Overview

Integrated tuning automates hyperparameter search for time-series learners in a single pipeline. It handles preprocessing, window size, and model hyperparameters, then evaluates and returns the best configuration for your training data.

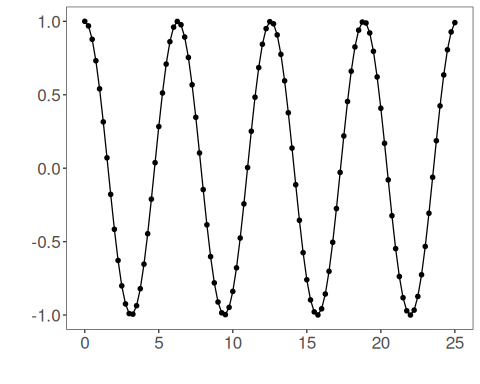
What you will learn - Create sliding windows suitable for supervised learning - Split the data into train/test respecting time order - Define a search space and run integrated tuning - Inspect evaluation metrics and visualize predictions

# Install tspredit if needed  
#install.packages("tspredit")

# Load packages  
library(daltoolbox)  
library(tspredit)

# Create a simple cosine series for demonstration  
  
i <- seq(0, 25, 0.25)  
x <- cos(i)

# Visualize the time series  
plot\_ts(x=i, y=x) + theme(text = element\_text(size=16))



# Sliding windows  
  
# Create a sliding-window matrix for supervised learning.  
# Each row contains 10 attributes (t9..t0) representing the last 10 observations.  
sw\_size <- 10  
ts <- ts\_data(x, sw\_size)  
ts\_head(ts, 3)

## t9 t8 t7 t6 t5 t4 t3 t2 t1 t0  
## [1,] 1.0000000 0.9689124 0.8775826 0.7316889 0.5403023 0.3153224 0.0707372 -0.1782461 -0.4161468 -0.6281736  
## [2,] 0.9689124 0.8775826 0.7316889 0.5403023 0.3153224 0.0707372 -0.1782461 -0.4161468 -0.6281736 -0.8011436  
## [3,] 0.8775826 0.7316889 0.5403023 0.3153224 0.0707372 -0.1782461 -0.4161468 -0.6281736 -0.8011436 -0.9243024

# Data sampling (train/test split)  
  
test\_size <- 1 # keep last step for testing  
samp <- ts\_sample(ts, test\_size)  
ts\_head(samp$train, 3)

## t9 t8 t7 t6 t5 t4 t3 t2 t1 t0  
## [1,] 1.0000000 0.9689124 0.8775826 0.7316889 0.5403023 0.3153224 0.0707372 -0.1782461 -0.4161468 -0.6281736  
## [2,] 0.9689124 0.8775826 0.7316889 0.5403023 0.3153224 0.0707372 -0.1782461 -0.4161468 -0.6281736 -0.8011436  
## [3,] 0.8775826 0.7316889 0.5403023 0.3153224 0.0707372 -0.1782461 -0.4161468 -0.6281736 -0.8011436 -0.9243024

ts\_head(samp$test)

## t9 t8 t7 t6 t5 t4 t3 t2 t1 t0  
## [1,] -0.7256268 -0.532833 -0.3069103 -0.06190529 0.1869486 0.424179 0.635036 0.8064095 0.9276444 0.9912028

# Define integrated tuning  
  
# We will:  
# - search over input window sizes (3..5)  
# - use ELM as the base model  
# - apply global min-max normalization as preprocessing  
# - explore ranges for hidden units and activation function  
  
tune <- ts\_integtune(input\_size=c(3:5), base\_model = ts\_elm(), preprocess = list(ts\_norm\_gminmax()),  
 ranges = list(nhid = 1:5, actfun=c('sig', 'radbas', 'tribas', 'relu', 'purelin')))

# Fit the tuned pipeline on training data  
  
io\_train <- ts\_projection(samp$train)  
model <- fit(tune, x=io\_train$input, y=io\_train$output)

# Evaluate training adjustment (in-sample)  
  
adjust <- predict(model, io\_train$input)  
ev\_adjust <- evaluate(model, io\_train$output, adjust)  
print(head(ev\_adjust$metrics))

## mse smape R2  
## 1 3.235577e-30 7.125308e-15 1

# Forecast on the test segment  
  
steps\_ahead <- 1  
io\_test <- ts\_projection(samp$test)  
prediction <- predict(model, x=io\_test$input, steps\_ahead=steps\_ahead)  
prediction <- as.vector(prediction)  
  
output <- as.vector(io\_test$output)  
if (steps\_ahead > 1)  
 output <- output[1:steps\_ahead]  
  
print(sprintf("%.2f, %.2f", output, prediction))

## [1] "0.99, 0.99"

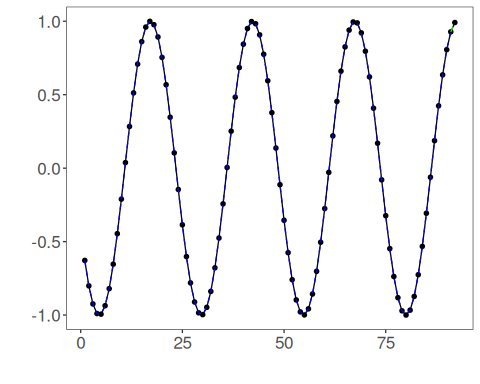
# Evaluate test performance  
  
ev\_test <- evaluate(model, output, prediction)  
print(head(ev\_test$metrics))

## mse smape R2  
## 1 1.49144e-30 1.232084e-15 -Inf

print(sprintf("smape: %.2f", 100\*ev\_test$metrics$smape))

## [1] "smape: 0.00"

# Plot results  
  
yvalues <- c(io\_train$output, io\_test$output)  
plot\_ts\_pred(y=yvalues, yadj=adjust, ypre=prediction) + theme(text = element\_text(size=16))



# Example hyperparameter ranges by model  
  
# ELM  
ranges\_elm <- list(nhid = 1:20, actfun=c('sig', 'radbas', 'tribas', 'relu', 'purelin'))  
  
# MLP  
ranges\_mlp <- list(size = 1:10, decay = seq(0, 1, 1/9), maxit=10000)  
  
# RF  
ranges\_rf <- list(nodesize=1:10, ntree=1:10)  
  
# SVM  
ranges\_svm <- list(kernel=c("radial", "poly", "linear", "sigmoid"), epsilon=seq(0, 1, 0.1), cost=seq(20, 100, 20))  
  
# LSTM  
ranges\_lstm <- list(input\_size = 1:10, epochs=10000)  
  
# CNN  
ranges\_cnn <- list(input\_size = 1:10, epochs=10000)