Package ‘tsLSTMx’

January 12, 2024

Type Package

Title Predict Time Series Using LSTM Model Including Exogenous Variable to Denote Zero Values

Version 0.1.0

Author Sandip Garai [aut, cre],
Krishna Pada Sarkar [aut]

Maintainer Sandip Garai <sandipnicksandy@gmail.com>

Description It is a versatile tool for predicting time series data using Long Short-Term Memory (LSTM) models. It is specifically designed to handle time series with an exogenous variable, allowing users to denote whether data was available for a particular period or not. The package encompasses various functionalities, including hyperparameter tuning, custom loss function support, model evaluation, and one-step-ahead forecasting. With an emphasis on ease of use and flexibility, it empowers users to explore, evaluate, and deploy LSTM models for accurate time series predictions and forecasting in diverse applications. More details can be found in Garai and Paul (2023) <doi:10.1016/j.iswa.2023.200202>.

License GPL-3

Encoding UTF-8

RoxygenNote 7.2.3

Imports tensorflow, AllMetrics, keras, reticulate

NeedsCompilation no

Repository CRAN

Date/Publication 2024-01-12 09:30:02 UTC

R topics documented:

`best_model_on_validation` *Evaluate the best LSTM model on the validation set*

Description

This function evaluates the performance of the best LSTM model on the provided validation set.

Usage

```
best_model_on_validation(best_model, X_val, y_val)
```

Arguments

`best_model`	The best LSTM model obtained from hyperparameter tuning.
`X_val`	The validation set input data.
`y_val`	The validation set target data.

Value

The validation loss of the best model on the provided validation set.

Examples

```
data <- data.frame(
  Date = as.Date(c("01-04-18", "02-04-18", "03-04-18", "04-04-18", "05-04-18",
                   "06-04-18", "07-04-18", "08-04-18", "09-04-18", "10-04-18",
                   "11-04-18", "12-04-18", "13-04-18", "14-04-18", "15-04-18",
                   "16-04-18", "17-04-18", "18-04-18", "19-04-18", "20-04-18"),
                 format = "%d-%m-%y"),
  A = c(0, 0, 4, 12, 20, 16, 16, 0, 12, 18, 12, 18, 18, 0, 0, 33, 31, 38, 76, 198)
)
check_and_format_data(data)
# Add a new column 'X' based on the values in the second column
data$X <- ifelse(data$A != 0, 1, 0)

result_embed <- embed_columns(data = data, n_lag = 2)
new_data <- result_embed$data_frame
embedded_colnames <- result_embed$column_names
```

```
result_split <- split_data(new_data = new_data, val_ratio = 0.1)
train_data <- result_split$train_data
validation_data <- result_split$validation_data
train_data <- result_split$train_data
validation_data <- result_split$validation_data
embedded_colnames <- result_embed$column_names
numeric_matrices <- convert_to_numeric_matrices(train_data = train_data,
                                                validation_data = validation_data,
                                                embedded_colnames = embedded_colnames)
X_train <- numeric_matrices$X_train
y_train <- numeric_matrices$y_train
X_val <- numeric_matrices$X_val
y_val <- numeric_matrices$y_val

#' initialize_tensorflow()

X_train <- numeric_matrices$X_train
X_val <- numeric_matrices$X_val
reshaped_data <- reshape_for_lstm(X_train = X_train, X_val = X_val)
X_train <- reshaped_data$X_train
X_val <- reshaped_data$X_val
X_train <- reshaped_data$X_train
y_train <- numeric_matrices$y_train
X_val <- reshaped_data$X_val
y_val <- numeric_matrices$y_val
tf <- reticulate::import("tensorflow")
tensors <- convert_to_tensors(X_train = X_train, y_train = y_train, X_val = X_val, y_val = y_val)
X_train <- tensors$X_train
y_train <- tensors$y_train
X_val <- tensors$X_val
y_val <- tensors$y_val
n_patience <- 50
early_stopping <- define_early_stopping(n_patience = n_patience)

X_train <- tensors$X_train
X_val <- tensors$X_val

y_train <- tensors$y_train
y_val <- tensors$y_val

embedded_colnames <- result_embed$column_names

# Define your custom loss function
custom_loss <- function(y_true, y_pred) {
  condition <- tf$math$equal(y_true, 0)
  loss <- tf$math$reduce_mean(tf$math$square(y_true - y_pred))  # Remove 'axis'
  loss <- tf$where(condition, tf$constant(0), loss)
  return(loss)
}

early_stopping <- define_early_stopping(n_patience = n_patience)
```

```
grid_search_results <- ts_lstm_x_tuning(
  X_train, y_train, X_val, y_val,
  embedded_colnames, custom_loss, early_stopping,
  n_lag = 2, # desired lag value
  lstm_units_list = c(32),
  learning_rate_list = c(0.001, 0.01),
  batch_size_list = c(32),
  dropout_list = c(0.2),
  l1_reg_list = c(0.001),
  l2_reg_list = c(0.001),
  n_iter = 10,
  n_verbose = 0 # or 1
)

results_df <- grid_search_results$results_df
all_histories <- grid_search_results$all_histories
lstm_models <- grid_search_results$lstm_models

# Find the row with the minimum val_loss_mae in results_df
min_val_loss_row <- results_df[which.min(results_df$val_loss_mae), ]

# Extract hyperparameters from the row
best_lstm_units <- min_val_loss_row$lstm_units
best_learning_rate <- min_val_loss_row$learning_rate
best_batch_size <- min_val_loss_row$batch_size
best_n_lag <- min_val_loss_row$n_lag
best_dropout <- min_val_loss_row$dropout
best_l1_reg <- min_val_loss_row$l1_reg
best_l2_reg <- min_val_loss_row$l2_reg

# Generate the lstm_model_name for the best model
best_model_name <- paste0("lstm_model_lu_", best_lstm_units, "_lr_", best_learning_rate,
                          "_bs_", best_batch_size, "_lag_", best_n_lag,
                         "_do_", best_dropout, "_l1_", best_l1_reg, "_l2_", best_l2_reg)

# Generate the history_name for the best model
best_history_name <- paste0("history_lu_", best_lstm_units, "_lr_", best_learning_rate,
                            "_bs_", best_batch_size, "_lag_", best_n_lag,
                         "_do_", best_dropout, "_l1_", best_l1_reg, "_l2_", best_l2_reg)

# Access the best model from lstm_models
best_model <- lstm_models[[best_model_name]]

best_model_details <- data.frame(min_val_loss_row)

colnames(best_model_details) <- colnames(results_df)

# Access the best model from lstm_models
best_history <- all_histories[[best_history_name]]

validation_loss_best <- best_model_on_validation(best_model, X_val, y_val)
```

