Time series tuning: Hyperparameter optimization explores a predefined search space to identify configurations that generalize well, typically assessed via cross-validation on the training segment without leaking future information. Searching can be grid- or random-based; for many problems, random search is competitive and simpler. Here, tuning spans both the input window length and the base learner’s key parameters.

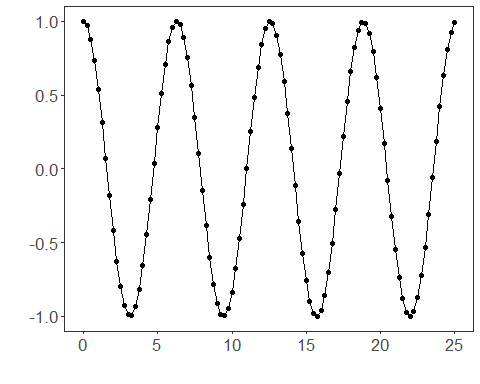
Objective: Perform hyperparameter search (window size and base model parameters) with cross-validation to improve time-series forecasting, and evaluate the result.

# Installing the package (if needed)  
#install.packages("tspredit")

# Loading the packages  
library(daltoolbox)  
library(tspredit)

# Cosine series for study  
  
i <- seq(0, 25, 0.25)  
x <- cos(i)

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



# Sliding windows  
# Create a matrix of windows (t9..t0) from the series for training.  
  
sw\_size <- 10  
ts <- ts\_data(x, sw\_size)  
ts\_head(ts, 3)

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

# Sampling (train and test)  
# Split the data into train and test.  
  
test\_size <- 1  
samp <- ts\_sample(ts, test\_size)  
ts\_head(samp$train, 3)

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

ts\_head(samp$test)

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

# Hyperparameter tuning  
# ts\_tune optimizes base model hyperparameters.  
# In this example, we use ELM with ranges for nhid and activation function.  
  
tune <- ts\_tune(input\_size=c(3:5), base\_model = ts\_elm(ts\_norm\_gminmax()),   
 ranges = list(nhid = 1:5, actfun=c('sig', 'radbas', 'tribas', 'relu', 'purelin')))

# Train projection and fit the best model  
  
io\_train <- ts\_projection(samp$train)  
  
# Generic fit of the chosen model  
model <- fit(tune, x=io\_train$input, y=io\_train$output)

# Fit evaluation (train)  
  
adjust <- predict(model, io\_train$input)  
ev\_adjust <- evaluate(model, io\_train$output, adjust)  
print(head(ev\_adjust$metrics))

## mse smape R2  
## 1 7.56133e-30 1.014266e-14 1

# Forecast on test set  
  
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"

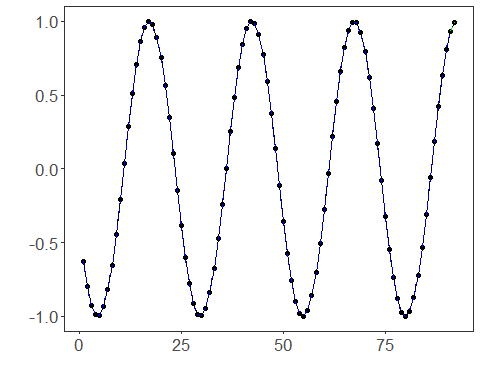
# Test evaluation  
  
ev\_test <- evaluate(model, output, prediction)  
print(head(ev\_test$metrics))

## mse smape R2  
## 1 2.496005e-29 5.040344e-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))



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

References - J. Bergstra and Y. Bengio (2012). Random search for hyper-parameter optimization. Journal of Machine Learning Research, 13, 281–305.