Industrial Metals Forecaster Documentation

Release 1.0.0

Oliver Boom

Aug 09, 2019

CONTENTS:

CHAPTER

ONE

PREPROCESSING MODULE

`Forecaster.preprocessing.`**`clean_data`**(*df*, *n_std=20*)

Removes any outliers that are further than a chosen number of standard deviations from the mean.

These values are most likely wrongly inputted data, and so are forward filled.

Parameters

- **`df`** (*`pd.DataFrame`*) – A time series
- **`n_std`** (*`int`*) – The number of standard deviations from the mean

Returns The cleaned time series

Return type pd.DataFrame

`Forecaster.preprocessing.`**`clean_dict_gen`**(*universe_dict*)

Generates a dictionary of cleaned DataFrames

Parameters **`universe_dict`** (*`dict`*) – The dictionary of time series

Returns The cleaned dictionary of time series

Return type dict

`Forecaster.preprocessing.`**`column_rename`**(*universe_dict*)

Appends the name of the instrument to the columns. To help keep track of the instruments in the full dataset.

Parameters **`universe_dict`** (*`dict`*) – The dictionary of time series

Returns The dictionary of time series

Return type dict

`Forecaster.preprocessing.`**`dimension_reduce`**(*data_X*, *n_dim*)

Performing PCA to reduce the dimensionality of the data.

Parameters

- **`data_X`** (*`np.array`*) – The dataset to perform reduction on
- **`n_dim`** (*`int`*) – Number of dimensions to reduce to

Returns The reduced dataset

Return type np.array

`Forecaster.preprocessing.`**`dimension_selector`**(*data_X*, *thresh=0.98*)

Calculated the number of dimensions required to reach a threshold level of variance.

Completes a PCA reduction to an increasing number of dimensions and calculates the total variance achieved for each reduction. If the reduction is above the threshold then that number of dimensions is returned

Parameters

- **`data_X`** (*`np.array`*) – The dataset to perform reduction on
- **`thresh`** (*`float`*) – The amount of variance that must be contained the in reduced dataset

Returns The column dimensionality required to contain the threshold variance

Return type int

`Forecaster.preprocessing.`**`feature_spawn`**(*df*)

Takes a time series and spawns several new features that explicitly detail information about the series.

The DataFrame spawned contains the following features spawned for each column in the input DataFrame:

Exponentially Weighted Moving Average of various half lives: Half Life: 1 day Half Life: 1 week Half Life: 1 month Half Life: 1 quarter Half Life: 6 months Half Life: 1 year

Rolling vol of different window sizes: Window Size: 1 week Window Size: 1 month Window Size: 1 quarter

Parameters **`df`** (*`pd.DataFrame`*) – The dataset of independent variables

Returns The DataFrame containing spawned features

Return type pd.DataFrame

`Forecaster.preprocessing.`**`generate_dataset`**(*universe_dict*, *price_only=True*, *lg_only=False*)

Generates the full dataset.

Parameters

- **`universe_dict`** (*`dict`*) – The dictionary of time series
- **`lag`** (*`int`*) – The lag in days between series
- **`lg_only`** (*`bool`*) – Whether to return a dataset of log returns only
- **`price_only`** (*`bool`*) – Whether to return a dataset of raw prices only

Returns The time series

Return type pd.DataFrame

`Forecaster.preprocessing.`**`generate_lg_return`**(*df_full*, *lag=1*)

Creates the log return series for each column in the DataFrame and returns the full dataset with log returns.

Parameters

- **`df_full`** (*`pd.DataFrame`*) – The time series
- **`lag`** (*`int`*) – The lag between the series (in days)

Returns The DataFrame of time series with log returns

Return type pd.DataFrame

`Forecaster.preprocessing.`**`log_returns`**(*series*, *lag=1*)

Calculates the log returns between adjacent close prices. A constant lag is used across the whole series. E.g a lag of one means a day to day log return.

Parameters

- **`series`** (*`np.array`*) – Prices to calculate the log returns on
- **`lag`** (*`int`*) – The lag between the series (in days)

Returns The series of log returns

Return type np.array

`Forecaster.preprocessing.`**`price_rename`**(*universe_dict*)

Renaming the column of the DataFrame values to price. This is actually the market closing price of the time series.

Parameters **`universe_dict`** (*`dict`*) – The dictionary of time series

Returns The dictionary of renamed time series

Return type dict

`Forecaster.preprocessing.`**`slice_series`**(*data_X*, *data_y*, *series_len*, *dataset_pct=1.0*)

Slices the train and target dataset time series.