check_and_format_data *Check and Format Data*

Description

This function checks the compatibility of a given data frame and performs necessary formatting.

Usage

```
check_and_format_data(data, n.head = 6)
```

Arguments

data	A data frame containing a 'Date' column and a numeric column 'A'.
n.head	Number of rows to display from the formatted data frame (default is 6).

Details

This function checks the format of the 'Date' column and ensures it is in the format 'dd-mm-yy'. It also checks the presence of the 'A' column and ensures it contains numeric values.

Value

A formatted data frame with the specified number of rows displayed.

Examples

```
data <- data.frame(
  Date = as.Date(c("01-04-18", "02-04-18", "03-04-18", "04-04-18", "05-04-18",
                   "06-04-18", "07-04-18", "08-04-18", "09-04-18", "10-04-18",
                   "11-04-18", "12-04-18", "13-04-18", "14-04-18", "15-04-18",
                   "16-04-18", "17-04-18", "18-04-18", "19-04-18", "20-04-18"),
                 format = "%d-%m-%y"),
  A = c(0, 0, 4, 12, 20, 16, 16, 0, 12, 18, 12, 18, 18, 0, 0, 33, 31, 38, 76, 198)
)
check_and_format_data(data)
# Add a new column 'X' based on the values in the second column
data$X <- ifelse(data$A != 0, 1, 0)
```

compare_predicted_vs_actual
Compare predicted and actual values for training and validation sets

Description

This function compares the predicted and actual values for the training and validation sets and computes metrics.

Usage

```
compare_predicted_vs_actual(
  train_data,
  validation_data,
  y_train_pred,
  y_val_pred
)
```

Arguments

`train_data`	The training set data, including actual y values.
`validation_data`	The validation set data, including actual y values.
`y_train_pred`	Predicted y values for the training set.
`y_val_pred`	Predicted y values for the validation set.

Value

A list containing data frames with the comparison of actual vs. predicted values for training and validation sets, as well as metrics for the training and validation sets.

Examples

```
data <- data.frame(
  Date = as.Date(c("01-04-18", "02-04-18", "03-04-18", "04-04-18", "05-04-18",
                   "06-04-18", "07-04-18", "08-04-18", "09-04-18", "10-04-18",
                   "11-04-18", "12-04-18", "13-04-18", "14-04-18", "15-04-18",
                   "16-04-18", "17-04-18", "18-04-18", "19-04-18", "20-04-18"),
                 format = "%d-%m-%y"),
  A = c(0, 0, 4, 12, 20, 16, 16, 0, 12, 18, 12, 18, 18, 0, 0, 33, 31, 38, 76, 198)
)
check_and_format_data(data)
# Add a new column 'X' based on the values in the second column
data$X <- ifelse(data$A != 0, 1, 0)

result_embed <- embed_columns(data = data, n_lag = 2)
new_data <- result_embed$data_frame
```

```
embedded_colnames <- result_embed$column_names

result_split <- split_data(new_data = new_data, val_ratio = 0.1)
train_data <- result_split$train_data
validation_data <- result_split$validation_data
train_data <- result_split$train_data
validation_data <- result_split$validation_data
embedded_colnames <- result_embed$column_names
numeric_matrices <- convert_to_numeric_matrices(train_data = train_data,
                                                validation_data = validation_data,
                                                embedded_colnames = embedded_colnames)
X_train <- numeric_matrices$X_train
y_train <- numeric_matrices$y_train
X_val <- numeric_matrices$X_val
y_val <- numeric_matrices$y_val

#' initialize_tensorflow()

X_train <- numeric_matrices$X_train
X_val <- numeric_matrices$X_val
reshaped_data <- reshape_for_lstm(X_train = X_train, X_val = X_val)
X_train <- reshaped_data$X_train
X_val <- reshaped_data$X_val
X_train <- reshaped_data$X_train
y_train <- numeric_matrices$y_train
X_val <- reshaped_data$X_val
y_val <- numeric_matrices$y_val
tf <- reticulate::import("tensorflow")
tensors <- convert_to_tensors(X_train = X_train, y_train = y_train, X_val = X_val, y_val = y_val)
X_train <- tensors$X_train
y_train <- tensors$y_train
X_val <- tensors$X_val
y_val <- tensors$y_val
n_patience <- 50
early_stopping <- define_early_stopping(n_patience = n_patience)

X_train <- tensors$X_train
X_val <- tensors$X_val

y_train <- tensors$y_train
y_val <- tensors$y_val

embedded_colnames <- result_embed$column_names

# Define your custom loss function
custom_loss <- function(y_true, y_pred) {
  condition <- tf$math$equal(y_true, 0)
  loss <- tf$math$reduce_mean(tf$math$square(y_true - y_pred))  # Remove 'axis'
  loss <- tf$where(condition, tf$constant(0), loss)
  return(loss)
}

early_stopping <- define_early_stopping(n_patience = n_patience)
```

```
grid_search_results <- ts_lstm_x_tuning(
  X_train, y_train, X_val, y_val,
  embedded_colnames, custom_loss, early_stopping,
  n_lag = 2, # desired lag value
  lstm_units_list = c(32),
  learning_rate_list = c(0.001, 0.01),
  batch_size_list = c(32),
  dropout_list = c(0.2),
  l1_reg_list = c(0.001),
  l2_reg_list = c(0.001),
  n_iter = 10,
  n_verbose = 0 # or 1
)

results_df <- grid_search_results$results_df
all_histories <- grid_search_results$all_histories
lstm_models <- grid_search_results$lstm_models

# Find the row with the minimum val_loss_mae in results_df
min_val_loss_row <- results_df[which.min(results_df$val_loss_mae), ]

# Extract hyperparameters from the row
best_lstm_units <- min_val_loss_row$lstm_units
best_learning_rate <- min_val_loss_row$learning_rate
best_batch_size <- min_val_loss_row$batch_size
best_n_lag <- min_val_loss_row$n_lag
best_dropout <- min_val_loss_row$dropout
best_l1_reg <- min_val_loss_row$l1_reg
best_l2_reg <- min_val_loss_row$l2_reg

# Generate the lstm_model_name for the best model
best_model_name <- paste0("lstm_model_lu_", best_lstm_units, "_lr_", best_learning_rate,
                          "_bs_", best_batch_size, "_lag_", best_n_lag,
                         "_do_", best_dropout, "_l1_", best_l1_reg, "_l2_", best_l2_reg)

# Generate the history_name for the best model
best_history_name <- paste0("history_lu_", best_lstm_units, "_lr_", best_learning_rate,
                            "_bs_", best_batch_size, "_lag_", best_n_lag,
                         "_do_", best_dropout, "_l1_", best_l1_reg, "_l2_", best_l2_reg)

# Access the best model from lstm_models
best_model <- lstm_models[[best_model_name]]

best_model_details <- data.frame(min_val_loss_row)

colnames(best_model_details) <- colnames(results_df)

# Access the best model from lstm_models
best_history <- all_histories[[best_history_name]]

validation_loss_best <- best_model_on_validation(best_model, X_val, y_val)
predicted_values <- predict_y_values(best_model, X_train, X_val, train_data, validation_data)
```

```
y_train_pred <- predicted_values$y_train_pred
y_val_pred <- predicted_values$y_val_pred
comparison <- compare_predicted_vs_actual(train_data, validation_data, y_train_pred, y_val_pred)
compare_train <- comparison$compare_train
compare_val <- comparison$compare_val
metrics_train <- comparison$metrics_train
metrics_val <- comparison$metrics_val
```

