horizon'`, forecasts for the same direct forecast horizon are combined with `aggregate`; for `type = 'error'`, the model that minimizes the error metric at the given direct forecast horizon produces the forecast.

`type` Default: 'horizon'. A character vector of length 1 that identifies the forecast combination method.

`aggregate` Default `median` for `type = 'horizon'`. A function–without parentheses–that aggregates forecasts if more than one model passed in `...` has the same direct forecast horizon and `type = 'horizon']`.

`data_error` Optional. A list of objects of class 'validation_error' from running `return_error()` on a training dataset. The length and order of `data_error` should match the models passed in `...`.

`metric` Required if `data_error` is given. A length 1 character vector naming the forecast error metric used to select the optimal model at each forecast horizon from the models passed in '...' e.g., 'mae'.

Value

An S3 object of class 'forecastML' with final h-step-ahead forecasts.

Forecast combination type:

- `type = 'horizon'`: 1 final h-step-ahead forecast is returned for each model object passed in `...`.
- `type = 'error'`: 1 final h-step-ahead forecast is returned by selecting, for each forecast horizon, the model that minimized the chosen error metric at that horizon on the outer-loop validation data sets.

Columns in returned 'forecastML' data.frame:

- `model`: User-supplied model name in `train_model()`.
- `model_forecast_horizon`: The direct-forecasting time horizon that the model was trained on.
- `horizon`: Forecast horizons, 1:h, measured in dataset rows.

- forecast_period: The forecast period in row indices or dates. The forecast period starts at either attributes(create_lagged_df())$data_stop + 1 for row indices or attributes(create_lagged_df())$d + 1 * frequency for date indices.
- "groups": If given, the user-supplied groups in create_lagged_df().
- "outcome_name"_pred: The final forecasts.
- "outcome_name"_pred_lower: If given, the lower forecast bounds returned by the user-supplied prediction function.
- "outcome_name"_pred_upper: If given, the upper forecast bounds returned by the user-supplied prediction function.

Methods and related functions

The output of combine_forecasts() has the following generic S3 methods

- plot

Examples

```
# Example with "type = 'horizon'".
data("data_seatbelts", package = "forecastML")

horizons <- c(1, 3, 12)
lookback <- 1:15

data_train <- create_lagged_df(data_seatbelts, type = "train", outcome_col = 1,
                               lookback = lookback, horizon = horizons)

windows <- create_windows(data_train, window_length = 0)

model_function <- function(data, my_outcome_col) {
  model <- lm(DriversKilled ~ ., data = data)
  return(model)
}

model_results <- train_model(data_train, windows, model_name = "LM", model_function)

data_forecast <- create_lagged_df(data_seatbelts, type = "forecast", outcome_col = 1,
                                  lookback = lookback, horizon = horizons)

prediction_function <- function(model, data_features) {
  x <- data_features
  data_pred <- data.frame("y_pred" = predict(model, newdata = x))
  return(data_pred)
}

data_forecasts <- predict(model_results, prediction_function = list(prediction_function),
                          data = data_forecast)

data_combined <- combine_forecasts(data_forecasts)

plot(data_combined)
```

create_lagged_df *Create model training and forecasting datasets with lagged, grouped, dynamic, and static features*

Description

Create a list of datasets with lagged, grouped, dynamic, and static features to (a) train forecasting models for specified forecast horizons and (b) forecast into the future with a trained ML model.

Usage

```
create_lagged_df(
  data,
  type = c("train", "forecast"),
  method = c("direct", "multi_output"),
  outcome_col = 1,
  horizons,
  lookback = NULL,
  lookback_control = NULL,
  dates = NULL,
  frequency = NULL,
  dynamic_features = NULL,
  groups = NULL,
  static_features = NULL,
  predict_future = NULL,
  use_future = FALSE,
  keep_rows = FALSE
)
```

Arguments

`data` A data.frame with the (a) target to be forecasted and (b) features/predictors. An optional date column can be given in the `dates` argument (required for grouped time series). Note that ‘`orecastML` only works with regularly spaced date/time intervals and that missing rows–usually due to periods when no data was collected–will result in incorrect feature lags. Use `fill_gaps` to fill in any missing rows/data prior to running this function.

`type` The type of dataset to return–(a) model training or (b) forecast prediction. The default is `train`.

`method` The type of modeling dataset to create. `direct` returns 1 data.frame for each forecast horizon and `multi_output` returns 1 data.frame for simultaneously modeling all forecast horizons. The default is `direct`.

`outcome_col` The column index–an integer–of the target to be forecasted. If `outcome_col != 1`, the outcome column will be moved to position 1 and `outcome_col` will be set to 1 internally.

`horizons` A numeric vector of one or more forecast horizons, h, measured in dataset rows. If `dates` are given, a horizon of 1, for example, would equal 1 * `frequency` in calendar time.

`lookback` A numeric vector giving the lags–in dataset rows–for creating the lagged features. All non-grouping, non-static, and non-dynamic features in the input dataset, `data`, are lagged by the same values. The outcome is also lagged by default. Either `lookback` or `lookback_control` need to be specified–but not both.

`lookback_control`

A list of numeric vectors, specifying potentially unique lags for each feature. The length of the list should equal `ncol(data)` and be ordered the same as the columns in `data`. Lag values for any grouping, static, or dynamic feature columns are automatically coerced to 0 and not lagged. `list(NULL)` `lookback_control` values drop columns from the input dataset. Either `lookback` or `lookback_control` need to be specified–but not both.

`dates` A vector or 1-column data.frame of dates/times with class 'Date' or 'POSIXt'. The length of `dates` should equal `nrow(data)`. Required if `groups` are given.

`frequency` Date/time frequency. Required if `dates` are given. A string taking the same input as `base::seq.Date(..., by = "frequency")` or `base::seq.POSIXt(..., by = "frequency")` e.g., '1 hour', '1 month', '7 days', '10 years' etc. The highest frequency supported at present is '1 sec'.

`dynamic_features`

A character vector of column names that identify features that change through time but which are not lagged (e.g., weekday or year). If `type = "forecast"` and `method = "direct"`, these features will receive `NA` values; though, they can be filled in by the user after running this function.

`groups` A character vector of column names that identify the groups/hierarchies when multiple time series are present. These columns are used as model features but are not lagged. Note that combining feature lags with grouped time series will result in `NA` values throughout the data.

`static_features`

For grouped time series only. A character vector of column names that identify features that do not change through time. These columns are not lagged. If `type = "forecast"`, these features will be filled forward using the most recent value for the group.

`predict_future` When `type = "forecast"`, a function for predicting the future values of any dynamic features. This function takes `data` and `dates` as positional arguments and returns a data.frame with (a) one or more rows, (b) an "index" column of future dates, (c) group columns if needed, and (d) 1 or more columns with name(s) in `dynamic_features`.

`use_future` Boolean. If `TRUE`, the `future.apply` package is used for creating lagged data.frames. `multisession` or `multicore` futures are especially useful for (a) grouped time series with many groups and (b) high-dimensional datasets with many lags per feature. Run `future::plan(future::multiprocess)` prior to this function to set up multissession or multicore parallel dataset creation.

`keep_rows` Boolean. For non-grouped time series, keep the `1:max(lookback)` rows at the beginning of the time series. These rows will contain missing values for lagged features that "look back" before the start of the dataset.

Value

An S3 object of class 'lagged_df' or 'grouped_lagged_df': A list of data.frames with new columns for the lagged/non-lagged features. For `method = "direct"`, the length of the returned list is equal to the number of forecast horizons and is in the order of horizons supplied to the `horizons` argument. Horizon-specific datasets can be accessed with `my_lagged_df$horizon_h` where 'h' gives the forecast horizon. For `method = "multi_output"`, the length of the returned list is 1. Horizon-specific datasets can be accessed with `my_lagged_df$horizon_1_3_5` where "1_3_5" represents the forecast horizons passed in `horizons`.

