Authors -
Noah Kruss
River Veek
Group - Keyboard Warriors
Last Modified - 2/10/21

This file will provide documentation for every function in the preprocessing, tree, modeling, visualization, and file I/O module. The parameters, returns and docstrings for each function will be included as well.

Programmer Documentation

Preprocessing Functions:

`denoise(ts, increment=10)`

Takes time series and an increment value and returns a new time series. Denoises data by applying a rolling mean whose size is determined by the increment.

Parameters:

`ts :` A Pandas DataFrame.

`increment :` An int.

Returns:

A Pandas DataFrame.

`impute_missing_data(ts)`

Takes a time series and returns a new time series.

Computes missing data point by taking mean of
adjacent points in time. Only one data point is
assumed to be missing. Missing point assumed to
be equal to 'NaN.'

If first data point is missing, the mean of the following two points
will be imputed. If the last data point is missing, the mean of the
previous two data points will be imputed. Otherwise, the mean of the previous
and the following point will be imputed.

Parameters:

ts : A Pandas DataFrame.

Returns:

A Pandas DataFrame.

---

impute_outliers(ts)

Takes a time series and returns a time series.

Identifies any outliers in the given time series and imputes new values in their place. Calculations for the new values will follow a similar process to impute_missing_data(); the new values will be equal to the mean of adjacent values.

If first data point is an outlier, the mean of the following two points will be imputed. If the last data point is an outlier, the mean of the previous two data points will be imputed. Otherwise, the mean of the previous and the following point will be imputed in place of the outlier.

Multiple outliers can exist in the given time series.

Parameters:

ts : A Pandas DataFrame.

Returns:

A Pandas DataFrame.

---

longest_continuous_run(ts)

Takes a time series and returns a time series.

Computes the longest continuous run and returns that subset run as a new time series. Here, the longest continuous run is the longest continuous time series subset without any "NaN" values.

This function allows for time series inputs with one or more (>= 1) missing data points.

If all values are "NaN," an empty DataFrame will be returned.

Parameters:

ts : A Pandas DataFrame.

Returns:

A Pandas DataFrame.

`difference(ts)`

Takes in a time series and returns a time series.

Creates a new time series with each value being the magnitude
of the difference between each consecutive element of the provided
time series.

Parameters:

`ts :` A Pandas DataFrame.

Returns:

A Pandas DataFrame.

`clip(ts, starting_date, final_date)`

Takes a time series, the starting date to clip the time series

Takes a time series, the starting date to clip the time series
by, and the ending date to clip the time series by and returns
a modified time series. starting_date and final_date must be of
type float; these two parameters will be assumed to be valid items
of the time series ts.

If starting_date greater than (>) final_date, then time series ts is
clipped from starting_date to the end of ts.

If starting_date or final_date are invalid (not valid keys of ts),
an empty time series will be returned.

Returns a new, clipped time series (does not alter original).

Parameters:

`ts :` A Pandas DataFrame.

`starting_date :` A float.

`final_date :` A float.

Returns:

A Pandas DataFrame

`assign_time(ts, start, increment)`

Takes a time series without timestamps, the starting time (start),

and the amount to be incremented by (increment) and returns an updated time series with the first column being named 'Times.'

Assign time stamps to data points in a given time series. Assumed that start and increment are both integers. It is also assumed that no column holding times exist. start and increment can be negative or positive values.

Parameters:

`ts :` A Pandas DataFrame.

`start :` A float.

`increment :` A float.

Returns:

A Pandas DataFrame

`scaling(ts)`

Takes in a time series and returns a time series.

Creates a new time series in which the magnitudes of each time series value is compressed into the range of [0, 1].

Parameters:

`ts :` A Pandas DataFrame.

Returns:

A Pandas DataFrame.

`standardize(ts)`

Takes in a time series and returns a time series.

Creates a new time series that translates the values of the existing time series to have a mean of 0 and a variance of 1.

Parameters:

`ts :` A Pandas DataFrame.

Returns:

A Pandas DataFrame.

`design_matrix(ts, data_start, data_end)`

Takes a time series, float representing the size of input array, float representing the size of the output array, and returns a

tuple that holds the two matrices (input matrix and output matrix,
both as numpy arrays) in that particular order.

Example of return:
(input array, output array)

data_start represents the size of the input matrix. data_end represents
the size of the output matrix. Both parameters are expected to be floats
but will immediately be converted into ints.

Parameters:

`ts :` A Pandas DataFrame.

`data_start :` A float.

`data_end :` A float.

Returns:

A Pandas DataFrame.

`logarithm(ts)`

Converts ts values to their log10 value, returns a time series

Parameters:

`ts :` A Pandas DataFrame.

Returns:

A Pandas DataFrame.

`cubic_root(ts)`

Converts ts values to their cubic root value, returns a time series.

Parameters:

`ts :` A Pandas DataFrame.

Returns:

A Pandas DataFrame.

`split_data(ts, perc_training, perc_valid, perc_test)`

Splits ts dataframe into percentage sized arrays based on input percentages: perc_traning, perc_valid, and perc_test.

Parameters:

`ts :` A Pandas DataFrame.

perc_training : A float.

perc_valid : A float.

perc_test : A float.

Returns:

Three time series in the form of Pandas DataFrames.

---

ts2db(input_file, perc_train, perc_val, perc_test, input_index, output_index, output_file)

Takes in an input data file to read in the time series, as well as how much the user wants the data to be split into three categories: training, validation, and testing. Then, it creates three separate time series that it turns into machine learning model friendly databases with the input and output sizes provided, and returns those.

Parameters:

input_file : A string.

perc_train : A float.

perc_val : A float.

perc_test : A float.

input_index : A float.

output_index : A float.

output_file : A string.

Returns:

A tuple of three Pandas DataFrames.

---

db2ts(db)

Takes in a numpy matrix (database) containing a set of values and converts it to a time series.

Parameters:

db : A numpy array.

Returns:

A Pandas DataFrame.

TS_Tree class:

`__init__(self)`

Initialize a Tree with a read_from_file operation stored in the root node. Tree stores list of nodes that it possesses in (self.nodes).

Parameters:

`self`.

Returns:

TS_Tree.

`print_tree(self)`

Function for displaying a string representation of the tree in the User terminal

Calls:

`RenderTree()`.

Parameters:

`self`.

Returns:

None.

```
add_node(self,
         operation: str,
         parent_index: int,
         data_start = None,
         data_end = None,
         increment = None,
         perc_training = None,
         perc_valid = None,
         perc_test = None,
         input_filename = None,
```

output_filename = None,
m_i = None,
t_i = None,
m_0 = None,
t_0 = None,
layers = None)

Function for adding a new node to the tree. Takes an operation string and the index of the parent (parent_index). New node gets added as a child of the parent and added into the list of tree nodes.

Calls:

tree.py : Operation_node.__init__().
tree.py : Operation_node.add_child().

Called By:

tree.py : add_subtree().

Parameters:

self
parent_index : Int value of the index of the node within the TS_Tree that the user wants to be the parent of this new node.
data_start : Float input that can be passed to functions (clip, assign_time, design_matrix, ts2db, mlp_model).
data_end : Float input that can be passed to functions (clip, design_matrix).
increment : Float input that can be passed to functions (assign_time, mlp_model).
perc_training : Float input that can be passed to functions (split_data, ts2db).
perc_valid : Float input that can be passed to functions (split_data, ts2db).
perc_test : Float input that can be passed to functions (split_data, ts2db).
input_filename : Str input that can be passed to functions (ts2db, mse, mape, smape).
output_filename : Str input that can be passed to functions (ts2db).
m_i : Float input that can be passed to (design_matrix_2).
t_i : Float input that can be passed to (design_matrix_2).
m_0 : Float input that can be passed to (design_matrix_2).
t_0 : Float input that can be passed to (design_matrix_2).
layers : List input that can be passed to (mlp_model).

Returns:

None.

---

replace_node(self,
new_operation: str,
node_index: int,
data_start = None,
data_end = None,
increment = None,
perc_training = None,
perc_valid = None,
perc_test = None,
input_filename = None,
output_filename = None,
m_i = None,
t_i = None,
m_0 = None,
t_0 = None,
layers = None)

Function to replace the operation with (new_operation) of the (node_index) node within the tree

Calls:

tree.py : Operation_node.change_operation().

Parameters:

self

node_index : Int value of the index of the node within the TS_Tree that the User wants to replace.

Data_start : Float input that can be passed to functions (clip, assign_time, design_matrix, ts2db, mlp_model).

data_end : Float input that can be passed to functions (clip, design_matrix).

increment : Float input that can be passed to functions (assign_time, mlp_model).

perc_training : Float input that can be passed to functions (split_data, ts2db).

perc_valid : Float input that can be passed to functions (split_data, ts2db).

perc_test : Float input that can be passed to functions (split_data, ts2db).

input_filename : Str input that can be passed to functions (ts2db, mse, mape, smape).

output_filename : Str input that can be passed to functions (ts2db).

m_i : Float input that can be passed to (design_matrix_2).

t_i : Float input that can be passed to (design_matrix_2).

m_0 : Float input that can be passed to (design_matrix_2).

t_0 : Float input that can be passed to (design_matrix_2).

layers : List input that can be passed to (mlp_model).

Returns:

None.

get_path(self, node_index: int)

Function to get a list of nodes of the pipeline leading to the node at (node_index) in the tree.

Calls:

tree.py : Operation_node.change_operation(new_operation).

Called By:

tree.py : TS_Tree.execute_path().

tree.py : copy_path().

Parameters:

node_index : An int.

Returns:

A list.

---

execute_path(self, data_input, node_index: int)

Function for running the pipeline from the root to a node with (node_index) within a tree on the given data_input (time series or file_name of time series).

Calls:

tree.py : TS_Tree.get_peth().

Called By:

tree.py : TS_Tree.execute_tree().

Parameters:

data_input : String representing name of a file.

node_index : An int.

Returns:

A time series.

---

execute_tree(self, data_input)

Function for running every possible pipeline from the root to a leaf within a tree on the given an data_input (time series or file_name of time series).

Calls:

tree.py : TS_Tree.execute_path().

Parameters:

data_input : String representing name of a file.

Returns:

Dictionary of results of pipelines.

Functions that Act upon a TS_Tree

---

copy_subtree(main_tree: TS_Tree, node_index: int)

Takes a Tree and returns a copy of the subtree starting at node (node_index) as a new TS_Tree object.

Calls:

tree.py : save_tree().

Parameters:

main_tree : A TS_Tree.

node_index : An int.

Returns:

Tree

---

add_subtree(tree: TS_Tree, node_index: int, sub_tree: TS_Tree)

Function to add (subtree) TS_Tree to be a child of the target node given by (node_index) of (tree).

Parameters:

tree : A TS_Tree.

node_index : An int.

sub_tree : A TS_Tree.

Returns:

None

---

copy_path(main_tree: TS_Tree, node_index: int)

Takes a Tree and returns a copy of the path starting at the root and going to (node_index) as a new TS_Tree object.

Calls:

tree.py : TS_Tree.get_path().

Parameters:

main_tree : A TS_Tree.

node_index : An int.

Returns:

A tree.

---

save_tree(tree: TS_Tree, save_file_name: str)

Function to save target tree (tree) structure into a file named (save_file_name).
The resulting save_file will save each node on a line as follows 'node.name,parent.name.'

Parameters:

Tree : A TS_Tree.

save_file_name : A string.

Returns:

None.

---

load_tree(save_file_name: str)

Function to load tree structure from given file (save_file_name).

Called By:

TS_Tree().

TS_Tree.add_node().

Parameters:

save_file_name : A string.

Return:

A TS_Tree.

Visualization Functions

`plot(ts | list ts, fname=None)`

Plots a time series or a list of time series.

Parameters:

`ts | list ts :` A Pandas DataFrame, or a list of dataframes.

`fname :` Optional file name to save the plot.

Returns:

A Pandas DataFrame, displays plot.

`histogram(ts, fname=None)`

Creates a histogram of a time series, and performs a normal plot alongside it.

Parameters:

`ts :` A Pandas DataFrame.

`fname :` Optional file name to save plot.

Returns:

A Pandas DataFrame, displays histogram.

`box_plot(ts, fname=None)`

Creates a box plot of a time series, and performs a normal plot alongside it.

Parameters:

`ts :` A Pandas DataFrame.

`fname :` Optional file name to save plot.

Returns:

A Pandas DataFrame, displays box plot.

`normality_test(ts)`

Runs a normality test of the data.

Parameters:

`ts :` A Pandas DataFrame, or a list of dataframes.

Returns:

A float.

`mse(y_forecast, y_test: str)`

Return the mean squared error between two datasets.

Parameters:

`y_forecast :` A Pandas DataFrame.

`y_test :` A string filename representing data to be read in.

Returns:

A float.

`mse(y_forecast, y_test: str)`

Return the mean squared error between two datasets.

Parameters:

`y_forecast :` A Pandas DataFrame.

`y_test :` A string filename representing data to be read in.

Returns:

A float.

`mape(y_forecast, y_test: str)`

Returns the mean absolute percentage error.

Parameters:

`y_forecast :` A Pandas DataFrame.

`y_test :` A string filename representing data to be read in.

Returns:

A float.

`smape(y_forecast, y_test: str)`

Returns the symmetric mean absolute percentage error.

Parameters:

`y_forecast :` A Pandas DataFrame.

`y_test :` A string filename representing data to be read in.

Returns:

A float.

File IO Functions

`read_in_file(input_file_name)`

Reads in data from a csv or pickle file.

Note: csv files (to be used in our system) MUST start with 1 header line which describes what each column in the file represents

Parameters:

`input_file_name :` A string containing the file's name.

Returns:

A Pandas DataFrame.

`write_out_file(output_file_name)`

Writes data to a csv file.

Parameters:

`output_file_name :` A string containing the file name.

Returns:

None.

Modeling Functions

`mlp_modeling(train, layers, window_size)`

Creates a multi-layer perceptron model with a provided number
of hidden layers. Trains the model with provided data.

Parameters:

`train :` A tuple of numpy matrices.

`layers :` A tuple or list specifying the how many nodes in each hidden layer.

`window_size :` An integer.

Returns:

A tuple containing a sklearn MLPRegresssor object as well as another tuple containing the bounds to the data.

`mlp_forcast(model_data, x_filename)`

Predicts a future set of values from a given set of values
and a trained model.

Parameters:

`model_data :` A tuple containing a sklearn MLPRegresssor object as well as another tuple containing the bounds to the data.

`x_filename :` A file containing a matrix of values to predict from.

Returns:

A numpy matrix of predictions.