`convert_to_numeric_matrices` *Function to convert columns to numeric matrices*

Description

This function converts specific columns in the data frames to numeric matrices.

Usage

```
convert_to_numeric_matrices(train_data, validation_data, embedded_colnames)
```

Arguments

`train_data`	Training data frame.
`validation_data`	Validation data frame.
`embedded_colnames`	Names of the embedded columns.

Value

A list containing numeric matrices for training and validation sets.

Examples

```
data <- data.frame(
  Date = as.Date(c("01-04-18", "02-04-18", "03-04-18", "04-04-18", "05-04-18",
                   "06-04-18", "07-04-18", "08-04-18", "09-04-18", "10-04-18",
                   "11-04-18", "12-04-18", "13-04-18", "14-04-18", "15-04-18",
                   "16-04-18", "17-04-18", "18-04-18", "19-04-18", "20-04-18"),
                 format = "%d-%m-%y"),
  A = c(0, 0, 4, 12, 20, 16, 16, 0, 12, 18, 12, 18, 18, 0, 0, 33, 31, 38, 76, 198)
)
check_and_format_data(data)
# Add a new column 'X' based on the values in the second column
data$X <- ifelse(data$A != 0, 1, 0)
```

```
result_embed <- embed_columns(data = data, n_lag = 2)
new_data <- result_embed$data_frame
embedded_colnames <- result_embed$column_names

result_split <- split_data(new_data = new_data, val_ratio = 0.1)
train_data <- result_split$train_data
validation_data <- result_split$validation_data
train_data <- result_split$train_data
validation_data <- result_split$validation_data
embedded_colnames <- result_embed$column_names
numeric_matrices <- convert_to_numeric_matrices(train_data = train_data,
                                                validation_data = validation_data,
                                                embedded_colnames = embedded_colnames)
X_train <- numeric_matrices$X_train
y_train <- numeric_matrices$y_train
X_val <- numeric_matrices$X_val
y_val <- numeric_matrices$y_val

#' initialize_tensorflow()
```

`convert_to_tensors` *Function to convert data to TensorFlow tensors*

Description

This function converts input data to TensorFlow tensors for compatibility with TensorFlow and keras models.

Usage

```
convert_to_tensors(X_train, y_train, X_val, y_val)
```

Arguments

`X_train`	Numeric matrix representing the training input data.
`y_train`	Numeric vector representing the training output data.
`X_val`	Numeric matrix representing the validation input data.
`y_val`	Numeric vector representing the validation output data.

Value

A list containing TensorFlow tensors for training and validation data.