The contents of the returned data.frames are as follows:

type = 'train', non-grouped: A data.frame with the outcome and lagged/dynamic features.

type = 'train', grouped: A data.frame with the outcome and unlagged grouping columns followed by lagged, dynamic, and static features.

type = 'forecast', non-grouped: (1) An 'index' column giving the row index or date of the forecast periods (e.g., a 100 row non-date-based training dataset would start with an index of 101). (2) A 'horizon' column that indicates the forecast period from `1:max(horizons)`. (3) Lagged features identical to the 'train', non-grouped dataset.

type = 'forecast', grouped: (1) An 'index' column giving the date of the forecast periods. The first forecast date for each group is the maximum date from the `dates` argument + 1 * `frequency` which is the user-supplied date/time frequency.(2) A 'horizon' column that indicates the forecast period from `1:max(horizons)`. (3) Lagged, static, and dynamic features identical to the 'train', grouped dataset.

Attributes

- `names`: The horizon-specific datasets that can be accessed with `my_lagged_df$horizon_h`.
- `type`: Training, `train`, or forecasting, `forecast`, dataset(s).
- `method`: `direct` or `multi_output`.
- `horizons`: Forecast horizons measured in dataset rows.
- `outcome_col`: The column index of the target being forecasted.
- `outcome_cols`: If `method = multi_output`, the column indices of the multiple outputs in the transformed dataset.
- `outcome_name`: The name of the target being forecasted.
- `outcome_names`: If `method = multi_output`, the column names of the multiple outputs in the transformed dataset. The names take the form "outcome_name_h" where 'h' is a horizon passed in `horizons`.
- `predictor_names`: The predictor or feature names from the input dataset.
- `row_indices`: The `row.names()` of the output dataset. For non-grouped datasets, the first `lookback` + 1 rows are removed from the beginning of the dataset to remove `NA` values in the lagged features.
- `date_indices`: If `dates` are given, the vector of `dates`.
- `frequency`: If `dates` are given, the date/time frequency.
- `data_start`: `min(row_indices)` or `min(date_indices)`.

- data_stop: max(row_indices) or max(date_indices).
- groups: If groups are given, a vector of group names.
- class: grouped_lagged_df, lagged_df, list

Methods and related functions

The output of create_lagged_df() is passed into

- create_windows

and has the following generic S3 methods

- summary
- plot

Examples

```
# Sampled Seatbelts data from the R package datasets.
data("data_seatbelts", package = "forecastML")
#------------------------------------------------------------------------------
# Example 1 - Training data for 2 horizon-specific models w/ common lags per predictor.
horizons <- c(1, 12)
lookback <- 1:15

data <- data_seatbelts

data_train <- create_lagged_df(data_seatbelts, type = "train", outcome_col = 1,
                               horizons = horizons, lookback = lookback)
head(data_train[[length(horizons)]])

# Example 1 - Forecasting dataset
# The last 'nrow(data_seatbelts) - horizon' rows are automatically used from data_seatbelts.
data_forecast <- create_lagged_df(data_seatbelts, type = "forecast", outcome_col = 1,
                                  horizons = horizons, lookback = lookback)
head(data_forecast[[length(horizons)]])

#------------------------------------------------------------------------------
# Example 2 - Training data for one 3-month horizon model w/ unique lags per predictor.
horizons <- 3
lookback <- list(c(3, 6, 9, 12), c(4:12), c(6:15), c(8))

data_train <- create_lagged_df(data_seatbelts, type = "train", outcome_col = 1,
                               horizons = horizons, lookback_control = lookback)
head(data_train[[length(horizons)]])
```

create_skeleton — *Remove the features from a lagged training dataset to reduce memory consumption*

Description

create_skeleton() strips the feature data from a create_lagged_df() object but keeps the outcome column(s), any grouping columns, and meta-data which allows the resulting lagged_df to be used downstream in the forecastML pipeline. The main benefit is that the custom modeling function passed in train_model() can read data directly from the disk or a database when the dataset is too large to fit into memory.

Usage

```
create_skeleton(lagged_df)
```

Arguments

lagged_df	An object of class 'lagged_df' from create_lagged_df(..., type = 'train').

Value

An S3 object of class 'lagged_df' or 'grouped_lagged_df': A list of data.frames with the outcome column(s) and any grouping columns but with all other features removed. A special attribute skeleton = TRUE is added.

Methods and related functions

The output of create_skeleton can be passed into

- create_windows

create_windows — *Create time-contiguous validation datasets for model evaluation*

Description

Flexibly create blocks of time-contiguous validation datasets to assess the forecast accuracy of trained models at various times in the past. These validation datasets are similar to the outer loop of a nested cross-validation model training setup.

Usage

```
create_windows(
  lagged_df,
  window_length = 12L,
  window_start = NULL,
  window_stop = NULL,
  skip = 0,
  include_partial_window = TRUE
)
```

Arguments

`lagged_df` An object of class 'lagged_df' or 'grouped_lagged_df' from `create_lagged_df`.

`window_length` An integer that defines the length of the contiguous validation dataset in dataset rows/dates. If dates were given in `create_lagged_df()`, the validation window is 'window_length' * 'date frequency' in calendar time. Setting `window_length = 0` trains the model on (a) the entire dataset or (b) between a single `window_start` and `window_stop` value. Specifying multiple `window_start` and `window_stop` values with vectors of length > 1 overrides `window_length`.

`window_start` Optional. A row index or date identifying the row/date to start creating contiguous validation datasets. A vector of start rows/dates can be supplied for greater control. The length and order of `window_start` should match `window_stop`. If `length(window_start)` > 1, `window_length`, `skip`, and `include_partial_window` are ignored.

`window_stop` Optional. An index or date identifying the row/date to stop creating contiguous validation datasets. A vector of start rows/dates can be supplied for greater control. The length and order of `window_stop` should match `window_start`. If `length(window_stop)` > 1, `window_length`, `skip`, and `include_partial_window` are ignored.

`skip` An integer giving a fixed number of dataset rows/dates to skip between validation datasets. If dates were given in `create_lagged_df()`, the time between validation windows is `skip` * 'date frequency'.

`include_partial_window` Boolean. If `TRUE`, keep validation datasets that are shorter than `window_length`.

Value

An S3 object of class 'windows': A data.frame giving the indices for the validation datasets.

Methods and related functions

The output of `create_windows()` is passed into

- `train_model`

and has the following generic S3 methods

- `plot`

Examples

```
# Sampled Seatbelts data from the R package datasets.
data("data_seatbelts", package = "forecastML")

# Example - Training data for 2 horizon-specific models w/ common lags per feature.
horizons <- c(1, 12)
lookback <- 1:15

data_train <- create_lagged_df(data_seatbelts, type = "train", outcome_col = 1,
                               lookback = lookback, horizon = horizons)

# All historical window lengths of 12 plus any partial windows at the end of the dataset.
windows <- create_windows(data_train, window_length = 12)
windows

# Two custom validation windows with different lengths.
windows <- create_windows(data_train, window_start = c(20, 80), window_stop = c(30, 100))
windows
```

---

data_buoy *NOAA buoy weather data*

Description

A dataset containing daily average sensor measurements of several environmental conditions collected by 14 buoys in Lake Michigan from 2012 through 2018.

