Package 'IFANN'

November 20, 2026

Type Package

Title Intuitionistic Fuzzy Artificial Neural Network for Time Series

Version 0.1.0

Description Time series forecasting with an intuitionistic fuzzy artificial neural network. Lagged values of the series and their intuitionistic fuzzy membership and non-membership degrees, obtained by intuitionistic fuzzy c-means clustering, are used as inputs of a feed-forward neural network with one hidden layer, averaged over several random starts. The lag order and the network hyper-parameters (number of hidden neurons and weight decay) can be set manually or tuned automatically by grid search over user-defined minimum, maximum and step ranges, with time-ordered validation and a progress display. The intuitionistic fuzzy approach follows Yolcu and Yolcu (2023) <doi:10.1016/j.eswa.2022.119336>.

License GPL-3

Encoding UTF-8

Imports nnet, stats, utils

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

Config/roxygen2/version 8.1.1

NeedsCompilation no

Author Anita Sarkar [aut],
Md Yeasin [aut, cre],
Ranjit Kumar Paul [aut],
Pushkar Bora [aut],
Ankit Kumar Singh [aut],
Amrit Kumar Paul [aut]

Maintainer Md Yeasin <yeasin.iasri@gmail.com>

Repository CRAN

Date/Publication 2026-11-20 12:20:08 UTC

Contents

IFANN 2

Index **6**

IFANN *Intuitionistic Fuzzy Artificial Neural Network for Time Series*

Description

Fits an intuitionistic fuzzy artificial neural network (IFANN) model to a univariate time series. The lagged values of the series and their intuitionistic fuzzy membership and non-membership degrees (from intuitionistic fuzzy c-means clustering) are used as inputs of a feed-forward neural network with one hidden layer. The lag order and the network hyper-parameters can each be set manually or tuned automatically by grid search.

Usage

```
IFANN(
  Data,
  Class = 3,
  Split = 0.8,
  Lag_Tuning = c("auto", "manual"),
  Lag = 1,
  Lag_Range = c(min = 1, max = 12, step = 1),
  ANN_Tuning = c("auto", "manual"),
  Size = 4,
  Decay = 0.01,
  Size_Range = c(min = 2, max = 8, step = 2),
  Decay_Range = c(min = 0.01, max = 0.09, step = 0.04),
  Maxit = 200,
  Repeats = 5,
  Validation = c("holdout", "tscv"),
  Val_Split = 0.2,
  Folds = 3,
  Metric = c("RMSE", "MAE", "MAPE"),
  Forecast_Type = c("one-step", "recursive"),
  m = 2,
  Alpha = 0.85,
  Max_Iter = 100,
  Tol = 1e-05,
  Verbose = TRUE
)
```

Arguments

`Data`	Numeric time series (vector or `ts`) without missing values.

`Class`	Number of intuitionistic fuzzy clusters (at least 2).
`Split`	Proportion of the data used for training, strictly between 0 and 1.
`Lag_Tuning`	`"auto"` (default) to tune the lag order over `Lag_Range`, or `"manual"` to use `Lag`.
`Lag`	Lag order used when `Lag_Tuning = "manual"` (a single positive integer).
`Lag_Range`	Lag search range `c(min, max, step)` for `Lag_Tuning = "auto"` (positive integers). Lags too large for the data are dropped with a warning.
`ANN_Tuning`	`"auto"` (default) to tune `Size` and `Decay`, or `"manual"` to use the single values given.
`Size`, `Decay`	Number of hidden neurons (a positive integer) and weight decay (non-negative), used when `ANN_Tuning = "manual"`.
`Size_Range`, `Decay_Range`	Search ranges `c(min, max, step)` for `ANN_Tuning = "auto"`. `Size` values must be positive integers and `Decay` values non-negative.
`Maxit`	Maximum number of training iterations of each network.
`Repeats`	Number of networks fitted from different random starting weights; their predictions are averaged.
`Validation`	`"holdout"` (default) or `"tscv"` (expanding-window time-series cross-validation).
`Val_Split`	Share of the training set used for validation, strictly between 0 and 1.
`Folds`	Number of folds for `Validation = "tscv"`.
`Metric`	Criterion minimised during tuning: `"RMSE"` (default), `"MAE"` or `"MAPE"`.
`Forecast_Type`	`"one-step"` (default): every test value is predicted from the observed previous values. `"recursive"`: the test period is forecast multi-step ahead, feeding predictions back as lags. The same type is used during validation.
`m`	Fuzzifier of the clustering (greater than 1).
`Alpha`	Exponent of the Yager non-membership generator, in (0, 1].
`Max_Iter`, `Tol`	Maximum number of iterations and convergence tolerance of the clustering.
`Verbose`	Logical; show the grid-search progress and the result summary as messages (default `TRUE`).

Details

Model. The first `round(length(Data) * Split)` observations form the training set and the rest the test set. Intuitionistic fuzzy c-means clustering (Chaira, 2011) with `Class` clusters is fitted to the training observations only. Non-membership degrees come from the Yager-type generator $\nu = (1 - \mu^{\alpha})^{1/\alpha}$, so the hesitation degree $\pi = 1 - \mu - \nu$ is non-negative for $0 < \alpha \leq 1$; `Alpha = 1` gives ordinary fuzzy sets. For a target y_t and lag order L the inputs are $y_{t-1}, \ldots, y_{t-L}$ (min-max scaled with training statistics) and the membership and non-membership degrees of each lagged value in every cluster, i.e. $p = L(1 + 2\,Class)$ inputs. Only past values enter the inputs, so the test-set forecasts use no information from the values being forecast.