Turns each time series into a series of time series, with each series displaced by one step forward to the previous series. And for each of these windows there is an accompanying target value

The effect of this is to create an array of time series (which is the depth equal to the amount of instruments in the dataset) with each entry in this array having a target series in the data_y array

The resulting data_X array shape: [amount of rolling windows, length of each series, number of instruments]

The resulting data_y array shape: [amount of rolling windows, number of instruments]

Parameters

- **`data_X`** (*`np.array`*) – The dataset of time series
- **`data_y`** (*`np.array`*) – The target dataset of time series
- **`series_len`** (*`int`*) – The length of each time series window
- **`dataset_pct`** (*`float`*) – The percentage of the full dataset to include

Returns

Return type

`Forecaster.preprocessing.`**`truncate_window_length`**(*universe_dict*)

Chopping the length of all of the DataFrames to ensure that they are all between the same dates.

Parameters **`universe_dict`** (*`dict`*) – The dictionary of time series

Returns the dictionary of truncated time series

Return type dict

`Forecaster.preprocessing.`**`universe_select`**(*path*, *commodity_name*)

Selects the financial time series relevant for the commodities selected.

Parameters

- **`path`** (*`string`*) – path to the folder containing csvs
- **`commodity_name`** (*`string`*) – the name of the metal/s being inspected

Returns The time series relevant to the commodities

Return type dict

CHAPTER

TWO

DEEPLEARNING MODULE

class Forecaster.deeplearning.**DeepLearning**(*model*, *data_X*, *data_y*, *n_epochs*, *optimiser*, *batch_size*, *loss_function=MSELoss()*, *device='cpu'*, *seed=42*, *debug=True*, *disp_freq=20*, *fig_disp_freq=50*, *early_stop=True*, *early_verbose=False*, *patience=50*, *rel_tol=0*, *scaler_data_X=None*, *scaler_data_y=None*)

Class to perform training and validation for a given model

Parameters

- **model** (*nn.module*) – The neural network model
- **data_X** (*np.array*) – The training dataset
- **data_y** (*np.array*) – the target dataset
- **n_epochs** (*int*) – The number of epochs of training
- **optimiser** (*torch.optim*) – The type of optimiser used
- **batch_size** (*int*) – The batch size
- **loss_function** (*torch.nn.modules.loss*) – The loss function used
- **device** (*string*) – The device to run on (Cpu or CUDA)
- **seed** (*int*) – The number that is set for the random seeds
- **debug** (*bool*) – Whether to print some parameters for checking
- **disp_freq** (*int*) – The epoch frequency that training/validation metrics will be printed on
- **fig_disp_freq** (*int*) – The frequency that training/validation prediction figures will be made
- **early_stop** (*bool*) – Whether early stopping is utilized
- **early_verbose** (*bool*) – Whether to print out the early stopping counter
- **patience** (*stopping int*) – The amount of epochs without improvement before
- **rel_tol** – The relative improvement percentage that must be achieved float
- **scaler_data_X** (*sklearn.preprocessing.data.MinMaxScaler*) – The data X scaler object for inverse scaling
- **scaler_data_y** (*sklearn.preprocessing.data.MinMaxScaler*) – The dataX y scaler object for inverse scaling

`create_data_loaders`()
Forms iterators to pipeline in the data/labels

`evaluate`(*model*, *test_loader*)
Evaluates the performance of the network on given data for a given model.

A lot of overlap of code with validation. Only kept separate due to the inspection of attributes being made easier when running simulations if kept separate.

Parameters

- **`model`** (*`nn.module`*) – The model to evaluate
- **`test_loader`** (*`torch.utils.data.dataloader.DataLoader`*) – The iterator that feeds in the data of choice

Returns The error metric for that dataset

Return type float

`live_pred_plot`()
Plots the training predictions, validation predictions and the training/validation losses as they are predicted.

`size_check`()
Checks the size of the datasets

`train`(*train_loader*)
Performs a single training epoch and returns the loss metric for the training dataset.

Parameters **`train_loader`** (*`torch.utils.data.dataloader.DataLoader`*) – The iterator that feeds in the training data

Returns The error metric for that epoch

Return type float

`train_val_test`()
Splits the DataFrames in to a training, validation and test set and creates torch tensors from the underlying numpy arrays

`training_wrapper`()
The wrapper that performs the training and validation

`validate`(*val_loader*)
Evaluates the performance of the network on unseen validation data.