Usage

```
data_buoy
```

Format

A data.frame with 30,821 rows and 9 columns:

date date

wind_spd average daily wind speed in kts

buoy_id the station ID for each buoy

lat latitude

lon longitude

day day of year

year calendar year

air_temperature air temperature in degrees Fahrenheit

sea_surface_temperature water temperature in degrees Fahrenheit

Source

http://www.ndbc.noaa.gov/

`data_buoy_gaps` *NOAA buoy weather data*

Description

A dataset containing daily average sensor measurements of several environmental conditions collected by 14 buoys in Lake Michigan from 2012 through 2018. This dataset is identical to the data_buoy dataset except that there are gaps in the daily sensor data. Running `fill_gaps()` on `data_buoy_gaps` will produce `data_buoy`.

Usage

```
data_buoy_gaps
```

Format

A data.frame with 23,646 rows and 9 columns:

date date

wind_spd average daily wind speed in kts

buoy_id the station ID for each buoy

lat latitude

lon longitude

day day of year

year calendar year

air_temperature air temperature in degrees Fahrenheit

sea_surface_temperature water temperature in degrees Fahrenheit

Source

`http://www.ndbc.noaa.gov/`

`data_seatbelts` *Road Casualties in Great Britain 1969-84*

Description

This is the `Seatbelts` dataset from the `datasets` package.

Usage

```
data_seatbelts
```

Format

A data.frame with 192 rows and 8 columns

Source

Harvey, A.C. (1989). Forecasting, Structural Time Series Models and the Kalman Filter. Cambridge University Press, pp. 519–523.

Durbin, J. and Koopman, S. J. (2001). Time Series Analysis by State Space Methods. Oxford University Press.

`https://stat.ethz.ch/R-manual/R-devel/library/datasets/html/UKDriverDeaths.html`

`fill_gaps` *Prepare a dataset for modeling by filling in temporal gaps in data collection*

Description

In order to create a modeling dataset with feature lags that are temporally correct, the entry function in `forecastML`, `create_lagged_df`, needs evenly-spaced time series with no gaps in data collection. `fill_gaps()` can help here. This function takes a `data.frame` with (a) dates, (b) the outcome being forecasted, and, optionally, (c) dynamic features that change through time, (d) group columns for multiple time series modeling, and (e) static or non-dynamic features for multiple time series modeling and returns a `data.frame` with rows evenly spaced in time. Specifically, this function adds rows to the input dataset while filling in (a) dates, (b) grouping information, and (c) static features. The (a) outcome and (b) dynamic features will be `NA` for any missing time periods; these `NA` values can be left as-is, user-imputed, or removed from modeling in the user-supplied modeling wrapper function for `train_model`.

Usage

```
fill_gaps(data, date_col = 1, frequency, groups = NULL, static_features = NULL)
```

Arguments

`data` A data.frame or object coercible to a data.frame with, minimally, dates and the outcome being forecasted.

`date_col` The column index–an integer–of the date index. This column should have class 'Date' or 'POSIXt'.

`frequency` Date/time frequency. A string taking the same input as `base::seq.Date(..., by = "frequency")` or `base::seq.POSIXt..., by = "frequency")` e.g., '1 hour', '1 month', '7 days', '10 years' etc. The highest frequency supported at present is '1 sec'.

`groups` Optional. A character vector of column names that identify the unique time series (i.e., groups/hierarchies) when multiple time series are present.

`static_features`

Optional. For grouped time series only. A character vector of column names that identify features that do not change through time. These columns are expected to be used as model features but are not lagged (e.g., a ZIP code column). The most recent values for each static feature for each group are used to fill in the resulting missing data in static features when new rows are added to the dataset.

Value

An object of class 'data.frame': The returned data.frame has the same number of columns and column order but with additional rows to account for gaps in data collection. For grouped data, any new rows added to the returned data.frame will appear between the minimum–or oldest–date for that group and the maximum–or most recent–date across all groups. If the user-supplied forecasting algorithm(s) cannot handle missing outcome values or missing dynamic features, these should either be imputed prior to `create_lagged_df()` or filtered out in the user-supplied modeling function for `train_model`.

Methods and related functions

The output of `fill_gaps()` is passed into

- `create_lagged_df`

Examples

```
# NOAA buoy dataset with gaps in data collection
data("data_buoy_gaps", package = "forecastML")

data_buoy_no_gaps <- fill_gaps(data_buoy_gaps, date_col = 1, frequency = '1 day',
                               groups = 'buoy_id', static_features = c('lat', 'lon'))

# The returned data.frame has the same number of columns but the time-series
# are now evenly spaced at 1 day apart. Additionally, the unchanging grouping
# columns and static features columns have been filled in for the newly created dataset rows.
dim(data_buoy_gaps)
dim(data_buoy_no_gaps)

# Running create_lagged_df() is the next step in the forecastML forecasting
# process. If there are long gaps in data collection, like in this buoy dataset,
# and the user-supplied modeling algorithm cannot handle missing outcomes data,
# the best option is to filter these rows out in the user-supplied modeling function
# for train_model()
```

`plot.forecastML` *Plot an object of class 'forecastML'*

Description

A forecast plot of h-step-ahead forecasts produced from multiple horizon-specific forecast models using `combine_forecasts()`.

Usage

```
## S3 method for class 'forecastML'
plot(
  x,
  data_actual = NULL,
  actual_indices = NULL,
  facet = ~model,
  models = NULL,
  group_filter = NULL,
  drop_facet = FALSE,
  interval_fill = NULL,
  interval_alpha = NULL,
  ...
)
```

Arguments

`x` An object of class 'forecastML' from `combine_forecasts()`.

`data_actual` A data.frame containing the target/outcome name and any grouping columns. The data can be historical actuals and/or holdout/test data.

`actual_indices` Required if `data_actual` is given. A vector or 1-column data.frame of numeric row indices or dates (class 'Date' or 'POSIXt') with length `nrow(data_actual)`. The data can be historical actuals and/or holdout/test data.

`facet` Optional. A formula with any combination of `model`, or `group` (for grouped time series) passed to `ggplot2::facet_grid()` internally (e.g., `~ model`, `model ~ .`, `~ model + group`).

`models` Optional. Filter results by user-defined model name from `train_model()`.

`group_filter` Optional. A string for filtering plot results for grouped time-series (e.g., `"group_col_1 == 'A'"`); passed to `dplyr::filter()` internally.

`drop_facet` Optional. Boolean. If actuals are given when forecasting factors, the plot facet with 'actual' data can be dropped.

`interval_fill` A character vector of color names or hex codes to fill the prediction intervals. For intervals with multiple levels, the first color corresponds to the fill with the widest interval.

`interval_alpha` A numeric vector of alpha values to shade the prediction intervals. For intervals with multiple levels, the first value corresponds to the shading with the widest interval.

`...` Not used.

Value

Forecast plot of class 'ggplot'.

plot.forecast_error *Plot forecast error*

Description

Plot forecast error at various levels of aggregation.

Usage

```
## S3 method for class 'forecast_error'
plot(
  x,
  type = c("global"),
  metric = NULL,
  facet = NULL,
  models = NULL,
  horizons = NULL,
  windows = NULL,
  group_filter = NULL,
  ...
)
```

Arguments

x An object of class 'forecast_error' from return_error().

type Select plot type; type = "global" is the default plot.

metric Select error metric to plot (e.g., "mae"); attributes(x)$error_metrics[1] is the default metric.

facet Optional. A formula with any combination of horizon, model, or group (for grouped time series). passed to ggplot2::facet_grid() internally (e.g., horizon ~ model, horizon + model ~ ., ~ horizon + group). Can be NULL. The default faceting is set internally depending on the plot type.

models Optional. A vector of user-defined model names from train_model() to filter results.

horizons Optional. A numeric vector to filter results by horizon.

windows Optional. A numeric vector to filter results by validation window number.

group_filter A string for filtering plot results for grouped time series (e.g., "group_col_1 == 'A'").

... Not used.

Value

Forecast error plots of class 'ggplot'.

`plot.forecast_model_hyper` *Plot hyperparameters*

Description

Plot hyperparameter stability and relationship with error metrics across validation datasets and horizons.

Usage

```
## S3 method for class 'forecast_model_hyper'
plot(
  x,
  data_results,
  data_error,
  type = c(”stability”, ”error”),
  horizons = NULL,
  windows = NULL,
  ...
)
```

Arguments

`x` An object of class ’forecast_model_hyper’ from `return_hyper()`.

`data_results` An object of class ’training_results’ from `predict.forecast_model()`.

`data_error` An object of class ’validation_error’ from `return_error()`.

`type` Select plot type; ’stability’ is the default.

`horizons` Optional. A numeric vector to filter results by horizon.

`windows` Optional. A numeric vector to filter results by validation window number.

`...` Not used.

Value

Hyper-parameter plots of class ’ggplot’.

Examples

```
# Sampled Seatbelts data from the R package datasets.
data(”data_seatbelts”, package = ”forecastML”)

# Example - Training data for 2 horizon-specific models w/ common lags per predictor.
horizons <- c(1, 12)
lookback <- 1:15

data_train <- create_lagged_df(data_seatbelts, type = ”train”, outcome_col = 1,
                               lookback = lookback, horizon = horizons)

# One custom validation window at the end of the dataset.
windows <- create_windows(data_train, window_start = 181, window_stop = 192)

# User-define model - LASSO
# A user-defined wrapper function for model training that takes the following
# arguments: (1) a horizon-specific data.frame made with create_lagged_df(..., type = "train")
# (e.g., my_lagged_df$horizon_h) and, optionally, (2) any number of additional named arguments
# which are passed as '...' in train_model().
library(glmnet)
model_function <- function(data, my_outcome_col) {

  x <- data[, -(my_outcome_col), drop = FALSE]
  y <- data[, my_outcome_col, drop = FALSE]
  x <- as.matrix(x, ncol = ncol(x))
  y <- as.matrix(y, ncol = ncol(y))

  model <- glmnet::cv.glmnet(x, y, nfolds = 3)
  return(model)
}

# my_outcome_col = 1 is passed in ... but could have been defined in model_function().
model_results <- train_model(data_train, windows, model_name = "LASSO", model_function,
                             my_outcome_col = 1)

# User-defined prediction function - LASSO
# The predict() wrapper takes two positional arguments. First,
# the returned model from the user-defined modeling function (model_function() above).
# Second, a data.frame of predictors--identical to the datasets returned from
# create_lagged_df(..., type = "train"). The function can return a 1- or 3-column data.frame
# with either (a) point forecasts or (b) point forecasts plus lower and upper forecast
# bounds (column order and column names do not matter).
prediction_function <- function(model, data_features) {

  x <- as.matrix(data_features, ncol = ncol(data_features))

  data_pred <- data.frame("y_pred" = predict(model, x, s = "lambda.min"))
  return(data_pred)
}

# Predict on the validation datasets.
data_valid <- predict(model_results, prediction_function = list(prediction_function),
                      data = data_train)

# User-defined hyperparameter function - LASSO
# The hyperparameter function should take one positional argument--the returned model
# from the user-defined modeling function (model_function() above). It should
# return a 1-row data.frame of the optimal hyperparameters.
hyper_function <- function(model) {

  lambda_min <- model$lambda.min
  lambda_1se <- model$lambda.1se
  data_hyper <- data.frame(”lambda_min” = lambda_min, ”lambda_1se” = lambda_1se)
  return(data_hyper)
}

data_error <- return_error(data_valid)

data_hyper <- return_hyper(model_results, hyper_function)

plot(data_hyper, data_valid, data_error, type = ”stability”, horizons = c(1, 12))
```

plot.forecast_results *Plot an object of class forecast_results*

Description

A forecast plot for each horizon for each model in `predict.forecast_model()`.

Usage

```
## S3 method for class 'forecast_results'
plot(
  x,
  data_actual = NULL,
  actual_indices = NULL,
  facet = horizon ~ model,
  models = NULL,
  horizons = NULL,
  windows = NULL,
  group_filter = NULL,
  ...
)
```

Arguments

`x` An object of class ’forecast_results’ from `predict.forecast_model()`.

`data_actual` A data.frame containing the target/outcome name and any grouping columns. The data can be historical actuals and/or holdout/test data.

`actual_indices` Required if `data_actual` is given. A vector or 1-column data.frame of numeric row indices or dates (class ’Date’ or ’POSIXt’) with length `nrow(data_actual)`. The data can be historical actuals and/or holdout/test data.

`facet` Optional. For numeric outcomes, a formula with any combination of `horizon`, `model`, or `group` (for grouped time series) passed to `ggplot2::facet_grid()` internally (e.g., `horizon` ~ `model`, `horizon` + `model` ~ `.`, ~ `horizon` + group). Can be `NULL`.

`models` Optional. Filter results by user-defined model name from `train_model()`.

`horizons` Optional. Filter results by horizon.

`windows` Optional. Filter results by validation window number.

`group_filter` Optional. A string for filtering plot results for grouped time-series (e.g., `"group_col_1 == 'A'"`); passed to `dplyr::filter()` internally.

`...` Not used.

Value

Forecast plot of class 'ggplot'.

`plot.lagged_df` *Plot datasets with lagged features*

Description

Plot datasets with lagged features to view ther direct forecasting setup across horizons.

Usage

```
## S3 method for class 'lagged_df'
plot(x, ...)
```

Arguments

`x` An object of class 'lagged_df' from `create_lagged_df()`.

`...` Not used.

Value

A single plot of class 'ggplot' if `lookback` was specified in `create_lagged_df()`; a list of plots, one per feature, of class 'ggplot' if `lookback_control` was specified.

Examples

```
# Sampled Seatbelts data from the R package datasets.
data("data_seatbelts", package = "forecastML")
#------------------------------------------------------------------------------
# Example 1 - Training data for 3 horizon-specific models w/ common lags per predictor.
horizons <- c(1, 6, 12)
lookback <- 1:15

data_train <- create_lagged_df(data_seatbelts, type = "train", outcome_col = 1,
                               lookback = lookback, horizon = horizons)
plot(data_train)
#------------------------------------------------------------------------------
# Example 2 - Training data for one 3-month horizon model w/ unique lags per predictor.
horizons <- 3
lookback <- list(c(3, 6, 9, 12), c(4:12), c(6:15), c(8))
```

```
data_train <- create_lagged_df(data_seatbelts, type = "train", outcome_col = 1,
                               lookback_control = lookback, horizon = horizons)
plot(data_train)
```

`plot.training_results` *Plot an object of class training_results*

Description

Several diagnostic plots can be returned to assess the quality of the forecasts based on predictions on the validation datasets.

Usage

```
## S3 method for class 'training_results'
plot(
  x,
  type = c("prediction", "residual", "forecast_stability"),
  facet = horizon ~ model,
  models = NULL,
  horizons = NULL,
  windows = NULL,
  valid_indices = NULL,
  group_filter = NULL,
  keep_missing = FALSE,
  ...
)
```

Arguments

`x` An object of class 'training_results' from `predict.forecast_model()`.

`type` Plot type. The default plot is "prediction" for validation dataset predictions.

`facet` Optional. For numeric outcomes, a formula with any combination of `horizon`, `model`, or `group` (for grouped time series) passed to `ggplot2::facet_grid()` internally (e.g., `horizon` ~ `model`, `horizon` + `model` ~ `.`, ~ `horizon` + `group`).

`models` Optional. Filter results by user-defined model name from `train_model()`.

`horizons` Optional. A numeric vector of model forecast horizons to filter results by horizon-specific model.

`windows` Optional. A numeric vector of window numbers to filter results.

`valid_indices` Optional. A numeric or date vector to filter results by validation row indices or dates.

`group_filter` Optional. A string for filtering plot results for grouped time series (e.g., `"group_col_1 == 'A'"`). The results are passed to `dplyr::filter()` internally.

`keep_missing` Boolean. If `TRUE`, predictions are plotted for indices/dates where the outcome is missing.

`...` Not used.

Value

Diagnostic plots of class 'ggplot'.

`plot.validation_error` *Plot validation dataset forecast error*

Description

Plot forecast error at various levels of aggregation across validation datasets.

Usage

```
## S3 method for class 'validation_error'
plot(
  x,
  type = c("window", "horizon", "global"),
  metric = NULL,
  facet = NULL,
  models = NULL,
  horizons = NULL,
  windows = NULL,
  group_filter = NULL,
  ...
)
```

Arguments

`x` An object of class 'validation_error' from `return_error()`.

`type` Select plot type; `type = "window"` is the default plot.

`metric` Select error metric to plot (e.g., "mae"); `attributes(x)$error_metrics[1]` is the default metric.

`facet` Optional. A formula with any combination of `horizon`, `model`, or `group` (for grouped time series). passed to `ggplot2::facet_grid()` internally (e.g., `horizon ~ model`, `horizon + model ~ .`, `~ horizon + group`). Can be `NULL`. The default faceting is set internally depending on the plot `type`.

`models` Optional. A vector of user-defined model names from `train_model()` to filter results.

`horizons` Optional. A numeric vector to filter results by horizon.

`windows` Optional. A numeric vector to filter results by validation window number.

`group_filter` A string for filtering plot results for grouped time series (e.g., `"group_col_1 == 'A'"`).

`...` Not used.

Value

Forecast error plots of class 'ggplot'.

plot.windows *Plot validation datasets*

Description

Plot validation datasets across time.

Usage

```
## S3 method for class 'windows'
plot(x, lagged_df, show_labels = TRUE, group_filter = NULL, ...)
```

Arguments

`x`	An object of class 'windows' from `create_windows()`.
`lagged_df`	An object of class 'lagged_df' from `create_lagged_df()`.
`show_labels`	Boolean. If `TRUE`, show validation dataset IDs on the plot.
`group_filter`	Optional. A string for filtering plot results for grouped time series (e.g., `"group_col_1 == 'A'"`). This string is passed to `dplyr::filter()` internally.
`...`	Not used.

Value

A plot of the outer-loop nested cross-validation windows of class 'ggplot'.

Examples

```
# Sampled Seatbelts data from the R package datasets.
data("data_seatbelts", package = "forecastML")

# Example - Training data for 3 horizon-specific models w/ common lags per predictor.
horizons <- c(1, 6, 12)
lookback <- 1:15

data_train <- create_lagged_df(data_seatbelts, type = "train", outcome_col = 1,
                               lookback = lookback, horizon = horizons)

# All historical window lengths of 12 plus any partial windows at the end of the dataset.
windows <- create_windows(data_train, window_length = 12)
plot(windows, data_train)

# Two custom validation windows with different lengths.
windows <- create_windows(data_train, window_start = c(20, 80), window_stop = c(30, 100))
plot(windows, data_train)
```

predict.forecast_model

Predict on validation datasets or forecast

Description

Predict with a 'forecast_model' object from `train_model()`. If `data = create_lagged_df(..., type = "train")`, predictions are returned for the outer-loop nested cross-validation datasets. If `data` is an object of class 'lagged_df' from `create_lagged_df(..., type = "forecast")`, predictions are returned for the horizons specified in `create_lagged_df(horizons = ...)`.

Usage

```
## S3 method for class 'forecast_model'
predict(..., prediction_function = list(NULL), data)
```

Arguments

`...` One or more trained models from `train_model()`.

`prediction_function` A list of user-defined prediction functions with length equal to the number of models supplied in `...`. The prediction functions take 2 required positional arguments–(1) a 'forecast_model' object from `train_model()` and (2) a data.frame of model features from `create_lagged_df()`. For numeric outcomes and `method = "direct"`, the function should `return()` 1- or 3-column data.frame of model predictions. If the prediction function returns a 1-column data.frame, point forecasts are assumed. If the prediction function returns a 3-column data.frame, lower and upper forecast bounds are assumed (the order and names of the 3 columns does not matter). For factor outcomes and `method = "direct"`, the function should `return()` (1) 1-column data.frame of the model-predicted factor level or (2) an L-column data.frame of class probabilities where 'L' equals the number of levels in the outcome; columns should be ordered, from left to right, the same as `levels(data$outcome)` which is the default behavior for most `predict(..., type = "prob")` functions. Column names do not matter. For numeric outcomes and `method = "multi_output"`, the function should `return()` and h-column data.frame of model predictions–1 column for each horizon. Forecast intervals and factor outcomes are not currently supported with `method = "multi_output"`.

`data` If `data` is a training dataset from `create_lagged_df(..., type = "train")`, validation dataset predictions are returned; else, if `data` is a forecasting dataset from `create_lagged_df(..., type = "forecast")`, forecasts from horizons 1:h are returned.

Value

If `data = create_lagged_df(..., type = "forecast")`, an S3 object of class 'training_results'.
If `data = create_lagged_df(..., type = "forecast")`, an S3 object of class 'forecast_results'.

Columns in returned ’training_results’ data.frame:

- `model`: User-supplied model name in `train_model()`.
- `model_forecast_horizon`: The direct-forecasting time horizon that the model was trained on.
- `window_length`: Validation window length measured in dataset rows.
- `window_number`: Validation dataset number.
- `valid_indices`: Validation dataset row names from `attributes(create_lagged_df())$row_indices`.
- `date_indices`: If given and `method = "direct"`, validation dataset date indices from `attributes(create_lagged_df` If given and `method = "multi_output"`, date_indices represents the date of the forecast.
- `"groups"`: If given, the user-supplied groups in `create_lagged_df()`.
- `"outcome_name"`: The target being forecasted.
- `"outcome_name"_pred`: The model predictions.
- `"outcome_name"_pred_lower`: If given, the lower prediction bounds returned by the user-supplied prediction function.
- `"outcome_name"_pred_upper`: If given, the upper prediction bounds returned by the user-supplied prediction function.
- `forecast_indices`: If `method = "multi_output"`, the validation index of the h-step-ahead forecast.
- `forecast_date_indices`: If `method = "multi_output"`, the validation date index of the h-step-ahead forecast.

Columns in returned ’forecast_results’ data.frame:

- `model`: User-supplied model name in `train_model()`.
- `model_forecast_horizon`: If `method = "direct"`, the direct-forecasting time horizon that the model was trained on.
- `horizon`: Forecast horizons, 1:h, measured in dataset rows.
- `window_length`: Validation window length measured in dataset rows.
- `forecast_period`: The forecast period in row indices or dates. The forecast period starts at either `attributes(create_lagged_df())$data_stop + 1` for row indices or `attributes(create_lagged_df())$d` `+ 1 * frequency` for date indices.
- `"groups"`: If given, the user-supplied groups in `create_lagged_df()`.
- `"outcome_name"`: The target being forecasted.
- `"outcome_name"_pred`: The model forecasts.
- `"outcome_name"_pred_lower`: If given, the lower forecast bounds returned by the user-supplied prediction function.
- `"outcome_name"_pred_upper`: If given, the upper forecast bounds returned by the user-supplied prediction function.

Examples

```
# Sampled Seatbelts data from the R package datasets.
data("data_seatbelts", package = "forecastML")

# Example - Training data for 2 horizon-specific models w/ common lags per predictor.
horizons <- c(1, 12)
lookback <- 1:15

data_train <- create_lagged_df(data_seatbelts, type = "train", outcome_col = 1,
                               lookback = lookback, horizon = horizons)

# One custom validation window at the end of the dataset.
windows <- create_windows(data_train, window_start = 181, window_stop = 192)

# User-define model - LASSO
# A user-defined wrapper function for model training that takes the following
# arguments: (1) a horizon-specific data.frame made with create_lagged_df(..., type = "train")
# (e.g., my_lagged_df$horizon_h) and, optionally, (2) any number of additional named arguments
# which are passed as '...' in train_model().
library(glmnet)
model_function <- function(data, my_outcome_col) {

  x <- data[, -(my_outcome_col), drop = FALSE]
  y <- data[, my_outcome_col, drop = FALSE]
  x <- as.matrix(x, ncol = ncol(x))
  y <- as.matrix(y, ncol = ncol(y))

  model <- glmnet::cv.glmnet(x, y, nfolds = 3)
  return(model)
}

# my_outcome_col = 1 is passed in ... but could have been defined in model_function().
model_results <- train_model(data_train, windows, model_name = "LASSO", model_function,
                             my_outcome_col = 1)

# User-defined prediction function - LASSO
# The predict() wrapper takes two positional arguments. First,
# the returned model from the user-defined modeling function (model_function() above).
# Second, a data.frame of predictors--identical to the datasets returned from
# create_lagged_df(..., type = "train"). The function can return a 1- or 3-column data.frame
# with either (a) point forecasts or (b) point forecasts plus lower and upper forecast
# bounds (column order and column names do not matter).
prediction_function <- function(model, data_features) {

  x <- as.matrix(data_features, ncol = ncol(data_features))

  data_pred <- data.frame("y_pred" = predict(model, x, s = "lambda.min"))
  return(data_pred)
}

# Predict on the validation datasets.
data_valid <- predict(model_results, prediction_function = list(prediction_function),
                        data = data_train)

# Forecast.
data_forecast <- create_lagged_df(data_seatbelts, type = "forecast", outcome_col = 1,
                                  lookback = lookback, horizon = horizons)

data_forecasts <- predict(model_results, prediction_function = list(prediction_function),
                          data = data_forecast)
```

`reconcile_forecasts` *Reconcile multiple temporal or hierarchical forecasts*

Description

The purpose of forecast reconciliation is to produce a single coherent forecast from multiple forecasts produced at (a) different time horizons (e.g., monthly and quarterly) and/or (b) different levels of aggregation (e.g., classroom, school, and school district). After forecast reconciliation, the bottom-level or most disaggregated forecast can simply be summed up to produce all higher-level forecasts.

Usage

```
reconcile_forecasts(
  forecasts,
  frequency,
  index,
  outcome,
  keys = NULL,
  method,
  keep_all = TRUE,
  keep_non_reconciled = FALSE
)
```

Arguments

`forecasts` A list of 2 or more dataframes with forecasts. Each dataframe must have a date column named `index` of class `Date` or `POSIXt` and a forecast column named `outcome` of class `numeric`. Forecasts should be sorted from oldest (top) to newest (bottom).

`frequency` A character vector of `length(forecasts)` that identifies the date/time frequency of the forecast. Each string should work with `base::seq.Date(..., by = "frequency")` or `base::seq.POSIXt(..., by = "frequency")` e.g., '1 hour', '1 month', '7 days', '10 years' etc.

`index` A string giving the column name of the date column which should be common across `forecasts`.

`outcome` A string giving the column name of the forecast which should be common across `forecasts`.

`keys` Optional. For forecast reconciliation across groups, a `unique()` vector of column names listing all of the keys that identify a distinct time series across the datasets in `forecasts`. If not specified, all columns that are not in `index` or `outcome` are treated as grouping keys for each dataset in `forecasts`.

`method` One of `c("temporal", "group")`. See the Implementation section for details.

`keep_all` Boolean. For `method = "temporal"`. If `TRUE`, reconciled forecasts at all levels are returned. If `FALSE`, only the bottom-level or most disaggregated forecast is returned which can be manually aggregated as needed.

`keep_non_reconciled`

Boolean. For `method = "temporal"`. If `TRUE`, any additional higher frequency forecasts that fell outside of the date range of the lowest frequency forecast are returned with their same forecast value from `forecasts`.

Value

A `data.frame` of reconciled forecasts.

Implementation

- **method = 'temporal'**: Forecasts are reconciled across forecast horizons.
  - Structural scaling with weights from temporal hierarchies from Athanasopoulos et al. (2017).
  - To produce correct forecast reconciliations, all forecasts at the lowest/disaggregated level should be present for all horizons contained in the forecasts with the higher levels of aggregation (e.g., 24 monthly forecasts for 2 annual forecasts or 21 daily forecasts for 3 weekly forecasts).
- **method = 'group'**: Forecasts are reconciled across groups independently at each forecast horizon.
  - Structural scaling from Hyndman et al. (2011).
  - A key column is not needed for the forecast at the highest level of aggregation.
  - Having input forecasts at each level of aggregation is not a requirement. For example, forecasts by nation, state, and city could be reconciled with only 2 input forecasts: 1 for nation (highest aggregation) and 1 for the combination of nation by state by city (lowest/no aggregation) without the 2 intermediate-level forecasts at the state and city levels.

References

Athanasopoulos, G., Hyndman, R. J., Kourentzes, N., & Petropoulos, F. (2017). Forecasting with temporal hierarchies. European Journal of Operational Research, 262(1), 60-74. https://robjhyndman.com/papers/temporalhierarchies.pdf

Hyndman, R. J., Ahmed, R. A., Athanasopoulos, G., & Shang, H. L. (2011). Optimal combination forecasts for hierarchical time series. Computational statistics & data analysis, 55(9), 2579-2589. http://robjhyndman.com/papers/hierarchical

Examples

```
#------------------------------------------------------------------------------
# Temporal example 1: 2 forecasts, daily/monthly, 2 forecast periods at highest aggregation.
freq <- c("1 day", "1 month")

data_1_day <- data.frame("index" = seq(as.Date("2020-1-1"), as.Date("2020-2-29"), by = freq[1]),
                         "forecast" = c(rep(5, 31), rep(7, 29)))

data_1_month <- data.frame("index" = seq(as.Date("2020-1-1"), as.Date("2020-2-1"), by = freq[2]),
                           "forecast" = c(150, 200))

forecasts_reconciled <- reconcile_forecasts(list(data_1_day, data_1_month), freq,
                                            index = "index", outcome = "forecast",
                                            method = "temporal")
#------------------------------------------------------------------------------
# Temporal example 2: 3 forecasts, monthly/4-monthly/annually, 1 forecast period at highest aggregation.
freq <- c("1 month", "4 months", "1 year")

data_1_month <- data.frame("index" = seq(as.Date("2020-1-1"), as.Date("2020-12-1"), by = freq[1]),
                           "forecast" = rep(10, 12))

data_4_months <- data.frame("index" = seq(as.Date("2020-1-1"), as.Date("2020-12-1"), by = freq[2]),
                            "forecast" = c(40, 50, 45))

data_1_year <- data.frame("index" = as.Date("2020-01-01"),
                          "forecast" = c(110))

forecasts_reconciled <- reconcile_forecasts(list(data_1_month, data_4_months, data_1_year), freq,
                                            index = "index", outcome = "forecast",
                                            method = "temporal")
#------------------------------------------------------------------------------
# Temporal example 3: 2 forecasts, weekly/monthly, 2 forecast periods at highest aggregation.
freq <- c("1 week", "1 month")

data_1_week <- data.frame("index" = seq(as.Date("2020-1-1"), as.Date("2020-3-1"), by = freq[1]),
                          "forecast" = c(rep(3, 5), rep(2, 4)))

data_1_month <- data.frame("index" = seq(as.Date("2020-1-1"), as.Date("2020-2-1"), by = freq[2]),
                           "forecast" = c(11, 12))

forecasts_reconciled <- reconcile_forecasts(list(data_1_week, data_1_month), freq,
                                            index = "index", outcome = "forecast",
                                            method = "temporal")
#------------------------------------------------------------------------------
# Temporal example 4: 2 forecasts, hourly/daily, 3 forecast periods at highest aggregation.
freq <- c("1 hour", "1 day")
timezone <- "UTC"

data_1_hour <- data.frame("index" = seq(as.POSIXct("2020-01-01 00:00:00", tz = timezone),
                                       as.POSIXct("2020-01-03 23:00:00", tz = timezone),
                                        by = freq[1]),
                          "forecast" = rep(c(3, 5), 72 / 2))
```

```
data_1_day <- data.frame("index" = seq(as.Date("2020-1-1"), as.Date("2020-1-3"), by = freq[2]),
                         "forecast" = c(90, 100, 105))

forecasts_reconciled <- reconcile_forecasts(list(data_1_hour, data_1_day), freq,
                                            index = "index", outcome = "forecast",
                                            method = "temporal")
#------------------------------------------------------------------------------
# Grouped example 1: 2 forecasts, completely nested/hierarchical.
freq <- c("1 month")

dates <- seq(as.Date("2020-1-1"), as.Date("2020-3-1"), by = freq)

data_total <- data.frame("index" = dates,
                         "forecast" = c(50, 100, 75))

data_state <- data.frame("index" = rep(dates, 2),
                        "state" = c(rep("IL", length(dates)), rep("WI", length(dates))),
                         "forecast" = c(20, 60, 40, 25, 40, 50))

forecasts <- list("total" = data_total, "state" = data_state)

forecasts_reconciled <- reconcile_forecasts(forecasts, freq,
                                            index = "index", outcome = "forecast",
                                            method = "group")
#------------------------------------------------------------------------------
# Grouped example 2: 4 forecasts, non-nested.
freq <- c("1 month")

dates <- seq(as.Date("2020-1-1"), as.Date("2020-3-1"), by = freq)

data_total <- data.frame("index" = dates,
                         "forecast" = c(50, 100, 75))

data_state <- data.frame("index" = rep(dates, 2),
                        "state" = c(rep("IL", length(dates)), rep("WI", length(dates))),
                         "forecast" = c(20, 60, 40, 25, 40, 50))

data_sex <- data.frame("index" = rep(dates, 2),
                       "sex" = c(rep("M", length(dates)), rep("F", length(dates))),
                       "forecast" = c(25, 45, 40, 35, 40, 20))

data_state_sex <- data.frame("index" = rep(dates, 4),
                     "state" = c(rep("IL", length(dates)*2), rep("WI", length(dates)*2)),
                           "sex" = c(rep("M", 3), rep("F", 3), rep("M", 3), rep("F", 3)),
                           "forecast" = c(5, 15, 10, 30, 10, 10, 25, 30, 20, 10, 10, 15))

forecasts <- list("total" = data_total, "state" = data_state,
                  "sex" = data_sex, "state_sex" = data_state_sex)

forecasts_reconciled <- reconcile_forecasts(forecasts, freq,
                                            index = "index", outcome = "forecast",
                                            method = "group")
```

residuals *Return model residuals*

Description

Return model residuals

Usage

```
residuals(object, ...)
```

Arguments

`object`	An object of class 'training_results' from running `predict()` on a training dataset.
`...`	Not used.

Value

A data.frame of model residuals of class 'training_residuals'.

return_error *Compute forecast error*

Description

Compute forecast error metrics on the validation datasets or a new test dataset.

Usage

```
return_error(
  data_results,
  data_test = NULL,
  test_indices = NULL,
  aggregate = stats::median,
  metrics = c("mae", "mape", "mdape", "smape", "rmse", "rmsse"),
  models = NULL,
  horizons = NULL,
  windows = NULL,
  group_filter = NULL
)
```

Arguments

`data_results` An object of class 'training_results' or 'forecast_results' from running (a) `predict` on a trained model or (b) `combine_forecasts()`.

`data_test` Required for forecast results only. If `data_results` is an object of class 'forecast_results', a data.frame used to assess the accuracy of a 'forecast_results' object. `data_test` should have the outcome/target columns and any grouping columns.

`test_indices` Required if `data_test` is given or 'rmsse' row indices or dates (class 'Date' or 'POSIXt') with length `nrow(data_test)`.

`aggregate` Default `median`. A function–without parentheses–that aggregates historical prediction or forecast error across time series. All error metrics are first calculated at the level of the individual time series. `aggregate` is then used to combine error metrics across validation windows and horizons. Aggregations are returned at the group level if `data_results` contains groups.

`metrics` A character vector of common forecast error metrics. The default behavior is to return all metrics.

`models` Optional. A character vector of user-defined model names supplied to `train_model()` to filter results.

`horizons` Optional. A numeric vector to filter results by horizon.

`windows` Optional. A numeric vector to filter results by validation window number.

`group_filter` Optional. A string for filtering plot results for grouped time series (e.g., `"group_col_1 == 'A'"`). `group_filter` is passed to `dplyr::filter()` internally.

Value

An S3 object of class 'validation_error', 'forecast_error', or 'forecastML_error': A list of data.frames of error metrics for the validation or forecast dataset depending on the class of `data_results`: 'training_results', 'forecast_results', or 'forecastML' from `combine_forecasts()`.

A list containing:

- Error metrics by model, horizon, and validation window
- Error metrics by model and horizon, collapsed across validation windows
- Global error metrics by model collapsed across horizons and validation windows

Error Metrics

- `mae`: Mean absolute error (works with factor outcomes)
- `mape`: Mean absolute percentage error
- `mdape`: Median absolute percentage error
- `smape`: Symmetrical mean absolute percentage error
- `rmse`: Root mean squared error
- `rmsse`: Root mean squared scaled error from the M5 competition

Methods and related functions

The output of `return_error()` has the following generic S3 methods

- `plot` from `return_error()`
- `plot` from `return_error()`

Examples

```
# Sampled Seatbelts data from the R package datasets.
data("data_seatbelts", package = "forecastML")

# Example - Training data for 2 horizon-specific models w/ common lags per predictor.
horizons <- c(1, 12)
lookback <- 1:15

data_train <- create_lagged_df(data_seatbelts, type = "train", outcome_col = 1,
                               lookback = lookback, horizon = horizons)

# One custom validation window at the end of the dataset.
windows <- create_windows(data_train, window_start = 181, window_stop = 192)

# User-define model - LASSO
# A user-defined wrapper function for model training that takes the following
# arguments: (1) a horizon-specific data.frame made with create_lagged_df(..., type = "train")
# (e.g., my_lagged_df$horizon_h) and, optionally, (2) any number of additional named arguments
# which are passed as '...' in train_model().
library(glmnet)
model_function <- function(data, my_outcome_col) {

  x <- data[, -(my_outcome_col), drop = FALSE]
  y <- data[, my_outcome_col, drop = FALSE]
  x <- as.matrix(x, ncol = ncol(x))
  y <- as.matrix(y, ncol = ncol(y))

  model <- glmnet::cv.glmnet(x, y, nfolds = 3)
  return(model)
}

# my_outcome_col = 1 is passed in ... but could have been defined in model_function().
model_results <- train_model(data_train, windows, model_name = "LASSO", model_function,
                             my_outcome_col = 1)

# User-defined prediction function - LASSO
# The predict() wrapper takes two positional arguments. First,
# the returned model from the user-defined modeling function (model_function() above).
# Second, a data.frame of predictors--identical to the datasets returned from
# create_lagged_df(..., type = "train"). The function can return a 1- or 3-column data.frame
# with either (a) point forecasts or (b) point forecasts plus lower and upper forecast
# bounds (column order and column names do not matter).
prediction_function <- function(model, data_features) {

  x <- as.matrix(data_features, ncol = ncol(data_features))
  data_pred <- data.frame("y_pred" = predict(model, x, s = "lambda.min"))
  return(data_pred)
}

# Predict on the validation datasets.
data_valid <- predict(model_results, prediction_function = list(prediction_function),
                      data = data_train)

# Forecast error metrics for validation datasets.
data_error <- return_error(data_valid)
```

`return_hyper` *Return model hyperparameters across validation datasets*

Description

The purpose of this function is to support investigation into the stability of hyperparameters in the nested cross-validation and across forecast horizons.

Usage

```
return_hyper(forecast_model, hyper_function)
```

Arguments

`forecast_model` An object of class ’forecast_model’ from `train_model`.

`hyper_function` A user-defined function for retrieving model hyperparameters. See the example below for details.

Value

An S3 object of class ’forecast_model_hyper’: A data.frame of model-specific hyperparameters.

Methods and related functions

The output of `return_hyper()` has the following generic S3 methods

- `plot`

Examples

```
# Sampled Seatbelts data from the R package datasets.
data("data_seatbelts", package = "forecastML")

# Example - Training data for 2 horizon-specific models w/ common lags per predictor.
horizons <- c(1, 12)
lookback <- 1:15
```

```
data_train <- create_lagged_df(data_seatbelts, type = "train", outcome_col = 1,
                               lookback = lookback, horizon = horizons)

# One custom validation window at the end of the dataset.
windows <- create_windows(data_train, window_start = 181, window_stop = 192)

# User-define model - LASSO
# A user-defined wrapper function for model training that takes the following
# arguments: (1) a horizon-specific data.frame made with create_lagged_df(..., type = "train")
# (e.g., my_lagged_df$horizon_h) and, optionally, (2) any number of additional named arguments
# which are passed as '...' in train_model().
library(glmnet)
model_function <- function(data, my_outcome_col) {

  x <- data[, -(my_outcome_col), drop = FALSE]
  y <- data[, my_outcome_col, drop = FALSE]
  x <- as.matrix(x, ncol = ncol(x))
  y <- as.matrix(y, ncol = ncol(y))

  model <- glmnet::cv.glmnet(x, y, nfolds = 3)
  return(model)
}

# my_outcome_col = 1 is passed in ... but could have been defined in model_function().
model_results <- train_model(data_train, windows, model_name = "LASSO", model_function,
                             my_outcome_col = 1)

# User-defined prediction function - LASSO
# The predict() wrapper takes two positional arguments. First,
# the returned model from the user-defined modeling function (model_function() above).
# Second, a data.frame of predictors--identical to the datasets returned from
# create_lagged_df(..., type = "train"). The function can return a 1- or 3-column data.frame
# with either (a) point forecasts or (b) point forecasts plus lower and upper forecast
# bounds (column order and column names do not matter).
prediction_function <- function(model, data_features) {

  x <- as.matrix(data_features, ncol = ncol(data_features))

  data_pred <- data.frame("y_pred" = predict(model, x, s = "lambda.min"))
  return(data_pred)
}

# Predict on the validation datasets.
data_valid <- predict(model_results, prediction_function = list(prediction_function),
                      data = data_train)

# User-defined hyperparameter function - LASSO
# The hyperparameter function should take one positional argument--the returned model
# from the user-defined modeling function (model_function() above). It should
# return a 1-row data.frame of the optimal hyperparameters.
hyper_function <- function(model) {

  lambda_min <- model$lambda.min
  lambda_1se <- model$lambda.1se

  data_hyper <- data.frame("lambda_min" = lambda_min, "lambda_1se" = lambda_1se)
  return(data_hyper)
}

data_error <- return_error(data_valid)

data_hyper <- return_hyper(model_results, hyper_function)

plot(data_hyper, data_valid, data_error, type = "stability", horizons = c(1, 12))
```

`summary.lagged_df` *Return a summary of a lagged_df object*

Description

Return a summary of a lagged_df object

Usage

```
## S3 method for class 'lagged_df'
summary(object, ...)
```

Arguments

`object`	An object of class 'lagged_df' from `create_lagged_df()`.
`...`	Not used.

Value

A printed summary of the contents of the lagged_df object.

`train_model` *Train a model across horizons and validation datasets*

Description

Train a user-defined forecast model for each horizon, 'h', and across the validation datasets, 'd'. If `method = "direct"`, a total of 'h' * 'd' models are trained. If `method = "multi_output"`, a total of 1 * 'd' models are trained. These models can be trained in parallel with the `future` package.

Usage

```
train_model(
  lagged_df,
  windows,
  model_name,
  model_function,
  ...,
  use_future = FALSE,
  python = FALSE
)
```

Arguments

`lagged_df` An object of class 'lagged_df' from `create_lagged_df`.

`windows` An object of class 'windows' from `create_windows`.

`model_name` A name for the model.

`model_function` A user-defined wrapper function for model training that takes the following arguments: (1) a horizon-specific data.frame made with `create_lagged_df(..., type = "train")` (i.e., the dataset(s) stored in `lagged_df`) and, optionally, (2) any number of additional named arguments which can be passed in `...` in this function.

`...` Optional. Named arguments passed into the user-defined `model_function`.

`use_future` Boolean. If `TRUE`, the `future` package is used for training models in parallel. The models will train in parallel across either (1) model forecast horizons or (b) validation windows, whichever is longer (i.e., `length(create_lagged_df())` or `nrow(create_windows())`). The user should run `future::plan(future::multiprocess)` or similar prior to this function to train these models in parallel.

`python` Boolean. If `TRUE`, the `reticulate` package is used for model training.

Value

An S3 object of class 'forecast_model': A nested list of trained models. Models can be accessed with `my_trained_model$horizon_h$window_w$model` where 'h' gives the forecast horizon and 'w' gives the validation dataset window number from `create_windows()`.

Methods and related functions

The output of `train_model` can be passed into

- `return_error`
- `return_hyper`

and has the following generic S3 methods

- `predict`
- `plot` (from `predict.forecast_model(data = create_lagged_df(..., type = "train"))`)
- `plot` (from `predict.forecast_model(data = create_lagged_df(..., type = "forecast"))`)

Examples

```
# Sampled Seatbelts data from the R package datasets.
data("data_seatbelts", package = "forecastML")

# Example - Training data for 2 horizon-specific models w/ common lags per predictor.
horizons <- c(1, 12)
lookback <- 1:15

data_train <- create_lagged_df(data_seatbelts, type = "train", outcome_col = 1,
                               lookback = lookback, horizon = horizons)

# One custom validation window at the end of the dataset.
windows <- create_windows(data_train, window_start = 181, window_stop = 192)

# User-define model - LASSO
# A user-defined wrapper function for model training that takes the following
# arguments: (1) a horizon-specific data.frame made with create_lagged_df(..., type = "train")
# (e.g., my_lagged_df$horizon_h) and, optionally, (2) any number of additional named arguments
# which are passed as '...' in train_model().
library(glmnet)
model_function <- function(data, my_outcome_col) {

  x <- data[, -(my_outcome_col), drop = FALSE]
  y <- data[, my_outcome_col, drop = FALSE]
  x <- as.matrix(x, ncol = ncol(x))
  y <- as.matrix(y, ncol = ncol(y))

  model <- glmnet::cv.glmnet(x, y, nfolds = 3)
  return(model)
}

# my_outcome_col = 1 is passed in ... but could have been defined in model_function().
model_results <- train_model(data_train, windows, model_name = "LASSO", model_function,
                             my_outcome_col = 1)

# View the results for the model (a) trained on the first horizon
# and (b) to be assessed on the first outer-loop validation window.
model_results$horizon_1$window_1$model
```

Index