Package 'hdflex'

October 13, 2025

Type Package

Title High-Dimensional Aggregate Density Forecasts

Version 0.3.2

Maintainer Sven Lehmann <slehman5@uni-muenster.de>

Description Provides a forecasting method that efficiently maps vast numbers of (scalar-valued) signals into an aggregate density forecast in a time-varying and computationally fast manner. The method proceeds in two steps: First, it transforms a predictive signal into a density forecast and, second, it combines the resulting candidate density forecasts into an ultimate aggregate density forecast. For a detailed explanation of the method, please refer to Adaemmer et al. (2025) <doi:10.1080/07350015.2025.2526424>.

License GPL (>= 2)

URL https://github.com/lehmasve/hdflex

BugReports https://github.com/lehmasve/hdflex/issues

Depends R (>= 4.2.0)

Imports checkmate (>= 2.3.1), ggplot2 (>= 3.5.1), parallel, Rcpp, reshape2 (>= 1.4.4), stats, utils

Suggests testthat (>= 3.2.1), cowplot (>= 1.1.3)

LinkingTo Rcpp, RcppArmadillo, RcppThread

Config/testthat/edition 3

Encoding UTF-8

LazyData true

NeedsCompilation yes

RoxygenNote 7.3.3

Language en-US

Author Sven Lehmann [aut, cre, cph], Philipp Adämmer [aut], Rainer Schüssler [aut]

Repository CRAN

Date/Publication 2025-10-13 15:00:02 UTC

Contents

`dsc` *Generate dynamic subset forecast combinations*

Description

The `dsc()` function generates dynamic forecast combinations from a set of candidate density forecasts. For each period, it selects and combines a subset of predictive densities with the highest ranks regarding local predictive accuracy. The identities of the candidate forecasting models and the subset sizes used for building the aggregate predictive density may vary over time based on the data. If only one candidate forecast is picked, the approach temporarily collapses to pure model selection.

Usage

```
dsc(
  y,
  point_forecasts,
  variance_forecasts,
  gamma_grid,
  psi_grid,
  delta,
  burn_in,
  burn_in_dsc,
  metric,
  equal_weight,
  incl,
  portfolio_params = NULL
)
```

Arguments

`y`	A matrix of dimension `T * 1` or numeric vector of length `T` containing the observations of the target variable.
`point_forecasts`	A matrix with `T` rows containing the first moments of (conditionally) normal distributed predictive densities in each column.

`variance_forecasts`
: A matrix with `T` rows containing the second moments of (conditionally) normal distributed predictive densities in each column.

`gamma_grid`
: A numeric vector containing potential discount factors between 0 and 1 to exponentially down-weight the past predictive performance of the candidate forecasting models. The values of this tuning parameter are chosen in a procedure that amounts to leave-one-out cross-validation, taking into account the time series structure of the data. For details, *see Adaemmer et al. (2023)*.

`psi_grid`
: An integer vector that controls the (possible) sizes of the subsets. The values of this tuning parameter are chosen in a procedure that amounts to leave-one-out cross-validation, taking taking into account the time series structure of the data. For details, *see Adaemmer et al. (2023)*.

`delta`
: A numeric value between 0 and 1 denoting the discount factor applied to down-weight the past predictive performance of the aggregate predictive densities.

`burn_in`
: An integer value `>= 1` that denotes the number of observations used to 'initialize' the rankings. After 'burn_in' observations, the rankings for both, the candidate forecasting models and aggregate predictive densities are reset. `burn_in = 1` means no burn-in period is applied.

`burn_in_dsc`
: An integer value `>= 1` that denotes the number of observations used to 'initialize' the rankings. After 'burn_in_dsc' observations, only the ranking of the aggregate predictive densities is reset. `burn_in_dsc = 1` means no burn-in period is applied.

`metric`
: An integer from the set `1, 2, 3, 4, 5` representing the metric used to rank the candidate forecasting models (TV-C models) and subset combinations based on their predictive performance. The default value is `metric = 5` which ranks them according to the sum of (discounted) Continuous-Ranked-Probability-Scores (CRPS). `metric = 1` uses discounted Predictive Log-Likelihoods, `metric = 2` uses discounted Squared-Errors, `metric = 3` uses discounted Absolute-Errors, `metric = 4` uses discounted Compounded-Returns (in this case the target variable y has to be a time series of financial returns).