Examples

```
data <- data.frame(
  Date = as.Date(c("01-04-18", "02-04-18", "03-04-18", "04-04-18", "05-04-18",
                   "06-04-18", "07-04-18", "08-04-18", "09-04-18", "10-04-18",
                   "11-04-18", "12-04-18", "13-04-18", "14-04-18", "15-04-18",
                   "16-04-18", "17-04-18", "18-04-18", "19-04-18", "20-04-18"),
                 format = "%d-%m-%y"),
  A = c(0, 0, 4, 12, 20, 16, 16, 0, 12, 18, 12, 18, 18, 0, 0, 33, 31, 38, 76, 198)
)
check_and_format_data(data)
# Add a new column 'X' based on the values in the second column
data$X <- ifelse(data$A != 0, 1, 0)

result_embed <- embed_columns(data = data, n_lag = 2)
new_data <- result_embed$data_frame
embedded_colnames <- result_embed$column_names

result_split <- split_data(new_data = new_data, val_ratio = 0.1)
train_data <- result_split$train_data
validation_data <- result_split$validation_data
train_data <- result_split$train_data
validation_data <- result_split$validation_data
embedded_colnames <- result_embed$column_names
numeric_matrices <- convert_to_numeric_matrices(train_data = train_data,
                                                validation_data = validation_data,
                                                embedded_colnames = embedded_colnames)
X_train <- numeric_matrices$X_train
y_train <- numeric_matrices$y_train
X_val <- numeric_matrices$X_val
y_val <- numeric_matrices$y_val

#' initialize_tensorflow()

X_train <- numeric_matrices$X_train
X_val <- numeric_matrices$X_val
reshaped_data <- reshape_for_lstm(X_train = X_train, X_val = X_val)
X_train <- reshaped_data$X_train
X_val <- reshaped_data$X_val
X_train <- reshaped_data$X_train
y_train <- numeric_matrices$y_train
X_val <- reshaped_data$X_val
y_val <- numeric_matrices$y_val
tf <- reticulate::import("tensorflow")
tensors <- convert_to_tensors(X_train = X_train, y_train = y_train, X_val = X_val, y_val = y_val)
X_train <- tensors$X_train
y_train <- tensors$y_train
X_val <- tensors$X_val
y_val <- tensors$y_val
```

define_early_stopping *Function to define early stopping callback*

Description

This function defines an early stopping callback for keras models.

Usage

```
define_early_stopping(n_patience)
```

Arguments

n_patience	Integer specifying the number of epochs with no improvement after which training will be stopped.

Value

A keras early stopping callback.

Examples

```
data <- data.frame(
  Date = as.Date(c("01-04-18", "02-04-18", "03-04-18", "04-04-18", "05-04-18",
                   "06-04-18", "07-04-18", "08-04-18", "09-04-18", "10-04-18",
                   "11-04-18", "12-04-18", "13-04-18", "14-04-18", "15-04-18",
                   "16-04-18", "17-04-18", "18-04-18", "19-04-18", "20-04-18"),
                 format = "%d-%m-%y"),
  A = c(0, 0, 4, 12, 20, 16, 16, 0, 12, 18, 12, 18, 18, 0, 0, 33, 31, 38, 76, 198)
)
check_and_format_data(data)
# Add a new column 'X' based on the values in the second column
data$X <- ifelse(data$A != 0, 1, 0)

result_embed <- embed_columns(data = data, n_lag = 2)
new_data <- result_embed$data_frame
embedded_colnames <- result_embed$column_names

result_split <- split_data(new_data = new_data, val_ratio = 0.1)
train_data <- result_split$train_data
validation_data <- result_split$validation_data
train_data <- result_split$train_data
validation_data <- result_split$validation_data
embedded_colnames <- result_embed$column_names
numeric_matrices <- convert_to_numeric_matrices(train_data = train_data,
                                                validation_data = validation_data,
                                                embedded_colnames = embedded_colnames)
X_train <- numeric_matrices$X_train
y_train <- numeric_matrices$y_train
```

```
X_val <- numeric_matrices$X_val
y_val <- numeric_matrices$y_val

#' initialize_tensorflow()

X_train <- numeric_matrices$X_train
X_val <- numeric_matrices$X_val
reshaped_data <- reshape_for_lstm(X_train = X_train, X_val = X_val)
X_train <- reshaped_data$X_train
X_val <- reshaped_data$X_val
X_train <- reshaped_data$X_train
y_train <- numeric_matrices$y_train
X_val <- reshaped_data$X_val
y_val <- numeric_matrices$y_val
tf <- reticulate::import("tensorflow")
tensors <- convert_to_tensors(X_train = X_train, y_train = y_train, X_val = X_val, y_val = y_val)
X_train <- tensors$X_train
y_train <- tensors$y_train
X_val <- tensors$X_val
y_val <- tensors$y_val
n_patience <- 50
early_stopping <- define_early_stopping(n_patience = n_patience)
```

`embed_columns` *Embed columns and create a new data frame*

Description

This function takes a data frame and embeds specified columns to create a new data frame.

Usage

```
embed_columns(data, n_lag = 2)
```

Arguments

data	A data frame containing the original columns.
n_lag	Number of lags for embedding.

Value

A list containing the new data frame and column names of the embedded columns.

Examples

```
data <- data.frame(
  Date = as.Date(c("01-04-18", "02-04-18", "03-04-18", "04-04-18", "05-04-18",
                   "06-04-18", "07-04-18", "08-04-18", "09-04-18", "10-04-18",
                   "11-04-18", "12-04-18", "13-04-18", "14-04-18", "15-04-18",
                   "16-04-18", "17-04-18", "18-04-18", "19-04-18", "20-04-18"),
                 format = "%d-%m-%y"),
  A = c(0, 0, 4, 12, 20, 16, 16, 0, 12, 18, 12, 18, 18, 0, 0, 33, 31, 38, 76, 198)
)
check_and_format_data(data)
# Add a new column 'X' based on the values in the second column
data$X <- ifelse(data$A != 0, 1, 0)

result_embed <- embed_columns(data = data, n_lag = 2)
new_data <- result_embed$data_frame
embedded_colnames <- result_embed$column_names
```

---

forecast_best_model *Perform forecasting using the best model*

Description

This function performs forecasting using the best-trained model.

Usage

```
forecast_best_model(
  best_model,
  best_learning_rate,
  custom_loss,
  n_lag = 2,
  new_data,
  test,
  forecast_steps
)
```

Arguments

best_model	The best-trained LSTM model.
best_learning_rate	The best learning rate used during training.
custom_loss	The custom loss function used during training.
n_lag	The lag value used during training.
new_data	The input data for forecasting.
test	The test data frame containing the input data for forecasting.
forecast_steps	The number of steps to forecast.

Value

A list containing the forecasted values, actual vs. forecasted data frame, and metrics for forecasting.

Examples

```
data <- data.frame(
  Date = as.Date(c("01-04-18", "02-04-18", "03-04-18", "04-04-18", "05-04-18",
                   "06-04-18", "07-04-18", "08-04-18", "09-04-18", "10-04-18",
                   "11-04-18", "12-04-18", "13-04-18", "14-04-18", "15-04-18",
                   "16-04-18", "17-04-18", "18-04-18", "19-04-18", "20-04-18"),
                 format = "%d-%m-%y"),
  A = c(0, 0, 4, 12, 20, 16, 16, 0, 12, 18, 12, 18, 18, 0, 0, 33, 31, 38, 76, 198)
)
check_and_format_data(data)
# Add a new column 'X' based on the values in the second column
data$X <- ifelse(data$A != 0, 1, 0)

result_embed <- embed_columns(data = data, n_lag = 2)
new_data <- result_embed$data_frame
embedded_colnames <- result_embed$column_names

result_split <- split_data(new_data = new_data, val_ratio = 0.1)
train_data <- result_split$train_data
validation_data <- result_split$validation_data
train_data <- result_split$train_data
validation_data <- result_split$validation_data
embedded_colnames <- result_embed$column_names
numeric_matrices <- convert_to_numeric_matrices(train_data = train_data,
                                                validation_data = validation_data,
                                                embedded_colnames = embedded_colnames)
X_train <- numeric_matrices$X_train
y_train <- numeric_matrices$y_train
X_val <- numeric_matrices$X_val
y_val <- numeric_matrices$y_val

#' initialize_tensorflow()

X_train <- numeric_matrices$X_train
X_val <- numeric_matrices$X_val
reshaped_data <- reshape_for_lstm(X_train = X_train, X_val = X_val)
X_train <- reshaped_data$X_train
X_val <- reshaped_data$X_val
X_train <- reshaped_data$X_train
y_train <- numeric_matrices$y_train
X_val <- reshaped_data$X_val
y_val <- numeric_matrices$y_val
tf <- reticulate::import("tensorflow")
tensors <- convert_to_tensors(X_train = X_train, y_train = y_train, X_val = X_val, y_val = y_val)
X_train <- tensors$X_train
y_train <- tensors$y_train
X_val <- tensors$X_val
```

```
y_val <- tensors$y_val
n_patience <- 50
early_stopping <- define_early_stopping(n_patience = n_patience)

X_train <- tensors$X_train
X_val <- tensors$X_val

y_train <- tensors$y_train
y_val <- tensors$y_val

embedded_colnames <- result_embed$column_names

# Define your custom loss function
custom_loss <- function(y_true, y_pred) {
  condition <- tf$math$equal(y_true, 0)
  loss <- tf$math$reduce_mean(tf$math$square(y_true - y_pred))  # Remove 'axis'
  loss <- tf$where(condition, tf$constant(0), loss)
  return(loss)
}

early_stopping <- define_early_stopping(n_patience = n_patience)

grid_search_results <- ts_lstm_x_tuning(
  X_train, y_train, X_val, y_val,
  embedded_colnames, custom_loss, early_stopping,
  n_lag = 2, # desired lag value
  lstm_units_list = c(32),
  learning_rate_list = c(0.001, 0.01),
  batch_size_list = c(32),
  dropout_list = c(0.2),
  l1_reg_list = c(0.001),
  l2_reg_list = c(0.001),
  n_iter = 10,
  n_verbose = 0 # or 1
)

results_df <- grid_search_results$results_df
all_histories <- grid_search_results$all_histories
lstm_models <- grid_search_results$lstm_models

# Find the row with the minimum val_loss_mae in results_df
min_val_loss_row <- results_df[which.min(results_df$val_loss_mae), ]

# Extract hyperparameters from the row
best_lstm_units <- min_val_loss_row$lstm_units
best_learning_rate <- min_val_loss_row$learning_rate
best_batch_size <- min_val_loss_row$batch_size
best_n_lag <- min_val_loss_row$n_lag
best_dropout <- min_val_loss_row$dropout
best_l1_reg <- min_val_loss_row$l1_reg
best_l2_reg <- min_val_loss_row$l2_reg

# Generate the lstm_model_name for the best model
```

```
best_model_name <- paste0("lstm_model_lu_", best_lstm_units, "_lr_", best_learning_rate,
                          "_bs_", best_batch_size, "_lag_", best_n_lag,
                         "_do_", best_dropout, "_l1_", best_l1_reg, "_l2_", best_l2_reg)

# Generate the history_name for the best model
best_history_name <- paste0("history_lu_", best_lstm_units, "_lr_", best_learning_rate,
                            "_bs_", best_batch_size, "_lag_", best_n_lag,
                         "_do_", best_dropout, "_l1_", best_l1_reg, "_l2_", best_l2_reg)

# Access the best model from lstm_models
best_model <- lstm_models[[best_model_name]]

best_model_details <- data.frame(min_val_loss_row)

colnames(best_model_details) <- colnames(results_df)

# Access the best model from lstm_models
best_history <- all_histories[[best_history_name]]

validation_loss_best <- best_model_on_validation(best_model, X_val, y_val)
predicted_values <- predict_y_values(best_model, X_train, X_val, train_data, validation_data)
y_train_pred <- predicted_values$y_train_pred
y_val_pred <- predicted_values$y_val_pred
comparison <- compare_predicted_vs_actual(train_data, validation_data, y_train_pred, y_val_pred)
compare_train <- comparison$compare_train
compare_val <- comparison$compare_val
metrics_train <- comparison$metrics_train
metrics_val <- comparison$metrics_val

test <- data.frame(
  Date = as.Date(c("01-04-23", "02-04-23", "03-04-23", "04-04-23", "05-04-23",
                   "06-04-23", "07-04-23", "08-04-23", "09-04-23", "10-04-23",
                   "11-04-23", "12-04-23", "13-04-23", "14-04-23", "15-04-23",
                   "16-04-23", "17-04-23", "18-04-23", "19-04-23", "20-04-23"),
                 format = "%d-%m-%y"),
  A = c(0, 0, 15, 4, -31, 24, 14, 0, 0, 33, 38, 33, 29, 29, 25, 0, 44, 67, 162, 278)
)

test$X <- ifelse(test$A != 0, 1, 0)

n_forecast <- nrow(test)

# Perform one-step-ahead forecasting
forecast_steps <- n_forecast
current_row <- nrow(new_data)
forecast_results <- forecast_best_model(best_model, best_learning_rate,
                                        custom_loss, n_lag = 2,
                                        new_data, test,
                                        forecast_steps)

# Access the results
forecast_values <- forecast_results$forecast_values
actual_vs_forecast <- forecast_results$actual_vs_forecast
```

```
metrics_forecast <- forecast_results$metrics_forecast
```

`initialize_tensorflow` *Function to initialize TensorFlow and enable eager execution*

Description

This function initializes TensorFlow and enables eager execution.

Usage

```
initialize_tensorflow()
```

Value

No return value, called for smooth running

Examples

```
initialize_tensorflow()
```

`predict_y_values` *Predict y values for the training and validation sets using the best LSTM model*

Description

This function predicts y values for the training and validation sets using the provided LSTM model.

Usage

```
predict_y_values(best_model, X_train, X_val, train_data, validation_data)
```

Arguments

`best_model`	The best LSTM model obtained from hyperparameter tuning.
`X_train`	The training set input data.
`X_val`	The validation set input data.
`train_data`	The training set data, including x values.
`validation_data`	The validation set data, including x values.

Value

A list containing the predicted y values for the training and validation sets.

Examples

```
data <- data.frame(
  Date = as.Date(c("01-04-18", "02-04-18", "03-04-18", "04-04-18", "05-04-18",
                   "06-04-18", "07-04-18", "08-04-18", "09-04-18", "10-04-18",
                   "11-04-18", "12-04-18", "13-04-18", "14-04-18", "15-04-18",
                   "16-04-18", "17-04-18", "18-04-18", "19-04-18", "20-04-18"),
                 format = "%d-%m-%y"),
  A = c(0, 0, 4, 12, 20, 16, 16, 0, 12, 18, 12, 18, 18, 0, 0, 33, 31, 38, 76, 198)
)
check_and_format_data(data)
# Add a new column 'X' based on the values in the second column
data$X <- ifelse(data$A != 0, 1, 0)

result_embed <- embed_columns(data = data, n_lag = 2)
new_data <- result_embed$data_frame
embedded_colnames <- result_embed$column_names

result_split <- split_data(new_data = new_data, val_ratio = 0.1)
train_data <- result_split$train_data
validation_data <- result_split$validation_data
train_data <- result_split$train_data
validation_data <- result_split$validation_data
embedded_colnames <- result_embed$column_names
numeric_matrices <- convert_to_numeric_matrices(train_data = train_data,
                                                validation_data = validation_data,
                                                embedded_colnames = embedded_colnames)
X_train <- numeric_matrices$X_train
y_train <- numeric_matrices$y_train
X_val <- numeric_matrices$X_val
y_val <- numeric_matrices$y_val

#' initialize_tensorflow()

X_train <- numeric_matrices$X_train
X_val <- numeric_matrices$X_val
reshaped_data <- reshape_for_lstm(X_train = X_train, X_val = X_val)
X_train <- reshaped_data$X_train
X_val <- reshaped_data$X_val
X_train <- reshaped_data$X_train
y_train <- numeric_matrices$y_train
X_val <- reshaped_data$X_val
y_val <- numeric_matrices$y_val
tf <- reticulate::import("tensorflow")
tensors <- convert_to_tensors(X_train = X_train, y_train = y_train, X_val = X_val, y_val = y_val)
X_train <- tensors$X_train
y_train <- tensors$y_train
X_val <- tensors$X_val
```

```
y_val <- tensors$y_val
n_patience <- 50
early_stopping <- define_early_stopping(n_patience = n_patience)

X_train <- tensors$X_train
X_val <- tensors$X_val

y_train <- tensors$y_train
y_val <- tensors$y_val

embedded_colnames <- result_embed$column_names

# Define your custom loss function
custom_loss <- function(y_true, y_pred) {
  condition <- tf$math$equal(y_true, 0)
  loss <- tf$math$reduce_mean(tf$math$square(y_true - y_pred))  # Remove 'axis'
  loss <- tf$where(condition, tf$constant(0), loss)
  return(loss)
}

early_stopping <- define_early_stopping(n_patience = n_patience)

grid_search_results <- ts_lstm_x_tuning(
  X_train, y_train, X_val, y_val,
  embedded_colnames, custom_loss, early_stopping,
  n_lag = 2, # desired lag value
  lstm_units_list = c(32),
  learning_rate_list = c(0.001, 0.01),
  batch_size_list = c(32),
  dropout_list = c(0.2),
  l1_reg_list = c(0.001),
  l2_reg_list = c(0.001),
  n_iter = 10,
  n_verbose = 0 # or 1
)

results_df <- grid_search_results$results_df
all_histories <- grid_search_results$all_histories
lstm_models <- grid_search_results$lstm_models

# Find the row with the minimum val_loss_mae in results_df
min_val_loss_row <- results_df[which.min(results_df$val_loss_mae), ]

# Extract hyperparameters from the row
best_lstm_units <- min_val_loss_row$lstm_units
best_learning_rate <- min_val_loss_row$learning_rate
best_batch_size <- min_val_loss_row$batch_size
best_n_lag <- min_val_loss_row$n_lag
best_dropout <- min_val_loss_row$dropout
best_l1_reg <- min_val_loss_row$l1_reg
best_l2_reg <- min_val_loss_row$l2_reg

# Generate the lstm_model_name for the best model
```

```
best_model_name <- paste0("lstm_model_lu_", best_lstm_units, "_lr_", best_learning_rate,
                           "_bs_", best_batch_size, "_lag_", best_n_lag,
                          "_do_", best_dropout, "_l1_", best_l1_reg, "_l2_", best_l2_reg)

# Generate the history_name for the best model
best_history_name <- paste0("history_lu_", best_lstm_units, "_lr_", best_learning_rate,
                             "_bs_", best_batch_size, "_lag_", best_n_lag,
                          "_do_", best_dropout, "_l1_", best_l1_reg, "_l2_", best_l2_reg)

# Access the best model from lstm_models
best_model <- lstm_models[[best_model_name]]

best_model_details <- data.frame(min_val_loss_row)

colnames(best_model_details) <- colnames(results_df)

# Access the best model from lstm_models
best_history <- all_histories[[best_history_name]]

validation_loss_best <- best_model_on_validation(best_model, X_val, y_val)
predicted_values <- predict_y_values(best_model, X_train, X_val, train_data, validation_data)
y_train_pred <- predicted_values$y_train_pred
y_val_pred <- predicted_values$y_val_pred
```

`reshape_for_lstm` *Function to reshape input data for LSTM*

Description

This function reshapes input data to be compatible with LSTM models.

Usage

```
reshape_for_lstm(X_train, X_val)
```

Arguments

`X_train`	Numeric matrix representing the training input data.
`X_val`	Numeric matrix representing the validation input data.

Value

A list containing reshaped training and validation input data.

Examples

```
data <- data.frame(
  Date = as.Date(c("01-04-18", "02-04-18", "03-04-18", "04-04-18", "05-04-18",
                   "06-04-18", "07-04-18", "08-04-18", "09-04-18", "10-04-18",
                   "11-04-18", "12-04-18", "13-04-18", "14-04-18", "15-04-18",
                   "16-04-18", "17-04-18", "18-04-18", "19-04-18", "20-04-18"),
                 format = "%d-%m-%y"),
  A = c(0, 0, 4, 12, 20, 16, 16, 0, 12, 18, 12, 18, 18, 0, 0, 33, 31, 38, 76, 198)
)
check_and_format_data(data)
# Add a new column 'X' based on the values in the second column
data$X <- ifelse(data$A != 0, 1, 0)

result_embed <- embed_columns(data = data, n_lag = 2)
new_data <- result_embed$data_frame
embedded_colnames <- result_embed$column_names

result_split <- split_data(new_data = new_data, val_ratio = 0.1)
train_data <- result_split$train_data
validation_data <- result_split$validation_data
train_data <- result_split$train_data
validation_data <- result_split$validation_data
embedded_colnames <- result_embed$column_names
numeric_matrices <- convert_to_numeric_matrices(train_data = train_data,
                                                validation_data = validation_data,
                                                embedded_colnames = embedded_colnames)
X_train <- numeric_matrices$X_train
y_train <- numeric_matrices$y_train
X_val <- numeric_matrices$X_val
y_val <- numeric_matrices$y_val

#' initialize_tensorflow()

X_train <- numeric_matrices$X_train
X_val <- numeric_matrices$X_val
reshaped_data <- reshape_for_lstm(X_train = X_train, X_val = X_val)
X_train <- reshaped_data$X_train
X_val <- reshaped_data$X_val
```

split_data   *Split data into training and validation sets*

Description

This function takes a data frame and splits it into training and validation sets.

Usage

```
split_data(new_data, val_ratio = 0.1)
```

Arguments

`new_data`	The data frame to be split.
`val_ratio`	The ratio of the data to be used for validation (default is 0.1).

Value

A list containing the training and validation data frames.

Examples

```
data <- data.frame(
  Date = as.Date(c("01-04-18", "02-04-18", "03-04-18", "04-04-18", "05-04-18",
                   "06-04-18", "07-04-18", "08-04-18", "09-04-18", "10-04-18",
                   "11-04-18", "12-04-18", "13-04-18", "14-04-18", "15-04-18",
                   "16-04-18", "17-04-18", "18-04-18", "19-04-18", "20-04-18"),
                 format = "%d-%m-%y"),
  A = c(0, 0, 4, 12, 20, 16, 16, 0, 12, 18, 12, 18, 18, 0, 0, 33, 31, 38, 76, 198)
)
check_and_format_data(data)
# Add a new column 'X' based on the values in the second column
data$X <- ifelse(data$A != 0, 1, 0)

result_embed <- embed_columns(data = data, n_lag = 2)
new_data <- result_embed$data_frame
embedded_colnames <- result_embed$column_names

result_split <- split_data(new_data = new_data, val_ratio = 0.1)
train_data <- result_split$train_data
validation_data <- result_split$validation_data
```

`ts_lstm_x_tuning` *Time Series LSTM Hyperparameter Tuning*

Description

This function performs hyperparameter tuning for a Time Series LSTM model using a grid search approach.

Usage

```
ts_lstm_x_tuning(
  X_train,
  y_train,
  X_val,
  y_val,
  embedded_colnames,
  custom_loss,
  early_stopping,
  n_lag = 2,
  lstm_units_list = c(32),
  learning_rate_list = c(0.001, 0.01),
  batch_size_list = c(32),
  dropout_list = c(0.2),
  l1_reg_list = c(0.001),
  l2_reg_list = c(0.001),
  n_iter = 10,
  n_verbose = 0
)
```

Arguments

`X_train`	Numeric matrix, the training input data.
`y_train`	Numeric vector, the training target data.
`X_val`	Numeric matrix, the validation input data.
`y_val`	Numeric vector, the validation target data.
`embedded_colnames`	Character vector, column names of the embedded features.
`custom_loss`	Function, custom loss function for the LSTM model.
`early_stopping`	keras early stopping callback.
`n_lag`	Integer, desired lag value.
`lstm_units_list`	Numeric vector, list of LSTM units to search over.
`learning_rate_list`	Numeric vector, list of learning rates to search over.
`batch_size_list`	Numeric vector, list of batch sizes to search over.
`dropout_list`	Numeric vector, list of dropout rates to search over.
`l1_reg_list`	Numeric vector, list of L1 regularization values to search over.
`l2_reg_list`	Numeric vector, list of L2 regularization values to search over.
`n_iter`	Integer, number of epochs for each model training.
`n_verbose`	Integer, level of verbosity during training (0 or 1).

Value

A list containing the results data frame, all histories, and LSTM models.

References

Garai, S., & Paul, R. K. (2023). Development of MCS based-ensemble models using CEEMDAN decomposition and machine intelligence. Intelligent Systems with Applications, 18, 200202.

Examples

```
data <- data.frame(
  Date = as.Date(c("01-04-18", "02-04-18", "03-04-18", "04-04-18", "05-04-18",
                   "06-04-18", "07-04-18", "08-04-18", "09-04-18", "10-04-18",
                   "11-04-18", "12-04-18", "13-04-18", "14-04-18", "15-04-18",
                   "16-04-18", "17-04-18", "18-04-18", "19-04-18", "20-04-18"),
                 format = "%d-%m-%y"),
  A = c(0, 0, 4, 12, 20, 16, 16, 0, 12, 18, 12, 18, 18, 0, 0, 33, 31, 38, 76, 198)
)
check_and_format_data(data)
# Add a new column 'X' based on the values in the second column
data$X <- ifelse(data$A != 0, 1, 0)

result_embed <- embed_columns(data = data, n_lag = 2)
new_data <- result_embed$data_frame
embedded_colnames <- result_embed$column_names

result_split <- split_data(new_data = new_data, val_ratio = 0.1)
train_data <- result_split$train_data
validation_data <- result_split$validation_data
train_data <- result_split$train_data
validation_data <- result_split$validation_data
embedded_colnames <- result_embed$column_names
numeric_matrices <- convert_to_numeric_matrices(train_data = train_data,
                                                validation_data = validation_data,
                                                embedded_colnames = embedded_colnames)
X_train <- numeric_matrices$X_train
y_train <- numeric_matrices$y_train
X_val <- numeric_matrices$X_val
y_val <- numeric_matrices$y_val

#' initialize_tensorflow()

X_train <- numeric_matrices$X_train
X_val <- numeric_matrices$X_val
reshaped_data <- reshape_for_lstm(X_train = X_train, X_val = X_val)
X_train <- reshaped_data$X_train
X_val <- reshaped_data$X_val
X_train <- reshaped_data$X_train
y_train <- numeric_matrices$y_train
X_val <- reshaped_data$X_val
y_val <- numeric_matrices$y_val
```

```
tf <- reticulate::import("tensorflow")
tensors <- convert_to_tensors(X_train = X_train, y_train = y_train, X_val = X_val, y_val = y_val)
X_train <- tensors$X_train
y_train <- tensors$y_train
X_val <- tensors$X_val
y_val <- tensors$y_val
n_patience <- 50
early_stopping <- define_early_stopping(n_patience = n_patience)

X_train <- tensors$X_train
X_val <- tensors$X_val

y_train <- tensors$y_train
y_val <- tensors$y_val

embedded_colnames <- result_embed$column_names

# Define your custom loss function
custom_loss <- function(y_true, y_pred) {
  condition <- tf$math$equal(y_true, 0)
  loss <- tf$math$reduce_mean(tf$math$square(y_true - y_pred))  # Remove 'axis'
  loss <- tf$where(condition, tf$constant(0), loss)
  return(loss)
}

early_stopping <- define_early_stopping(n_patience = n_patience)

grid_search_results <- ts_lstm_x_tuning(
  X_train, y_train, X_val, y_val,
  embedded_colnames, custom_loss, early_stopping,
  n_lag = 2, # desired lag value
  lstm_units_list = c(32),
  learning_rate_list = c(0.001, 0.01),
  batch_size_list = c(32),
  dropout_list = c(0.2),
  l1_reg_list = c(0.001),
  l2_reg_list = c(0.001),
  n_iter = 10,
  n_verbose = 0 # or 1
)

results_df <- grid_search_results$results_df
all_histories <- grid_search_results$all_histories
lstm_models <- grid_search_results$lstm_models

# Find the row with the minimum val_loss_mae in results_df
min_val_loss_row <- results_df[which.min(results_df$val_loss_mae), ]

# Extract hyperparameters from the row
best_lstm_units <- min_val_loss_row$lstm_units
best_learning_rate <- min_val_loss_row$learning_rate
best_batch_size <- min_val_loss_row$batch_size
best_n_lag <- min_val_loss_row$n_lag
```

```
best_dropout <- min_val_loss_row$dropout
best_l1_reg <- min_val_loss_row$l1_reg
best_l2_reg <- min_val_loss_row$l2_reg

# Generate the lstm_model_name for the best model
best_model_name <- paste0("lstm_model_lu_", best_lstm_units, "_lr_", best_learning_rate,
                          "_bs_", best_batch_size, "_lag_", best_n_lag,
                         "_do_", best_dropout, "_l1_", best_l1_reg, "_l2_", best_l2_reg)

# Generate the history_name for the best model
best_history_name <- paste0("history_lu_", best_lstm_units, "_lr_", best_learning_rate,
                            "_bs_", best_batch_size, "_lag_", best_n_lag,
                         "_do_", best_dropout, "_l1_", best_l1_reg, "_l2_", best_l2_reg)

# Access the best model from lstm_models
best_model <- lstm_models[[best_model_name]]

best_model_details <- data.frame(min_val_loss_row)

colnames(best_model_details) <- colnames(results_df)

# Access the best model from lstm_models
best_history <- all_histories[[best_history_name]]
```

Index