Network. The network has p inputs, one hidden layer of `Size` logistic neurons and a linear output, and is fitted to the target scaled to [0, 1] with `nnet` (BFGS with weight decay `Decay`, at most `Maxit` iterations). Because the result depends on the random starting weights, `Repeats` networks are fitted from different starts and their predictions are averaged. `Maxit` and `Repeats` are fixed settings used for every model, including during tuning.

Tuning. `Lag_Tuning` and `ANN_Tuning` are set independently:

- `"manual"`: the single values `Lag` and / or `Size`, `Decay` are used as given.
- `"auto"`: the parameters are tuned by grid search. Each search range is given as `c(min, max, step)` and expands to `seq(min, max, by = step)`; for example `Lag_Range = c(1, 12, 1)` gives the lags 1 to 12 and `Size_Range = c(2, 8, 2)` gives 2, 4, 6, 8 hidden neurons. If `max` is not reached exactly by the steps, the last value is the largest one below `max`. Use `min = max` for a single value.

When both are `"auto"` every combination of lag and network parameters is evaluated. With `Verbose = TRUE` (default) the search prints the values of every parameter, the number of combinations, and a progress bar with the best score so far, the elapsed time and the estimated time remaining.

Validation. Candidates are compared on the last `Val_Split` share of the training set. With `"holdout"` the model is fitted on the observations before that block and evaluated on the block. With `"tscv"` the block is divided into `Folds` consecutive parts and each part is predicted from a model fitted on all earlier observations (expanding window). The clustering and scaling are re-estimated on each fitting window. After tuning, the model is refitted on the whole training set with the best parameters.

Output. With `Verbose = TRUE` a summary of the selected parameters and the accuracy is shown as a message at the end (silence it with `Verbose = FALSE` or `suppressMessages()`), and the result is returned invisibly; assign it (e.g. `Model <- IFANN(...)`) to keep it.

Reproducibility. The clustering starts from a random membership matrix and the network weights are initialised at random, so call `set.seed()` before `IFANN()` to obtain reproducible results.

Value

A list (returned invisibly) with components

`Accuracy` Matrix of RMSE, MAE and MAPE (%) for the training and test sets.

`Train_Fitted` Fitted values for the training targets (the first `Lag` training observations have no fitted value).

`Test_Prediction` Predictions for the test set.

`Train_Actual`, `Test_Actual` The corresponding observed values.

`Best_Params` List with `Lag`, `Size` and `Decay` of the final model.

`Architecture` Number of inputs, hidden neurons and outputs, number of weights per network, `Maxit`, `Repeats` and `Converged`, the number of networks whose optimiser stopped before reaching `Maxit`.

`Tuning` `NULL` when nothing was tuned; otherwise a list with the validation scheme, metric, number of combinations, best validation score, elapsed time and `History` (a data frame with the score of every combination).

`Networks` List of the `Repeats` fitted `nnet` objects of the final model (fitted to the target scaled to [0, 1]).

`Scaling` Minimum and maximum of the training data used for the [0, 1] scaling.

`IFCM` The intuitionistic fuzzy clustering of the training set (centres, membership, non-membership and hesitation degrees).

`Settings` Other settings used.

References

Yolcu, O.C. and Yolcu, U. (2023). A novel intuitionistic fuzzy time series prediction model with cascaded structure for financial time series. *Expert Systems with Applications*, 215, 119336. doi:10.1016/j.eswa.2022.119336

Chaira, T. (2011). A novel intuitionistic fuzzy C means clustering algorithm and its application to medical images. *Applied Soft Computing*, 11(2), 1711–1717.

Venables, W.N. and Ripley, B.D. (2002). *Modern Applied Statistics with S*. Fourth edition. Springer, New York.

Examples

```
set.seed(123)
dat <- 50 + 10 * sin(2 * pi * (1:120) / 12) + rnorm(120)

# Everything manual: lag 12, 4 hidden neurons, weight decay 0.05
Model1 <- IFANN(dat, Class = 3, Split = 0.8,
                Lag_Tuning = "manual", Lag = 12,
                ANN_Tuning = "manual", Size = 4, Decay = 0.05,
                Repeats = 3)
Model1$Accuracy

# Manual lag, network parameters tuned on a very small grid
Model2 <- IFANN(dat, Class = 3, Split = 0.8,
                Lag_Tuning = "manual", Lag = 12,
                Size_Range = c(2, 4, 2),           # 2, 4
                Decay_Range = c(0.01, 0.09, 0.08), # 0.01, 0.09
                Repeats = 2, Verbose = FALSE)
Model2$Best_Params


# Lag and network parameters tuned jointly (small grid)
Model3 <- IFANN(dat, Class = 3, Split = 0.8,
                Lag_Range = c(2, 12, 5),           # 2, 7, 12
                Size_Range = c(2, 4, 2),           # 2, 4
                Decay_Range = c(0.05, 0.05, 0),    # 0.05
                Repeats = 2, Verbose = FALSE)
head(Model3$Tuning$History)
```

Index

IFANN, 2

nnet, *3*, *4*