Parameters **`val_loader`** (*`torch.utils.data.dataloader.DataLoader`*) – the iterator that feeds in the validation data

Returns the error metric for that epoch

Return type float

class `Forecaster.deeplearning.`**`EarlyStopping`**(*patience*, *rel_tol*, *verbose=False*)
Used to facilitate early stopping during the training of neural networks.

When called if the validation accuracy has not relative improved below a relative tolerance set by the user the a counter is incremented. If the counter passes a set value then the stop attribute is set to true. This should be used as a break condition in the training loop.

If rel_tol is set to 0 then the metric just needs to improve from it's existing value

Parameters

- **`patience`** (*`int`*) – The amount of epochs without improvement before stopping

- **rel_tol** (*float*) – The relative improvement % that must be achieved
- **verbose** (*bool*) – Whether to print the count number
- **best** (*float*) – The best score achieved so far
- **counter** (*int*) – The amount of epochs without improvement so far
- **stop** (*bool*) – Whether stopping criteria is achieved

`Forecaster.deeplearning.`**`full_save`**(*model*, *model_name*, *optimiser*, *num_epoch*, *learning_rate*, *momentum*, *weight_decay*, *use_lg_returns*, *PCA_used*, *data_X*, *train_loss*, *val_loss*, *test_loss*, *train_time*, *hidden_dim*, *mse*, *mae*, *mde*, *path='Models/CSVs/'*)

Saves the models run details and hyper-parameters to a csv file :param model: The model run :type model: nn.module

Parameters

- **model_name** (*strin*) – The name the model is saved under
- **optimiser** (*torch.optim*) – The optimiser type used
- **num_epoch** (*int*) – The number of epochs run for
- **learning_rate** (*float*) – The learning rate learning hyper-parameter
- **momentum** (*float*) – The momentum learning hyper-parameter
- **weight_decay** (*float*) – The weight decay learning hyper-parameter
- **use_lg_returns** (*bool*) – Whether log returns was used
- **PCA_used** (*bool*) – Whether PCA was used
- **data_X** (*np.array*) – The training dataset (used to save the shape)
- **train_loss** (*float*) – The loss on the training dataset
- **val_loss** (*float*) – The loss on the validation dataset
- **test_loss** (*float*) – The loss on the test dataset
- **train_time** (*float*) – The amount of time to train
- **hidden_dim** (*int*) – The number of neurons in the hidden layers
- **mse** (*floot*) – The mean squared error metric
- **mae** (*floot*) – The mean absolute error metric
- **mde** (*floot*) – The mean direction error metric
- **path** (*string*) – The directory path to save in

`Forecaster.deeplearning.`**`model_load`**(*model_name*, *path='Results/Pths/'*)

Loading function for the models.

Parameters

- **model_name** (*string*) – The model name to load
- **path** (*string*) – The directory path to load the model from

`Forecaster.deeplearning.`**`model_save`**(*model*, *name*, *path='Results/Pths/'*)

Saving function for the model.

Parameters

- **model** (*torch.nn*) – The model to save

- **name** (*string*) – The name to save the model under
- **path** (*string*) – The directory path to save the model in

`Forecaster.deeplearning.`**`param_strip`**(*param*)

Strips the key text info out of certain parameters. Used to save the text info of which models/optimiser objects are used

Parameters **param** (*object*) – The parameter object to find the name of

`Forecaster.deeplearning.`**`set_seed`**(*seed*)

Sets the random seeds to ensure deterministic behaviour.

Parameters **seed** (*int*) – The number that is set for the random seeds

Returns Confirmation that seeds have been set

Return type bool

CHAPTER

THREE

EVALUATION AND INSPECTION MODULE

`Forecaster.eval_inspect.`**`check_day_frequency`**`(`*df*, *col_name='ds'*`)`

Creates a bar chart showing the frequency of the days of the week.

Used to check that only business days are included in the dataset, and that there is a roughly equal distribution of entries across the week.

Parameters

- **`df`** (*`pd.DataFrame`*) – A DataFrame containing the time series to check
- **`col_name`** (*`string`*) – The name of the column of interest

`Forecaster.eval_inspect.`**`check_length`**`(`*universe_dict*`)`

Checks the name of all the DataFrames in the dictionary of time series.

Parameters **`universe_dict`** (*`dict`*) – The dictionary of time series

`Forecaster.eval_inspect.`**`df_std`**`(`*df*, *col_name*`)`

Calculates standard deviation of a DataFrames column.