`equal_weight`
: A boolean that denotes whether equal weights are used to combine the candidate forecasts within a subset. If `FALSE`, the weights are calculated applying the softmax function on the ranking scores of the candidate forecasting models. The method proposed in Adaemmer et al. (2023) uses equal weights to combine the candidate forecasting models.

`incl`
: An optional integer vector that denotes signals that must be included in the subset combinations. For example, `incl = c(1, 3)` includes all candidate forecasting models generated by the first and third signals. If `NULL`, no signal is forced to be included.

`portfolio_params`
: A numeric vector of length 3 containing the following elements:

  risk_aversion A non-negative double representing the investor's risk aversion. Higher values indicate more risk-averse behavior.

  min_weight A double specifying the minimum weight allocated to the market. A non-negative lower bound effectively rules out short sales.

max_weight A double specifying the maximum weight allocated to the market. For example, a value of 2 allows for a maximum leverage ratio of two.

This parameter is only required if `metric = 4`.

Value

A list containing:

Forecasts A list containing:

Realization A vector with the actual values of the target variable.

Point_Forecasts A vector with the first moments of the aggregate predictive densities of the DSC model.

Variance_Prediction A vector with the second moments of the aggregate predictive densities of the DSC model.

Tuning_Parameters A list containing:

Gamma A vector containing the selected values for the tuning parameter gamma.

Psi A vector containing the selected values for the tuning parameter psi.

CFM A matrix containing the selected candidate forecasting models.

Model A list containing:

Gamma_grid The grid of gamma values used in the model.

Psi_grid The grid of psi values used in the model.

Delta The delta value used in the model.

Burn_in The burn-in period used in the model.

Burn_in_dsc The burn-in period used in the model.

Metric The ranking metric used in the model.

Equal_weight A boolean indicating if equal weighting was used.

Incl Additional included parameters.

Author(s)

Philipp Adämmer, Sven Lehmann, Rainer Schüssler

References

Beckmann, J., Koop, G., Korobilis, D., and Schüssler, R. A. (2020) "Exchange rate predictability and dynamic bayesian learning." *Journal of Applied Econometrics*, 35 (4): 410–421.

Dangl, T. and Halling, M. (2012) "Predictive regressions with time-varying coefficients." *Journal of Financial Economics*, 106 (1): 157–181.

Del Negro, M., Hasegawa, R. B., and Schorfheide, F. (2016) "Dynamic prediction pools: An investigation of financial frictions and forecasting performance." *Journal of Econometrics*, 192 (2): 391–405.

Koop, G. and Korobilis, D. (2012) "Forecasting inflation using dynamic model averaging." *International Economic Review*, 53 (3): 867–886.

Koop, G. and Korobilis, D. (2023) "Bayesian dynamic variable selection in high dimensions." *International Economic Review*.

Raftery, A. E., Kárn'y, M., and Ettler, P. (2010) "Online prediction under model uncertainty via dynamic model averaging: Application to a cold rolling mill." *Technometrics*, 52 (1): 52–66.

West, M. and Harrison, J. (1997) "Bayesian forecasting and dynamic models" *Springer*, 2nd edn.

See Also

https://github.com/lehmasve/hdflex#readme

Examples

```
# See example for tvc().
```

`hdflex` *hdflex: High-Dimensional Density Forecasts*

Description

hdflex contains the forecasting algorithm STSC developed by Adämmer, Lehmann and Schüssler (2023) doi:10.2139/ssrn.4342487. STSC is a novel time series forecasting method designed to handle very large sets of predictive signals, many of which are irrelevant or have only short-lived predictive power. Please cite the paper when using the package.

Author(s)

Philipp Adämmer, Sven Lehmann, Rainer Schüssler

`inflation_data` *Quarterly U.S. Inflation Dataset (Total CPI)*

Description

A high-dimensional dataset created by *Koop and Korobilis (2023)* that integrates predictive signals from various macroeconomic and financial sources.

Usage

```
inflation_data
```

Format

A matrix with 245 quarterly observations (rows) and 462 signals (columns):

Column 1 Transformed target variable: Total CPI (CPIAUCSL)

Columns 2-3 First and second lag of the target variable

Columns 4-442 Lagged and transformed signals from the sources listed above

Columns 443-462 External point forecasts available from 1976-Q1 to 2021-Q4 for quarterly Total CPI (CPIAUCSL), including:

First 12 forecasts Generated using regression trees, ridge regressions, and elastic nets over expanding and rolling windows

Remaining 8 forecasts Based on models discussed in Koop and Korobilis (2023) such as Gaussian process regressions (GPR_FAC5), Unobserved Component Stochastic Volatility (UCSV), and Variational Bayes Dynamic Variable Selection (VBDVS_X)

Details

The dataset includes data from the following sources:

- **FRED-QD dataset** (McCracken and Ng, 2020)
- **Portfolio data** (Jurado et al., 2015)
- **Stock market predictors** (Welch and Goyal, 2008)
- **University of Michigan consumer surveys**
- **World Bank's Pink Sheet commodity prices**
- **Key macroeconomic indicators** from the Federal Reserve Economic Data for Canada, Germany, Japan, and the United Kingdom

The dataset is pre-processed for one-step-ahead forecasts and includes external point forecasts. It spans from 1960-Q3 to 2021-Q4.

Source

doi:10.1111/iere.12623

References

Jurado, K., Ludvigson, S. C., and Ng, S. (2015) "Measuring uncertainty." *American Economic Review*, 105 (3): 1177–1216.

Koop, G. and Korobilis, D. (2023) "Bayesian dynamic variable selection in high dimensions." *International Economic Review*.

McCracken, M., and S. Ng (2020) "FRED-QD: A Quarterly Database for Macroeconomic Research" *National Bureau of Economic Research*, Working Paper 26872.

Welch, I. and Goyal, A. (2008) "A comprehensive look at the empirical performance of equity premium prediction." *The Review of Financial Studies*, 21 (4): 1455–1508.

stsc *Signal-Transformed-Subset-Combination (STSC)*

Description

`stsc()` is a time series forecasting method designed to handle vast sets of predictive signals, many of which may be irrelevant or short-lived. This method transforms heterogeneous scalar-valued signals into candidate density forecasts via time-varying coefficient models (TV-C), and subsequently, combines them into an ultimate aggregate density forecast via dynamic subset combinations (DSC).

Usage

```
stsc(
  y,
  X,
  Ext_F,
  init,
  lambda_grid,
  kappa_grid,
  bias,
  gamma_grid,
  psi_grid,
  delta,
  burn_in,
  burn_in_dsc,
  metric,
  equal_weight,
  incl,
  parallel = FALSE,
  n_threads = parallel::detectCores() - 2,
  portfolio_params = NULL
)
```

Arguments

`y`	A matrix of dimension `T * 1` or numeric vector of length `T` containing the observations of the target variable.
`X`	A matrix with `T` rows containing the lagged 'P-signals' in each column. Use `NULL` if no (external) 'P-signal' is to be included.
`Ext_F`	A matrix with `T` rows containing the (external) 'F-signals' in each column. For 'F-Signals', the slope of the TV-C models is fixed to 1. Use `NULL` if no (external) 'F-signal' is to be included.
`init`	An integer that denotes the number of observations used to initialize the observational variance and the coefficients' variance in the TV-C models.

`lambda_grid` A numeric vector which takes values between 0 and 1 denoting the discount factor(s) that control the dynamics of the time-varying coefficients. Each signal in combination with each value of lambda provides a separate candidate forecast. Constant coefficients are nested for the case `lambda = 1`.

`kappa_grid` A numeric vector which takes values between 0 and 1 to accommodate time-varying volatility in the TV-C models. The observational variance is estimated via Exponentially Weighted Moving Average and kappa denotes the underlying decay factor. Constant variance is nested for the case `kappa = 1`. Each signal in combination with each value of kappa provides a separate candidate density forecast. For the values of kappa, we follow the recommendation of RiskMetrics (Reuters, 1996).

`bias` A boolean to indicate whether the TV-C-models allow for a bias correction to F-signals. `TRUE` allows for a time-varying intercept, and `FALSE` sets (and fixes) the intercept to 0.

`gamma_grid` A numeric vector containing potential discount factors between 0 and 1 to exponentially down-weight the past predictive performance of the candidate forecasting models. The values of this tuning parameter are chosen in a procedure that amounts to leave-one-out cross-validation, taking into account the time series structure of the data. For details, *see Adaemmer et al. (2023)*.

`psi_grid` An integer vector that controls the (possible) sizes of the subsets. The values of this tuning parameter are chosen in a procedure that amounts to leave-one-out cross-validation, taking taking into account the time series structure of the data. For details, *see Adaemmer et al. (2023)*.

`delta` A numeric value between 0 and 1 denoting the discount factor applied to down-weight the past predictive performance of the aggregate predictive densities.

`burn_in` An integer value `>= 1` that denotes the number of observations used to 'initialize' the rankings. After 'burn_in' observations, the rankings for both, the candidate forecasting models and aggregate predictive densities are reset. `burn_in = 1` means no burn-in period is applied.

`burn_in_dsc` An integer value `>= 1` that denotes the number of observations used to 'initialize' the rankings. After 'burn_in_dsc' observations, only the ranking of the aggregate predictive densities is reset. `burn_in_dsc = 1` means no burn-in period is applied.

`metric` An integer from the set `1, 2, 3, 4, 5` representing the metric used to rank the candidate forecasting models (TV-C models) and subset combinations based on their predictive performance. The default value is `metric = 5` which ranks them according to the sum of (discounted) Continuous-Ranked-Probability-Scores (CRPS). `metric = 1` uses discounted Predictive Log-Likelihoods, `metric = 2` uses discounted Squared-Errors, `metric = 3` uses discounted Absolute-Errors, `metric = 4` uses discounted Compounded-Returns (in this case the target variable y has to be a time series of financial returns).

`equal_weight` A boolean that denotes whether equal weights are used to combine the candidate forecasts within a subset. If `FALSE`, the weights are calculated applying the softmax function on the ranking scores of the candidate forecasting models. The method proposed in Adaemmer et al. (2023) uses equal weights to combine the candidate forecasting models.

`incl` An optional integer vector that denotes signals that must be included in the subset combinations. For example, `incl = c(1, 3)` includes all candidate forecasting models generated by the first and third signals. If `NULL`, no signal is forced to be included.

`parallel` A boolean indicating whether the function should be parallelized.

`n_threads` An integer that denotes the number of cores used for parallelization. Only necessary if `parallel = TRUE`.

`portfolio_params`

A numeric vector of length 3 containing the following elements:

risk_aversion A non-negative double representing the investor's risk aversion. Higher values indicate more risk-averse behavior.

min_weight A double specifying the minimum weight allocated to the market. A non-negative lower bound effectively rules out short sales.

max_weight A double specifying the maximum weight allocated to the market. For example, a value of 2 allows for a maximum leverage ratio of two.

This parameter is only required if `metric = 4`.

Value

A list containing:

Forecasts A list containing:

- **Realization** A vector with the actual values of the target variable.
- **Point_Forecast** A vector with the first moments of the aggregate predictive densities of the STSC model.
- **Variance_Prediction** A vector with the second moments of the aggregate predictive densities of the STSC model.

Tuning_Parameters A list containing:

- **Gamma** A vector containing the selected values for the tuning parameter gamma.
- **Psi** A vector containing the selected values for the tuning parameter psi.
- **Signals** A matrix containing the selected signals.
- **Lambda** A matrix containing the selected values for the tuning parameter lambda.
- **Kappa** A matrix containing the selected values for the tuning parameter kappa.

Model A list containing:

- **Lambda_grid** The grid of lambda values used in the model.
- **Kappa_grid** The grid of kappa values used in the model.
- **Gamma_grid** The grid of gamma values used in the model.
- **Psi_grid** The grid of psi values used in the model.
- **Delta** The delta value used in the model.
- **Init** The init value used in the model.
- **Burn_in** The burn-in period used in the model.
- **Burn_in_dsc** The burn-in period used in the model.
- **metric** The ranking metric used in the model.
- **Equal_weight** A boolean indicating if equal weighting was used.
- **Bias** A boolean indicating if bias correction was applied.
- **Incl** Additional included parameters.

Author(s)

Philipp Adämmer, Sven Lehmann, Rainer Schüssler

References

Beckmann, J., Koop, G., Korobilis, D., and Schüssler, R. A. (2020) "Exchange rate predictability and dynamic bayesian learning." *Journal of Applied Econometrics*, 35 (4): 410–421.

Dangl, T. and Halling, M. (2012) "Predictive regressions with time-varying coefficients." *Journal of Financial Economics*, 106 (1): 157–181.

Del Negro, M., Hasegawa, R. B., and Schorfheide, F. (2016) "Dynamic prediction pools: An investigation of financial frictions and forecasting performance." *Journal of Econometrics*, 192 (2): 391–405.

Koop, G. and Korobilis, D. (2012) "Forecasting inflation using dynamic model averaging." *International Economic Review*, 53 (3): 867–886.

Koop, G. and Korobilis, D. (2023) "Bayesian dynamic variable selection in high dimensions." *International Economic Review*.

Raftery, A. E., Kárn'y, M., and Ettler, P. (2010) "Online prediction under model uncertainty via dynamic model averaging: Application to a cold rolling mill." *Technometrics*, 52 (1): 52–66.

West, M. and Harrison, J. (1997) "Bayesian forecasting and dynamic models" *Springer*, 2nd edn.

See Also

https://github.com/lehmasve/hdflex#readme

Examples

```
############################################################
######### Forecasting quarterly U.S. inflation ##########
#### Please see Koop & Korobilis (2023) for further  ####
#### details regarding the data & external forecasts ####
############################################################

# Load Package
library("hdflex")
library("ggplot2")
library("cowplot")

########## Get Data ##########
# Load Package Data
inflation_data <- inflation_data

# Set Target Variable
y <- inflation_data[,  1]

# Set 'P-Signals'
X <- inflation_data[, 2:442]

# Set 'F-Signals'
```

```
Ext_F <- inflation_data[, 443:462]

# Get Dates and Number of Observations
tdates <- rownames(inflation_data)
tlength <- length(tdates)

# First complete observation (no missing values)
first_complete <- which(complete.cases(inflation_data))[1]

########## Rolling AR2-Benchmark ##########
# Set up matrix for predictions
benchmark <- matrix(NA, nrow = tlength,
                    ncol = 1, dimnames = list(tdates, "AR2"))

# Set Window-Size (15 years of quarterly data)
window_size <- 15 * 4

# Time Sequence
t_seq <- seq(window_size, tlength - 1)

# Loop with rolling window
for (t in t_seq) {

  # Split Data for Training Train Data
  x_train <- cbind(int = 1, X[(t - window_size + 1):t, 1:2])
  y_train <- y[(t - window_size + 1):t]

  # Split Data for Prediction
  x_pred <- cbind(int = 1, X[t + 1, 1:2, drop = FALSE])

  # Fit AR-Model
  model_ar <- .lm.fit(x_train, y_train)

  # Predict and store in benchmark matrix
  benchmark[t + 1, ] <- x_pred %*% model_ar$coefficients
}

########## STSC ##########
# Set TV-C-Parameter
init <- 5 * 4
lambda_grid <- c(0.90, 0.95, 1.00)
kappa_grid <- c(0.94, 0.96, 0.98)
bias <- TRUE

# Set DSC-Parameter
gamma_grid <- c(0.40, 0.50, 0.60, 0.70, 0.80, 0.90,
                0.91, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98, 0.99, 1.00)
n_tvc <- (ncol(X) + ncol(Ext_F)) * length(lambda_grid) * length(kappa_grid)
psi_grid <- c(1:100, sapply(1:4, function(i) floor(i * n_tvc / 4)))
delta <- 0.95
burn_in <- first_complete + init / 2
burn_in_dsc <- 1
metric <- 5
```

```
equal_weight <- TRUE
incl <- NULL
parallel <- FALSE
n_threads <- NULL

# Apply STSC-Function
results <- hdflex::stsc(y,
                        X,
                        Ext_F,
                        init,
                        lambda_grid,
                        kappa_grid,
                        bias,
                        gamma_grid,
                        psi_grid,
                        delta,
                        burn_in,
                        burn_in_dsc,
                        metric,
                        equal_weight,
                        incl,
                        parallel,
                        n_threads,
                        NULL)

########## Evaluation ##########
# Define Evaluation Period (OOS-Period)
eval_period <- which(tdates >= "1991-04-01" & tdates <= "2021-12-01")

# Get Evaluation Summary for STSC
eval_results <- summary(obj = results, eval_period = eval_period)

# Calculate (Mean-)Squared-Errors for AR2-Benchmark
se_ar2 <- (y[eval_period] - benchmark[eval_period, 1])^2
mse_ar2 <- mean(se_ar2)

# Create CSSED-Plot
cssed <- cumsum(se_ar2 - eval_results$MSE[[2]])
plot_cssed <- ggplot(
  data = data.frame(eval_period, cssed),
  aes(x = eval_period, y = cssed)
  ) +
  geom_line() +
  ylim(-0.0008, 0.0008) +
  ggtitle("Cumulative Squared Error Differences") +
  xlab("Time Index") +
  ylab("CSSED") +
  geom_hline(yintercept = 0, linetype = "dashed", color = "darkgray") +
  theme_minimal(base_size = 15) +
  theme(
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    panel.border = element_rect(colour = "black", fill = NA),
```

```
    axis.ticks = element_line(colour = "black"),
    plot.title = element_text(hjust = 0.5)
  )

# Show Plots
options(repr.plot.width = 15, repr.plot.height = 15)
plots_list <- eval_results$Plots
plots_list <- c(list(plot_cssed), plots_list)
cowplot::plot_grid(plotlist = plots_list,
                   ncol = 2,
                   nrow = 3,
                   align = "hv")

# Relative MSE
print(paste("Relative MSE:", round(eval_results$MSE[[1]] / mse_ar2, 4)))
```

`summary.dsc_obj` *Summary for 'dsc' object*

Description

This function plots the evolution of the tuning parameters for a 'dsc' object and returns basic performance metrics.

Usage

```
## S3 method for class 'dsc_obj'
summary(object, eval_period = NULL, ...)
```

Arguments

`object`	An object of type 'dsc'.
`eval_period`	(Optional) A vector of indices to specify the evaluation period. Defaults to the entire period after burn-in.
`...`	Additional arguments to be consistent with the S3 print() function.

Value

A list containing:

MSE A list with the mean squared error (MSE) and squared errors (SE).

ACRPS A list with the average continuous ranked probability score (ACRPS) and CRPS values.

APLL A list with the average predictive log-likelihood (APLL) and predictive log-likelihood (PLL) values.

Plots A list of ggplot objects for visualizing the tuning parameters and selected CFMs.

References

Gneiting, T., Raftery, A. E., Westveld, A. H., and Goldman, T. (2005): Calibrated Probabilistic Forecasting Using Ensemble Model Output Statistics and Minimum CRPS Estimation. *Monthly Weather Review*, 133: 1098–1118.

Jordan, A., Krueger, F., and Lerch, S. (2019): "Evaluating Probabilistic Forecasts with scoringRules." *Journal of Statistical Software*, 90(12): 1-37.

Examples

```
# See example for tvc().
```

`summary.stsc_obj` *Summary for 'stsc' object*

Description

This function plots the evolution of the tuning parameters for an 'stsc' object and returns basic performance metrics.

Usage

```
## S3 method for class 'stsc_obj'
summary(object, eval_period = NULL, ...)
```

Arguments

`object`	An object of type 'stsc'.
`eval_period`	(Optional) A vector of indices to specify the evaluation period. Defaults to the entire period after burn-in.
`...`	Additional arguments to be consistent with the S3 print() function.

Value

A list containing:

MSE A list with the mean squared error (MSE) and squared errors (SE).

ACRPS A list with the average continuous ranked probability score (ACRPS) and CRPS values.

APLL A list with the average predictive log-likelihood (APLL) and predictive log-likelihood (PLL) values.

Plots A list of ggplot objects for visualizing the tuning parameters and selected signals.

References

Gneiting, T., Raftery, A. E., Westveld, A. H., and Goldman, T. (2005): Calibrated Probabilistic Forecasting Using Ensemble Model Output Statistics and Minimum CRPS Estimation. *Monthly Weather Review*, 133: 1098–1118.

Jordan, A., Krueger, F., and Lerch, S. (2019): "Evaluating Probabilistic Forecasts with scoringRules." *Journal of Statistical Software*, 90(12): 1-37.

Examples

```
# See example for stsc().
```

`tvc` *Compute density forecasts using univariate time-varying coefficient (TV-C) models*

Description

The `tvc()` function generates density forecasts based on univariate time-varying coefficient models in state-space form. Each forecasting model includes an intercept and one predictive signal, which can either be a 'P-signal' or 'F-signal'. All models are estimated independently and both estimation and forecasting are carried out recursively.

Usage

```
tvc(y, X, Ext_F, init, lambda_grid, kappa_grid, bias)
```

Arguments

`y`	A matrix of dimension `T * 1` or numeric vector of length `T` containing the observations of the target variable.
`X`	A matrix with `T` rows containing the lagged 'P-signals' in each column. Use `NULL` if no (external) 'P-signal' is to be included.
`Ext_F`	A matrix with `T` rows containing the (external) 'F-signals' in each column. For 'F-Signals', the slope of the TV-C models is fixed to 1. Use `NULL` if no (external) 'F-signal' is to be included.
`init`	An integer that denotes the number of observations used to initialize the observational variance and the coefficients' variance in the TV-C models.
`lambda_grid`	A numeric vector which takes values between 0 and 1 denoting the discount factor(s) that control the dynamics of the time-varying coefficients. Each signal in combination with each value of lambda provides a separate candidate forecast. Constant coefficients are nested for the case `lambda = 1`.

`kappa_grid` A numeric vector which takes values between 0 and 1 to accommodate time-varying volatility in the TV-C models. The observational variance is estimated via Exponentially Weighted Moving Average and kappa denotes the underlying decay factor. Constant variance is nested for the case `kappa = 1`. Each signal in combination with each value of kappa provides a separate candidate density forecast. For the values of kappa, we follow the recommendation of RiskMetrics (Reuters, 1996).

`bias` A boolean to indicate whether the TV-C-models allow for a bias correction to F-signals. `TRUE` allows for a time-varying intercept, and `FALSE` sets (and fixes) the intercept to 0.

Value

A list containing:

Forecasts A list containing:

Realization: A vector with the actual values of the target variable.

Point_Forecasts: A vector with the first moments of the predictive densities.

Variance_Forecasts: A vector with the second moments of the predictive densities.

Model A list containing:

Lambda_grid The grid of lambda values used in the model.

Kappa_grid The grid of kappa values used in the model.

Init The init value used in the model.

Bias A boolean indicating if bias correct was applied to F-signals.

Author(s)

Philipp Adämmer, Sven Lehmann, Rainer Schüssler

References

Beckmann, J., Koop, G., Korobilis, D., and Schüssler, R. A. (2020) "Exchange rate predictability and dynamic bayesian learning." *Journal of Applied Econometrics*, 35 (4): 410–421.

Dangl, T. and Halling, M. (2012) "Predictive regressions with time-varying coefficients." *Journal of Financial Economics*, 106 (1): 157–181.

Del Negro, M., Hasegawa, R. B., and Schorfheide, F. (2016) "Dynamic prediction pools: An investigation of financial frictions and forecasting performance." *Journal of Econometrics*, 192 (2): 391–405.

Koop, G. and Korobilis, D. (2012) "Forecasting inflation using dynamic model averaging." *International Economic Review*, 53 (3): 867–886.

Koop, G. and Korobilis, D. (2023) "Bayesian dynamic variable selection in high dimensions." *International Economic Review*.

Raftery, A. E., Kárn‘y, M., and Ettler, P. (2010) "Online prediction under model uncertainty via dynamic model averaging: Application to a cold rolling mill." *Technometrics*, 52 (1): 52–66.

West, M. and Harrison, J. (1997) "Bayesian forecasting and dynamic models" *Springer*, 2nd edn.

See Also

https://github.com/lehmasve/hdflex#readme

Examples

```
############################################################
######### Forecasting quarterly U.S. inflation ##########
#### Please see Koop & Korobilis (2023) for further  ####
#### details regarding the data & external forecasts ####
############################################################

# Load Package
library("hdflex")
library("ggplot2")
library("cowplot")

########## Get Data ##########
# Load Package Data
inflation_data <- inflation_data

# Set Target Variable
y <- inflation_data[,  1]

# Set 'P-Signals'
X <- inflation_data[, 2:442]

# Set 'F-Signals'
Ext_F <- inflation_data[, 443:462]

# Get Dates and Number of Observations
tdates <- rownames(inflation_data)
tlength <- length(tdates)

# First complete observation (no missing values)
first_complete <- which(complete.cases(inflation_data))[1]

########## Rolling AR2-Benchmark ##########
# Set up matrix for predictions
benchmark <- matrix(NA, nrow = tlength,
                    ncol = 1, dimnames = list(tdates, "AR2"))

# Set Window-Size (15 years of quarterly data)
window_size <- 15 * 4

# Time Sequence
t_seq <- seq(window_size, tlength - 1)

# Loop with rolling window
for (t in t_seq) {

  # Split Data for Training Train Data
```

```
  x_train <- cbind(int = 1, X[(t - window_size + 1):t, 1:2])
  y_train <- y[(t - window_size + 1):t]

  # Split Data for Prediction
  x_pred <- cbind(int = 1, X[t + 1, 1:2, drop = FALSE])

  # Fit AR-Model
  model_ar <- .lm.fit(x_train, y_train)

  # Predict and store in benchmark matrix
  benchmark[t + 1, ] <- x_pred %*% model_ar$coefficients
}

########## STSC ##########
### Part 1: TVC-Function
# Set TV-C-Parameter
init <- 5 * 4
lambda_grid <- c(0.90, 0.95, 1.00)
kappa_grid <- c(0.94, 0.96, 0.98)
bias <- TRUE

# Apply TVC-Function
tvc_results <- hdflex::tvc(y,
                           X,
                           Ext_F,
                           init,
                           lambda_grid,
                           kappa_grid,
                           bias)

# Assign TVC-Results
forecast_tvc <- tvc_results$Forecasts$Point_Forecasts
variance_tvc <- tvc_results$Forecasts$Variance_Forecasts

# First complete forecast period (no missing values)
sub_period <- seq(which(complete.cases(forecast_tvc))[1], tlength)

### Part 2: DSC-Function
# Set DSC-Parameter
gamma_grid <- c(0.40, 0.50, 0.60, 0.70, 0.80, 0.90,
                0.91, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98, 0.99, 1.00)
psi_grid <- c(1:100, sapply(1:4, function(i) floor(i * ncol(forecast_tvc) / 4)))
delta <- 0.95
burn_in_tvc <- (init / 2) + 1
burn_in_dsc <- 1
metric <- 5
equal_weight <- TRUE
incl <- NULL

# Apply DSC-Function
dsc_results <- hdflex::dsc(y[sub_period],
                           forecast_tvc[sub_period, , drop = FALSE],
                           variance_tvc[sub_period, , drop = FALSE],
```

```
                              gamma_grid,
                              psi_grid,
                              delta,
                              burn_in_tvc,
                              burn_in_dsc,
                              metric,
                              equal_weight,
                              incl,
                              NULL)

 # Assign DSC-Results
 pred_stsc <- dsc_results$Forecasts$Point_Forecasts
 var_stsc <- dsc_results$Forecasts$Variance_Forecasts

 ########## Evaluation ##########
 # Define Evaluation Period (OOS-Period)
eval_period <- which(tdates[sub_period] >= "1991-04-01" & tdates[sub_period] <= "2021-12-01")

 # Get Evaluation Summary for STSC
 eval_results <- summary(obj = dsc_results, eval_period = eval_period)

 # Calculate (Mean-)Squared-Errors for AR2-Benchmark
 oos_y <- y[sub_period][eval_period]
 oos_benchmark <- benchmark[sub_period[eval_period], , drop = FALSE]
 se_ar2 <- (oos_y - oos_benchmark)^2
 mse_ar2 <- mean(se_ar2)

 # Create Cumulative Squared Error Differences (CSSED) Plot
 cssed <- cumsum(se_ar2 - eval_results$MSE[[2]])
 plot_cssed <- ggplot(
   data.frame(eval_period, cssed),
   aes(x = eval_period, y = cssed)
 ) +
   geom_line() +
   ylim(-0.0008, 0.0008) +
   ggtitle("Cumulative Squared Error Differences") +
   xlab("Time Index") +
   ylab("CSSED") +
   geom_hline(yintercept = 0, linetype = "dashed", color = "darkgray") +
   theme_minimal(base_size = 15) +
   theme(
     panel.grid.major = element_blank(),
     panel.grid.minor = element_blank(),
     panel.border = element_rect(colour = "black", fill = NA),
     axis.ticks = element_line(colour = "black"),
     plot.title = element_text(hjust = 0.5)
   )

 # Show Plots
 options(repr.plot.width = 15, repr.plot.height = 15)
 plots_list <- eval_results$Plots
 plots_list <- c(list(plot_cssed), plots_list)
 cowplot::plot_grid(plotlist = plots_list,
```

```
                    ncol = 2,
                    nrow = 3,
                    align = "hv")

# Relative MSE
print(paste("Relative MSE:", round(eval_results$MSE[[1]] / mse_ar2, 4)))
```

Index