Parameters

- **`df`** (*`pd.DataFrame`*) – A DataFrame of time series
- **`col_name`** (*`string`*) – The column of interest

Returns The standard deviation of the series

Return type float

`Forecaster.eval_inspect.`**`evaluate`**`(`*y_true*, *y_pred*, *log_ret=False*`)`

Calculates the error metrics for between two arrays.

The error metrics calculated are: Means Squared Error Mean Absolute Error Mean Directional Accuracy

For a log returns series the definition of mean directional accuracy changes. This is as for a log return series it is the signum values of the series that details which direction the series has moved. This is as a log return series is the first difference of the original series. For raw price The signal needs to be differenced before the signum function is applied.

Parameters

- **`y_true`** (*`np.array`*) – The observed values
- **`y_pred`** (*`np.array`*) – The predicted values
- **`log_ret`** (*`bool`*) – Whether the series compared are log returns

Return error_metrics The error metrics of the series

Return type List

`Forecaster.eval_inspect.`**`inverse_log_returns`**(*original_prices*, *log_returns*, *lag=5*, *offset=0*)

Takes a DataFrame of predicted log returns and original prices and returns an array of predicted absolute prices

The offset parameter moves the series forwards or backwards to align the series with the DataFrame it might be appended to.

Parameters

- **`original_prices`** (*`pd.DataFrame`*) – A DataFrame of absolute prices
- **`log_returns`** (*`pd.DataFrame`*) – A DataFrame of log returns
- **`lag`** (*`int`*) – The lag in days between series
- **`offset`** (*`int`*) – Amount to offset the series forwards of backwards

Returns The raw prices given by the log returns

Return type pd.Series

`Forecaster.eval_inspect.`**`mean_absolute_percentage_error`**(*y_true*, *y_pred*)

Calculates the mean absolute percentage error between two arrays.

Parameters

- **`y_true`** (*`np.array`*) – The observed values
- **`y_pred`** (*`np.array`*) – The predicted values

Returns The mean absolute percentage error of the series

Return type float

`Forecaster.eval_inspect.`**`mean_directional_accuracy`**(*y_true*, *y_pred*)

Calculated the mean directional accuracy error metric between two series.

Parameters

- **`y_true`** (*`np.array`*) – The observed values
- **`y_pred`** (*`np.array`*) – The predicted values

Returns The mean directional accuracy of the series

Return type float

`Forecaster.eval_inspect.`**`mean_directional_accuracy_log_ret`**(*y_true*, *y_pred*)

Calculates the mean directional accuracy error metric between two series of log returns.

Parameters

- **`y_true`** (*`np.array`*) – The observed values
- **`y_pred`** (*`np.array`*) – The predicted values

Returns The mean directional accuracy of the series

Return type float

`Forecaster.eval_inspect.`**`visualise_df`**(*df*)

Visualises each time series in a DataFrame.

Parameters **`df`** (*`pd.DataFrame`*) – The DataFrame of time series to visualise

CHAPTER

FOUR

MODELS MODULE

class Forecaster.models.**LSTM**(*num_features*, *hidden_dim*, *dense_hidden*, *output_dim*, *batch_size*, *series_length*, *device*, *dropout=0.1*, *num_layers=2*)

A Long Short Term Memory network model with an additional dense layer

Parameters

- **num_features** (*int*) – The number of features in the dataset
- **hidden_dim** (*int*) – The number of neurons in the LSTMs hidden layer/s
- **dense_hidden** (*int*) – The number of neurons in the dense layers
- **output_dim** (*int*) – The number of neurons in the output layer
- **batch_size** (*int*) – The number of items in each batch
- **series_length** (*Int*) – The length of the time series
- **device** (*string*) – The device to run on (Cpu or CUDA)
- **dropout** (*float*) – The probability of dropout
- **num_layers** (*int*) – The number of stacked LSTM layers

forward(*x*)

Forward pass through the neural network

Parameters x (*torch.Tensor*) – The input into the network

init_hidden(*batch_size*)

Initialised the hidden state to be zeros. This clears the hidden state between batches. If you are running a stateful LSTM then this needs to be changed.

To change to a stateful LSTM requires not detaching the backprop and storing the computational graph. This strongly increases runtime and shouldn't make a big difference. Hence a stateful LSTM was not used.

Parameters batch_size (*string*) – The batch size to be zeroed

4.1 Indices and tables

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX