msdlib

Release 1.1.13

Abdullah Al Masud

Mar 02, 2024

INTRO

msdlib is a collection of frequently used functions and modules. A common data scientist or data analyst has to do a lot of things. He has to work on data processing, data cleaning, data visualization, story-telling, insight generation, trend analysis etc.

Additionally, he aslo needs to take care algorithm development for data modeling. Now-a-days Machine Learning and Deep Learning are very efficient techniques to model data and provide solutions like binary and multi-class classification, regression etc. Additionally, some programmers need to build much more complex models too.

The idea of this library comes to light from this concept to make every general tasks easier. To prepare business level plots, tables and heatmaps very easily, to build Deep LEarning models very fast and train them with minimal coding.

msdlib provides supports to different types of things altogether. Some of the core features are-

- Generalized time series data visualization library for multivariate time series data
- Quick heatmap and table generation functions
- dataset splitting method (random, k-fold cross-validation and sequece splitting for LSTM)
- easy hyper-parameter optimization module
- Scikit-like wrapper for Pytorch models. (fit(), predict() and evaluate() for regression, binary classification and multi-label classification)
- Hyper-parameter optimization of Pytorch Deep Learning model.
- regression and classification result calculation functionality with many metrics (precision, recall, f1 score, accuracy, specificity, r-square, rmse etc.)
- Statistical feature importance functionality
- FIR filtering and filter and data visualization functionality with minimal coding.

CHAPTER

ONE

MSDLIB DOCUMENTATION

Get started from this page

1.1 msdlib

Author : Abdullah Al Masud

email : abdullahalmasud.buet@gmail.com

1.1.1 Introduction

The main purpose of this library is to make data science works easier and simpler with less amount of coding, providing helper functions for plotting, ML training, evaluation, result summarization etc. The purpose is to focus more on making common tasks easier so that a beginner to mid level developer is able to do his/her jobs easily and can get started career with enough pace.

1.1.2 Dependencies

- Numpy
- Pandas
- Matplotlib
- Scipy
- Seaborn
- joblib
- Pytorch
- tensorboard

All of these packages except Pytorch will be installed automatically during msdlib installation.

Pytorch should be installed by following installation procedure suggested here.

1.1.3 Installation

```
pip install msdlib
```

or if you have –user related issues during installation, please use

```
pip install --user msdlib
```

1.1.4 License

MIT open source License has been issued for this library.

1.1.5 Examples

You can find easy examples on how to use the functions and classes from this library here. Necessary data is also provided in this directory.

1.1.6 Documentation

Complete documentation of classes and functions can be found here https://msdlib.readthedocs.io/.

1.1.7 Call for contributions

We seek active participation of enthusiastic developers from around the world to enrich this library more, adding more functionalities from different aspects, giving more flexibility, completing unfinished functionalities and maintain the library in regular manner. We would be grateful for your invaluable suggestions and participations.

1.1.8 Overview

The whole library can be divided into 4 main portions.

1. Machine learning tools
2. Visualization tools
3. Data processing tools
4. Fintech and Miscellaneous

Some of the frequently used programs are shown bellow.

1.1.9 1. Machine Learning Tools

mlutils:

This module provides functionalities for easier implementation of Pytorch Deep Learning models. It offers several facilities such as-

- Scikit-like easy implementation of Pytorch models using fit, predict and evaluate methods
- Constructing Deep Learning models in a few lines of code
- Producing automated results with beautiful tables having precision, recall, f1_score, accuracy and specificity in classification problems
- Producing automated graphs of true-vs-prediction and result preparation for regression model

Examples are available for regression, binary and multi-class classification models here.

paramOptimizer:

This is a class which can conduct easy Hyper-parameter optimization process. Currently it enables us to apply grid search and random search for any model/function/mathematical entity

SplitDataset:

This is one of the most useful classes in this library. It enables us to split data set into train, validation and test sets. We have three options here to split data set-

- random_split
- cross_validation_split
- sequence_split (specially necessary for RNN/LSTM)

Examples are available here.

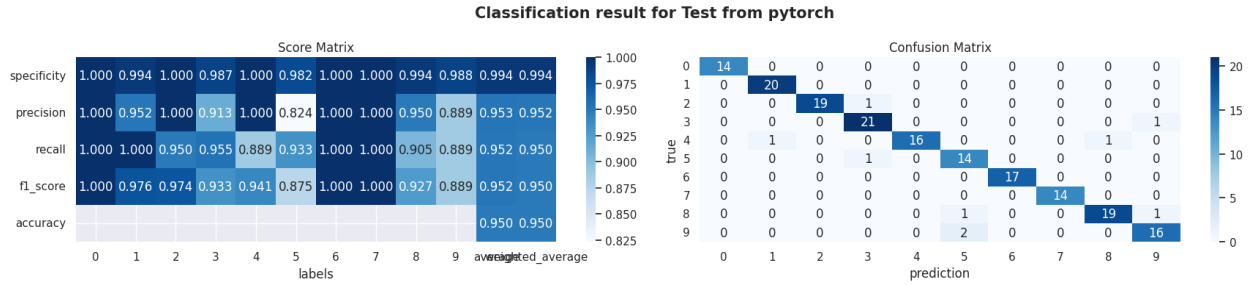
one_hot_encoding:

This function converts classification labels in one hot encoded format

feature_evaluator:

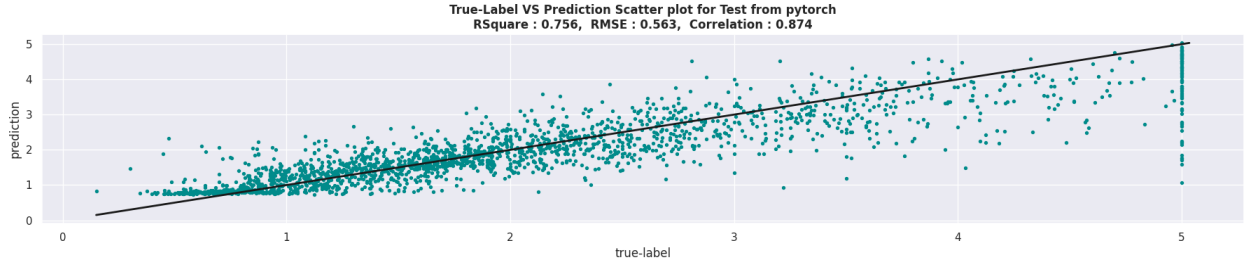
This function is one of the most useful tools. It can calculate feature importance from statistical point of view. It can show the results using bar plot and can handle classification and regression both kind of labels.

class_result:



This function calculates classification model evaluation parameters like precision, recall, accuracy, f1 score, specificity etc. and also able to show confusion matrix as a pandas dataframe. Example is available here.

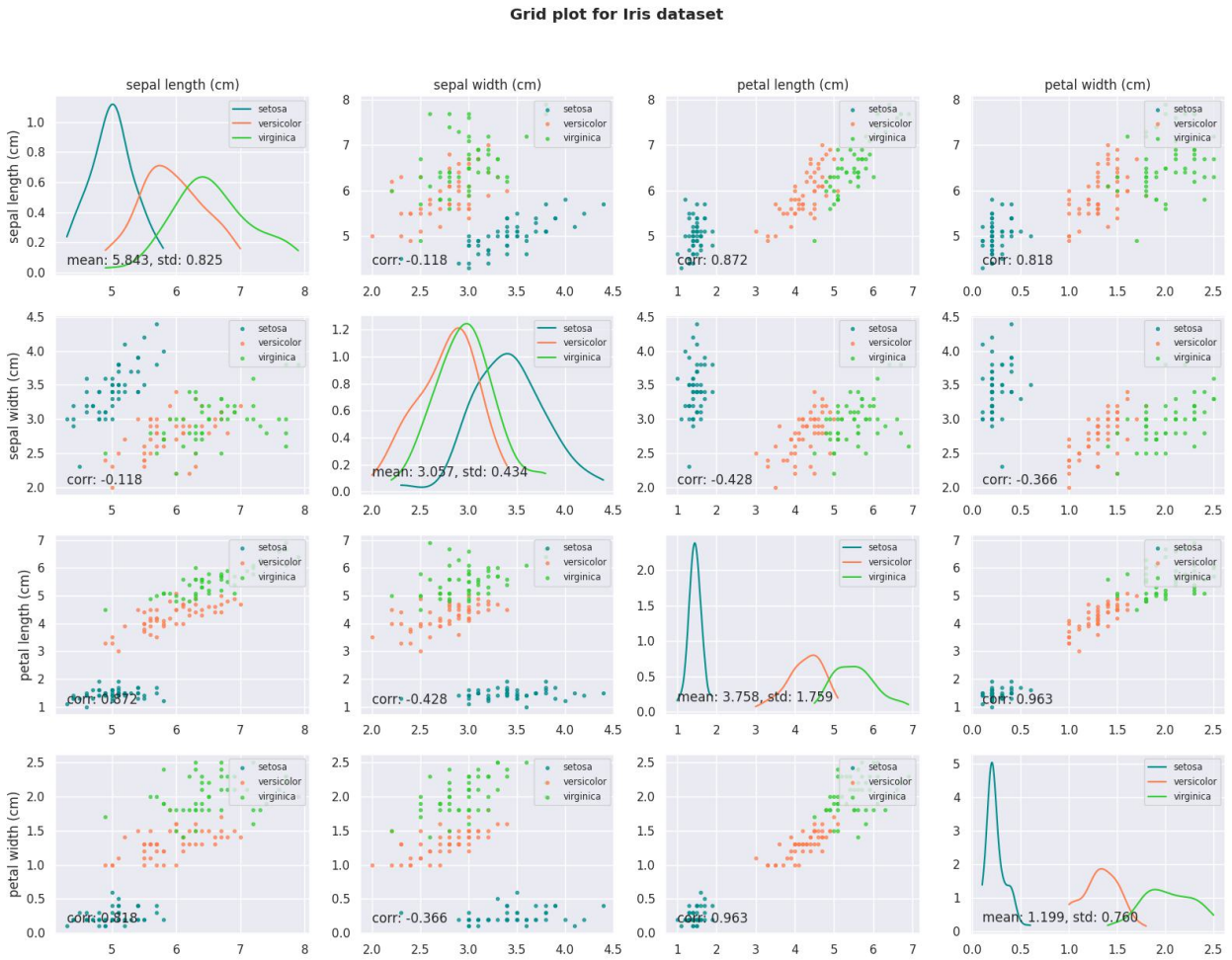
rsquare_rmse:



This function calculates r square value and root mean square error.

1.1.10 2. Visualization Tools

data_gridplot:



Its a function for scatter plots between every pair of features along with distributions (similar to matrix_plot in pandas). But it enables you to save the image, change figure_size, titles etc and also has one special feature for clusters in the data if any. Example is available here.

plot_time_series:

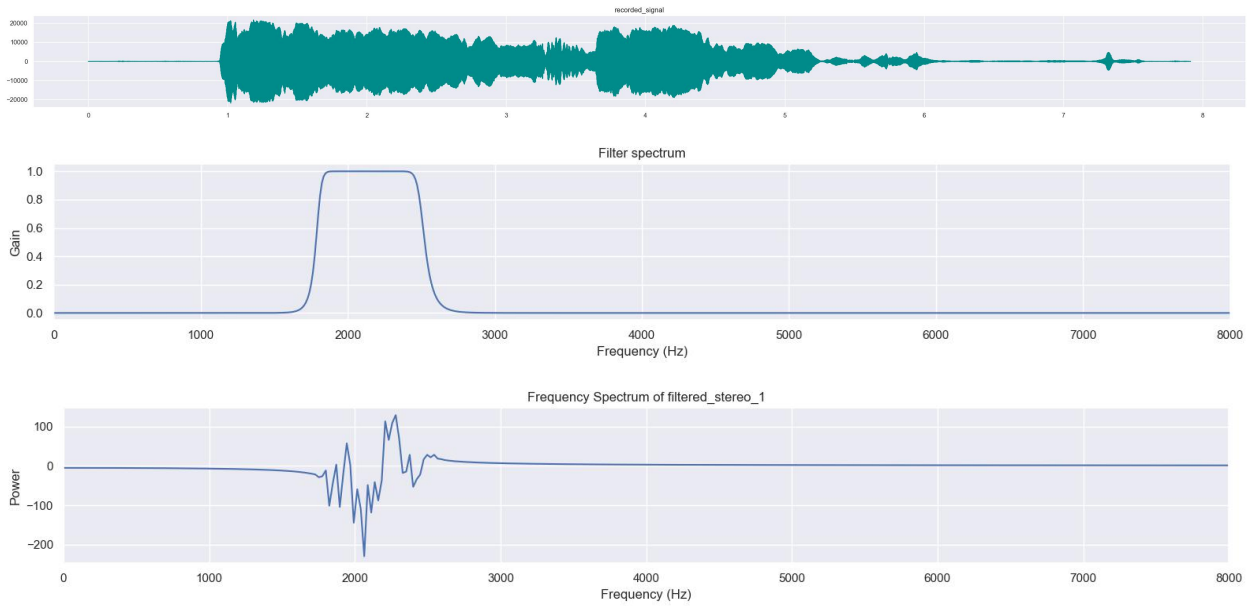
This is one of the the most useful functions in this library. It helps to plot time series data with a lot of flexibility. It helps to plot time series data with a lot of flexibility. Please check out the example scripts for illustrations and guidance to use it. Example is available here.

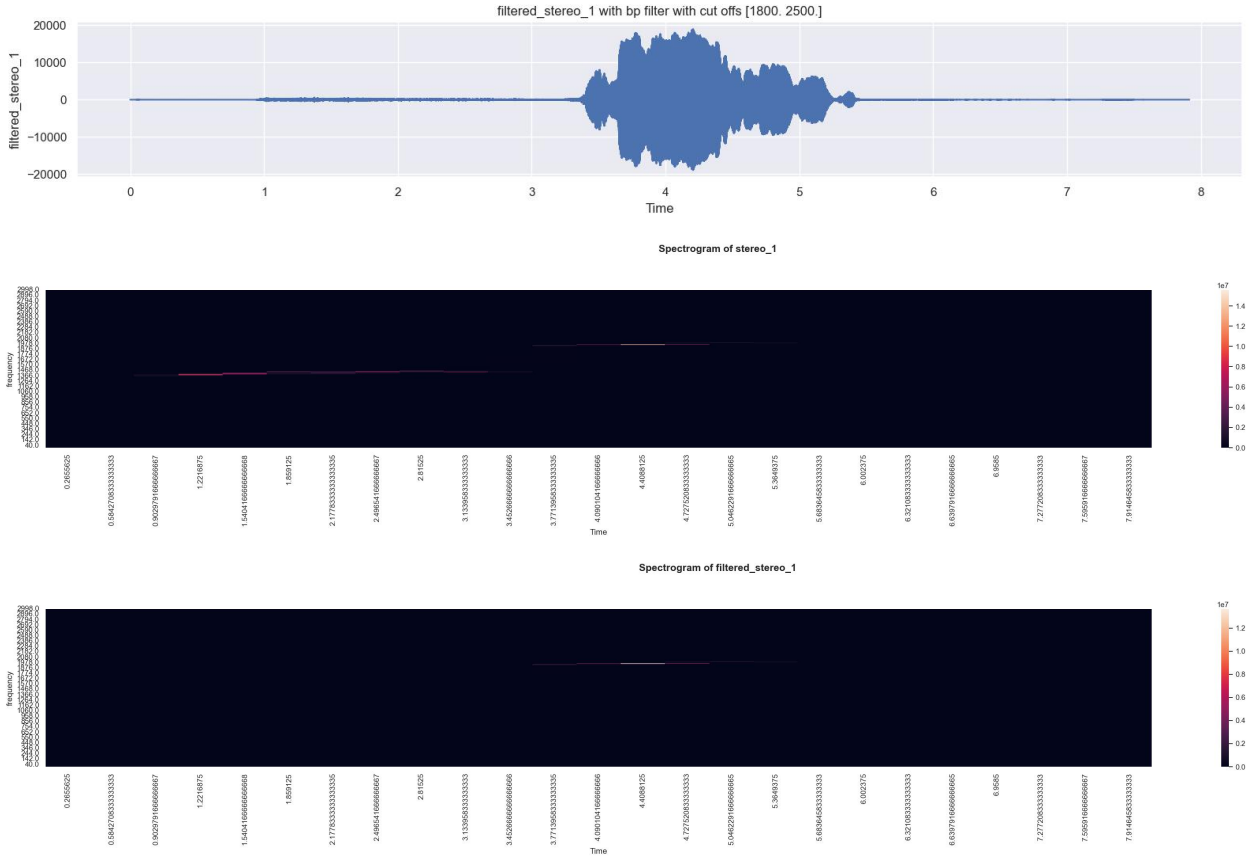
plot_heatmap:

Flexible heatmap plotter function with options to remove symmetrical triangular side and several other options.

1.1.11 3. Data Processing Tools

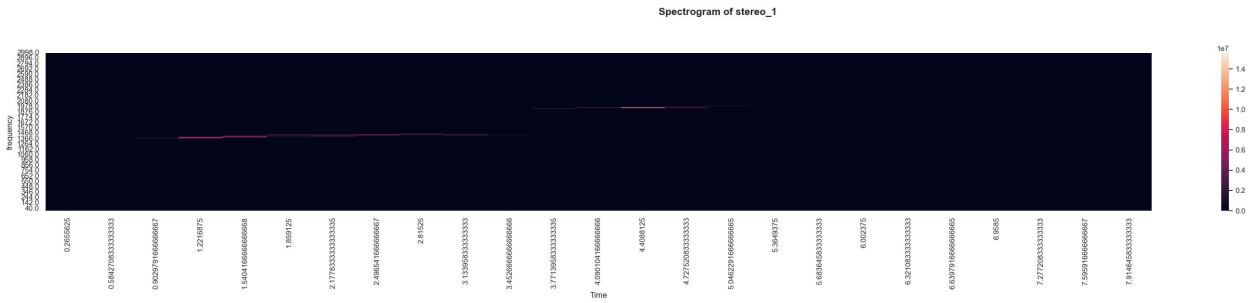
Filters:





This is a class defined for applying low pass, high pass, band pass and band stop filters. It also enables us to visualize frequency domain of the signal, designed filter and also let us visualize the filtered signal if we apply a filter on the signal. Example is available here.

get_spectrogram:



This is a function that allows us to calculate spectrogram of any time series signal and also plots heatmap for that spectrogram with proper frequency bins and time axis. Example is available here.

1.1.12 4. Fintech & Miscellaneous

msdbacktest (under development):

This module intends to provide helper functionalities for trading automation, strategy implementation, back-testing, evaluating strategy by different popular ratios like maximum drawdown, calmar ratio, sharpe ratio etc. Currently only a few functionalities are available and is still under development.

ProgressBar:

This is a custom progress bar which shows loop progress with a visual bar along with other information like elapsed and remaining time, loop count, total count, percentage of completion etc. (You should only use it if you dont print anything inside your loop) Example can be found here.

1.1.13 Reference/citation

```
@manual{msdlib,
title="{msdlib}: A package for easier data science practices",
author="{Abdullah Al Masud and {msdlib Developers}}",
year=2020,
month=Jan,
url="{https://github.com/abdullah-al-masud/msdlib}"
}
```

1.2 msd

This module provides many useful functions and classes to ease works for a data scientist/ML engineer. The key elements are like below.

-Data visualization

- Time series data visualizationn
- Drawing heatmap, table etc.
- Drawing FIR filter response and filter effects (before and after applying filter) etc.
- GRID plot for understanding relationship among features
- Visualizing Spectrogram

-Data processing

- FIR filtering for time series data
- Computing Spectogram
- Data standardization and normalization
- One hot encoding
- Moving slope and mode for time series

-Tools for ML

- Feature ranking using simple statistics

- Prediction evaluation for both classification (2 and more than 2 classes) and regression
- Hyper-parameter optimization
- **splitting data set into train, validation and test usingn**
  - Random splitting method
  - K-fold cross validation splitting
  - Sequence splitting (necessary for LSTM modeling)

Author : Abdullah Al Masud

email : abdullahalmasud.buet@gmail.com

class msdlib.msd.**Filters**(*T*, *N=1000*, *savepath=None*, *save=False*, *show=True*)

Bases: object

This class is used to apply FIR filters as high-pass, low-pass, band-pass and band-stop filters. It also shows the proper graphical representation of the filter response, signal before filtering and after filtering and filter spectram. The core purpose of this class is to make the filtering process super easy and check filter response comfortably.

Inputs:

T
float, indicating the sampling period of the signal, must be in seconds (doesnt have any default values)

n
int, indicating number of fft frequency bins, default is 1000

savepath
str, path to the directory to store the plot, default is None (doesnt save plots)

show
bool, whether to show the plot or not, default is True (Shows figures)

save
bool, whether to store the plot or not, default is False (doesnt save figures)

apply(*sr*, *filt_type*, *f_cut*, *order=10*, *response=False*, *plot=False*, *f_lim=[]*, *savepath=None*, *show=None*, *save=None*)

The purpose of this function is to apply an FIR filter on a time series signal 'sr' and get the output filtered signal y

Inputs:

sr
numpy ndarray/pandas Series/list, indicating the time series signal you want to apply the filter on

filt_type
str, indicating the type of filter you want to apply on sr.

{'lp', 'low_pass', 'low pass', 'lowpass'} for applying low pass filter

and similar for 'high pass', 'band pass' and 'band stop' filters

f_cut
float/list/numpy 1d array, n indicating cut off frequencies. Please follow the explanations bellow

for lowpass/highpass filters, it must be int/float

for bandpass/bandstop filters, it must be a list or numpy array of length divisible by 2

order
int, filter order, the more the values, the sharp edges at the expense of complexer computation. Default is 10.

response
bool, whether to check the frequency response of the filter or not, Default is False

plot
bool, whether to see the spectrum and time series plot of the filtered signal or not, Default is False

f_lim
list of length 2, frequency limit for the plot. Default is []

savepath
str, path to the directory to store the plot, default is None (follows Filters.savepath attribute)

show
bool, whether to show the plot or not, default is None (follows Filters.show attribute)

save
bool, whether to store the plot or not, default is None (follows Filters.save attribute)

Outputs:

y
pandas Series, filtered output signal

`filter_response`(*sos*, *f_lim=[]*, *savepath=None*, *show=None*, *save=None*)

This function plots the filter spectram in frequency domain.

Inputs:

sos
sos object found from butter function

flim
list/tuple as [lower_cutoff, higher_cutoff], default is []

savepath
str, path to the directory to store the plot, default is None (follows Filters.savepath attribute)

show
bool, whether to show the plot or not, default is None (follows Filters.show attribute)

save
bool, whether to store the plot or not, default is None (follows Filters.save attribute)

Outputs:
It doesnt return anything.

`plot_filter`(*y*, *filt_type*, *f_cut*, *f_lim=[]*, *savepath=None*, *show=None*, *save=None*)

This function plots filter frequency response, time domain signal etc.

Inputs:

y
pandas Series, time series data to filter

filt_type
str, indicating the type of filter you want to apply on y.

{'lp', 'low_pass', 'low pass', 'lowpass'} for applying low pass filter

and similar for 'high pass', 'band pass' and 'band stop' filters

f_cut
float/list/numpy 1d array, n indicating cut off frequencies. Please follow the explanations bellow

for lowpass/highpass filters, it must be int/float

for bandpass/bandstop filters, it must be a list or numpy array of length divisible by 2

f_lim
list of length 2, frequency limit for the plot. Default is []

savepath
str, path to the directory to store the plot, default is None (follows Filters.savepath attribute)

show
bool, whether to show the plot or not, default is None (follows Filters.show attribute)

save
bool, whether to store the plot or not, default is None (follows Filters.save attribute)

Outputs:
No output is returned. The function generates plot.

`raise_cutoff_error`(*msg*)

`raise_filter_error`(*msg*)

`vis_spectrum`(*sr*, *f_lim=[]*, *see_neg=False*, *show=None*, *save=None*, *savepath=None*, *figsize=(30, 3)*)
The purpose of this function is to produce spectrum of time series signal 'sr'.

Inputs:

sr
numpy ndarray or pandas Series, indicating the time series signal you want to check the spectrum for

f_lim
python list of len 2, indicating the limits of visualizing frequency spectrum. Default is []

see_neg
bool, flag indicating whether to check negative side of the spectrum or not. Default is False

show
bool, whether to show the plot or not, default is None (follows Filters.show attribute)

save
bool, whether to store the plot or not, default is None (follows Filters.save attribute)

savepath
str, path to the directory to store the plot, default is None (follows Filters.savepath attribute)

figsize
tuple, size of the figure plotted to show the fft version of the signal. Default is (30, 3)

Outputs:
doesnt return anything as the purpose is to generate plots

class msdlib.msd.**ProgressBar**(*arr*, *desc='progress'*, *barlen=40*, *front_space=20*, *tblink_max=0.3*, *tblink_min=0.18*)

Bases: object

Inputs:

arr
iterable, it is the array you will use to run the loop, it can be range(10) or any numpy array or python list or any other iterator

desc
str, description of the loop, default - 'progress'

barlen
int, length of the progress bar, default is 40

front space
int, allowed space for description, default is 20

tblink_max
float/int indicates maximum interval in seconds between two adjacent blinks, default is .4

tblink_min
float/int indicates minimum interval in seconds between two adjacent blinks, default is .18

Outputs:
there is no output. It creates a progress bar which shows the loop progress.

barprint(*end=''*)

blink_func()

calc_time()

conv_time()

inc()

set_barelap()

class msdlib.msd.**SplitDataset**(*data*, *label*, *index=None*, *test_ratio=0*, *np_seed=1216*, *same_ratio=False*, *make_onehot=False*)

Bases: object

This class contains method to split data set into train, validation and test. It allows both k-fold cross validation and casual random splitting. Additionally it allows sequence splitting, necessary to model time series data using LSTM.

Inputs:

data
pandas DataFrame or Series or numpy ndarray of dimension #samples X other dimensions with rank 2 or higher in total

label
pandas dataframe or series or numpy ndarray with dimension #samples X #classes with rank 2 in total

index
numpy array, we can explicitly insert indices of the data set and labels, default is None

test_ratio
float, ratio of test data, default is 0

np_seed
int, numpy seed used for random shuffle, default is 1216

same_ratio
bool, keep the test and validation ratio same for all classes.

Should be only true if the labels are classification labels, default is False

make_onehot
bool, if True, the labels will be converted into one hot encoded format. Default is False

`check_data_label()`

`cross_validation_split(fold=5, only_index=False)`

This method applies cross validation splitting.

Inputs:

fold
int, number of folds being applied to the data, default is 5

only_index
bool, whether to return only index or data, label, index all. Default is False meaning the function will return data, label and index for all folds.

Outputs:

outdata
dict containing data, label and indices for train, validation and test data sets

`data_label_mismatch(msg)`

`prepare_class_index(same_ratio)`

`random_split(val_ratio=0.15)`

This method splits the data randomly into train, validation and test sets.

Inputs:

val_ratio
float, validation data set ratio, default is .15

Outputs:

outdata
dict containing data, label and indices for train, validation and test data sets

`sequence_split(seq_len, val_ratio=0.15, data_stride=1, label_shift=1, split_method='multiple_train_val', sec=1, dist=0, inference=False)`

This method creates sequences of time series data and then splits those sequence data into train, validation and test sets. We can create mutiple pairs of train and validation data sets if we want. Test data set will be only one set.

Note : As this method creates sequential data, it may also be needed in inference time.

Inputs:

seq_len
int, length of each sequence

val_ratio
float, validation data set ratio, deafult is .15

data_stride
int, indicates the number of shift between two adjacent example data to prepare the data set, default is 1

label_shift
temporal difference between the label and the corresponding data's last time step, default is 1

split_method
{'train_val', 'multiple_train_val'}, default is 'multiple_train_val'

sec
int or pandas dataframe with columns naming 'start' and 'stop', number of sections for multiple_train_val method splitting (basically train_val splitting is multiple_train_val method with sec = 1), deafult is 1

dist
int, number of data to be leftover between two adjacent sec, default is 0

inference
bool, whether the method is called in inference or not. Default is False

Outputs:

outdata
dict containing data, label and indices for train, validation and test data sets

`split_method_error`(*msg*)

msdlib.msd.**class_result**(*y*, *pred*, *out_confus=False*)

This function computes classification result with confusion matrix For classification result, it calculates precision, recall, f1_score, accuracy and specificity for each classes.

Inputs:

y
numpy 1-d array, list or pandas Series, true classification labels, a vector holding class names/indices for each sample

pred
numpy 1-d array, list or pandas Series, predicted values, a vector containing prediction opt for class indices similar to y.

Must be same length as y

out_confus
bool, returns confusion matrix if set True. Default is False

Outputs:

result
DataFrame containing precision, recall and f1 scores for all labels

con_mat
DataFrame containing confusion matrix, depends on out_confus

msdlib.msd.**data_gridplot**(*data*, *idf=[]*, *idf_pref=''*, *idf_suff=''*, *diag='hist'*, *bins=25*, *figsize=(16, 12)*, *alpha=0.7*, *s=None*, *lg_font='x-small'*, *lg_loc=1*, *fig_title=''*, *show_corr=True*, *show_stat=True*, *show=True*, *save=False*, *savepath=None*, *fname=''*, *cmap=None*)

This function creates a grid on nxn subplots showing scatter plot for each columns of 'data', n being the number of columns in 'data'. This function also shows histogram/kde/line plot for each columns along grid diagonal. The additional advantage is that we can separate each subplot data by a classifier item 'idf' which shows the class

names for each row of data. This functionality is specially helpful to understand class distribution of the data. It also shows the correlation values (non-diagonal subplots) and general statistics (mean, median etc in diagonal subplots) for each subplot.

Inputs:

data
pandas dataframe, list or numpy ndarray of rank-2, columns are considered as features

idf
pandas series with length equal to total number of samples in the data, works as clussifier of each sample data,

specially useful in clustering, default is []

idf_pref
str, idf prefix, default is ‘’

idf_suff
str, idf suffix, default is ‘’

diag
{‘hish’, ‘kde’, ‘plot’}, selection of diagonal plot, default is ‘hist’

bins
int, number of bins for histogram plot along diagonal, default is 25

figsize
tuple, size of the whole figure, default is (16, 12)

alpha
float (0~1), transparency parameter for scatter plot, deafult is .7

s
float, point size for scatter plot, deafult is None

lg_font
float or {‘xx-small’, ‘x-small’, ‘small’, ‘medium’, ‘large’, ‘x-large’, ‘xx-large’}, legend font size, default is ‘x-small’

lg_loc
int, location parameter for plot legend, deafult is 1

fig_title
str, titile of the whole figure, default is ‘All Columns Grid Plot’

show_corr
bool, whether to show correlation value for each scatter plot, default is True

show_stat
bool, whether to show mean and std for each columns od data along diagonal plots, default is True

show
bool, whether to show the figure or not, default is True

save
bool, whether to save the graph or not, default is False

savepath
str, path to store the graph, default is None

fname
str, name used to store the graph in savepath, defualt is fig_title

cmap
str, matplotlib color map, default is None ('jet')

Outputs:
This function doesnt return anything.

msdlib.msd.**each_row_max**(*data*)

The purpose of this function is to get the maximum values and corresponding column names for each row of a matrix

Inputs:

data
list of lists/numpy ndarray or pandas dataframe, matrix data from where max values will be calculated

Outputs:
returns same data with two new columns with max values and corresponding column names

msdlib.msd.**feature_evaluator**(*data*, *label_name*, *label_type_num*, *n_bin=40*, *is_all_num=False*, *is_all_cat=False*, *num_cols=[]*, *cat_cols=[]*, *cmap='gist_rainbow'*, *fig_width=30*, *fig_height=5*, *rotation=40*, *show=True*, *save=False*, *savepath=None*, *fname=None*, *fig_title=''*)

This function calculates feature importance by statistical measures. Here its okay to use labels before converting into one hot encoding, its even better not to convert into one hot.

The process is below-

——— feature evaluation criteria ———

for numerical feature evaluation:

group- a variable is divided into groups based on a constant interval of its values

group diversity- std of mean of every group

group uncertainty- mean of std of points for every group

for categorical feature evaluation:

group diversity- mean of std of percentage of categories inside a feature

group uncertainty- mean of std of percentage of groups for each category

group confidence (Importance)- group diversity / group uncertainty

Inputs:

data
must be a pandas dataframe containing feature and label data inside it

(it must contain only features and labels, no unnecessary columns are allowed)

label_name
list; containing names of the columns used as labels

label_type_num
list of bool, containing flag info if the labels are numerical or not (must be corresponding to label_name)

n_bin
int, expressing number of divisions of the label values.

If any label is categorical, n_bin for that label will be equal to the number of categories.

Default is 40 for numerical labels.

is_all_num
bool, is a simplification flag indicates whether all the features are numerical or not

is_all_cat
bool, is a simplification flag indicates whether all the features are categorical or not

num_cols
list, containing name of numerical columns if needed

cat_cols
list, containing categorical feature names if needed

cmap
str, is the matplotlib colormap name, default is 'gist_rainbow'

fig_width
float, is the width of figure, default is 30

fig_height
float, is the height of figure, default is 5

rotation
float, rotational angle of x axis labels of the figure, default is 40

show
bool, whether to show the graph or not, default is True

save
bool, whether to save the figure or not, default is False

savepath
str, path where the data will be saved, default is None

fname
str, filename which will be used for saving the figure, default is None

fig_title
str, title of the figure, default is 'Feature Confidence Evaluation for categorical Features'

Outputs:

num_lab_confids
dict, contains confidence values for each label

msdlib.msd.**get_category_edges**(*dt*, *categories=None*, *names=None*)

This function calculates edges for categorical values and returns start, stop, duration and interval

Inputs:

dt
pandas Series, numpy array or list, time series signal containing categorical variable values.

categories
list or None. all possible values for the signal dt. If None, then unique values found in dt

are considered only possible values for dt.

names
dict, contains name of span DataFrame for each unique value of dt. Default is None means name of span DataFrame will be designated by its value

Outputs:

spans
list of pandas DataFrame, each DataFrame contains four columns 'start', 'stop', 'duration', 'interval'

which describes the edges of each categorical value in categories.

msdlib.msd.**get_color_from_cmap**(*n*, *cmap='jet'*, *rng=[0.05, 0.95]*)

this function gets color from matplotlib colormap

Inputs:

n
int, number of colors you want to create

cmap
str, matplotlib colormap name, default is 'jet'

rng
array, list or tuple of length 2, edges of colormap, default is [.05, .95]

Outputs:
It returns a list of colors from the selected colormap

msdlib.msd.**get_edges_from_ts**(*sr*, *th_method='median'*, *th_factor=0.5*, *th=None*, *del_side='up'*, *name=None*)

The purpose of this function is to find the edges specifically indices of start and end of certain regions by thresholding the desired time series data.

Inputs:

sr
pandas Series, time series data with proper timestamp as indices of the series. Index timestamps must be timedelta type values. Index timestamps must be sorted from old to new time (ascending order).

th_method
{'mode', 'median', 'mean'}, method of calculating threshold, Default is 'median'

th_factor
flaot/int, multiplication factor to be multiplied with th_method value to calculate threshold. Default is .5

th
int/flaot, threshold value inserted as argument. Default is None

del_side
{'up', 'down'}, indicating which side to be removed to get edges. Default is 'up'

name
name of the events, default is None

Note: the algorithm starts recording when it exceeds the threshold value, so open interval system.

Outputs:

edges
pandas DataFrame, contains columns 'start', 'end', 'duration', 'interval' etc. related to the specific events found after thresholding

msdlib.msd.**get_named_colors**()

This function returns 36 custom selected CSS4 named colors available in matplotlib

Outputs:
list of colors/color_names available in matplotlib

msdlib.msd.**get_spectrogram**(*ts_sr*, *fs=None*, *win=('tukey', 0.25)*, *nperseg=None*, *noverlap=None*, *mode='psd'*, *figsize=None*, *vis_frac=1*, *ret_sxx=False*, *show=True*, *save=False*, *savepath=None*, *fname=None*)

The purpose of this function is to find the spectrogram of a time series signal. It computes spectrogram, returns the spectrogram output and also i able to show spectrogram plot as heatmap.

Inputs:

ts_sr
pandas.Series object containing the time series data, the series should contain its name as ts_sr.name

fs
int/flaot, sampling frequency of the signal, default is 1

win
tuple of (str, float), tuple of window parameters, default is (tukey, .25)

nperseg
int, number of data in each segment of the chunk taken for STFT, default is 256

noverlap
int, number of data in overlapping region, default is (nperseg // 8)

mode
str, type of the spectrogram output, default is power spectral density('psd')

figsize
tuple, figsize of the plot, default is (30, 6)

vis_frac
float or a list of length 2, fraction of the frequency from lower to higher, you want to visualize, default is 1(full range)

ret_sxx
bool, whehter to return spectrogram dataframe or not, default is False

show
bool, whether to show the plot or not, default is True

save
bool, whether to save the figure or not, default is False

savepath
str, path to save the figure, default is None

fname
str, name of the figure to save in the savepath, default is figure title

Outputs:

sxx
returns spectrogram matrix if ret_sxx flag is set to True

msdlib.msd.**get_time_estimation**(*time_st*, *count_ratio=None*, *current_ep=None*, *current_batch=None*, *total_ep=None*, *total_batch=None*, *string_out=True*)

This function estimates remaining time inside any loop. he function is prepared for estimating remaining time in machine learning training with mini-batch. But it can be used for other purposes also by providing count_ratio input value.

Inputs:

time_st
time.time() instance indicating the starting time count

count_ratio
float, ratio of elapsed time at any moment.

Must be 0 ~ 1, where 1 will indicate that this is the last iteration of the loop

current_ep
current epoch count

current_batch
current batch count in mini-batch training

total_ep
total epoch in the training model

total_batch
total batch in the mini-batch training

string_out
bool, whether to output the elapsed and estimated time in string format or not. True will output time string

Outputs:

output time
the ouput can be a single string in format

'elapsed hour : elapsed minute : elapsed second < remaining hour : remaining minute : remaining second '

if string_out flag is True

or it can output 6 integer values in the above order for those 6 elements in the string format

msdlib.msd.**get_weighted_scores**(*score*, *y_test*)

This function calculates weighted average of score values we get from class_result function

Inputs:

score
pandas DataFrame, contains classifiaction scores that we get from msd.class_result() function

y_test
numpy array, pandas Series or python list, contains true classification labels

Outputs:

_score
pandas DataFrame, output score DataFrame with an additional column containing weighted score values

msdlib.msd.**grouped_mode**(*data*, *bins=None*, *neglect_values=[]*, *neglect_above=None*, *neglect_bellow=None*, *neglect_quan=0*)

The purpose of this function is to calculate mode (mode of mean-median-mode) value

Inputs:

data
pandas Series, list, numpy ndarray - must be 1-D

bins
int, list or ndarray, indicates bins to be tried to calculate mode value. Default is None

neglect_values
list, ndarray, the values inside the list will be removed from data. Default is []

neglect_above
float, values above this will be removed from the data. Default is None

neglect_beloow
float, values bellow this will be removed from the data. Default is None

neglect_quan
0 < float < 1 , percentile range which will be removed from both sides from data distribution. Default is 0

Outputs:
mode for the time series data

msdlib.msd.**input_variable_error**(*msg*)

msdlib.msd.**invalid_bins**(*msg*)

msdlib.msd.**moving_slope**(*df*, *fs=None*, *win=60*, *take_abs=False*, *nan_valid=True*)
The purpose of this function is to calculate the slope inside a window for a variable in a rolling method

Inputs

df
pandas DataFrame or Series, contains time series columns

fs
str, the base sampling period of the time series, default is the mode value of the difference in time between two consecutive samples

win
int, window length, deafult is 60

take_abs
bool, indicates whether to take the absolute value of the slope or not. Default is False

returns the pandas DataFrame containing resultant windowed slope values. Default is True

Outputs:

new_df
pandas DataFrame, new dataframe with calculated rolling slope

msdlib.msd.**name_separation**(*string*, *maxlen*)
This function separates a string based on its length and creates multiple lines and adds them finally to form new string with multiple lines

Inputs:

string
str, input string

maxlen
int, maximum allowable length for each line

Outputs:

newstr
str, output string after dividing the inserted string

msdlib.msd.**normalize**(*data*, *zero_range=1*, *method='min_max_0_1'*)

The purpose of this function is to apply normalization of the input data

Inputs:

data
pandas series, dataframe, list or numpy ndarray, input data to be standardized

zero_range
float, value used to replace range values in case range is 0 for any column. Default is 1

method
{'zero_mean', 'min_max_0_1', 'min_max_-1_1'}.

'zero_mean' : normalizes the data in a way that makes the data mean as 0

'min_max_0_1' : normalizes the data in a way that the data becomes confined within 0 and 1

'min_max_-1_1' : normalizes the data in a way that the data becomes confined within -1 and 1

Default is "min_max_0_1"

Outputs:
Normalized data

msdlib.msd.**one_hot_encoding**(*arr*, *class_label=None*, *ret_label=False*, *out_type='ndarray'*)

This function converts categorical values into one hot encoded format.

Inputs:

arr
numpy 1-d array, list or pandas Series, containing all class names/indices

class_label
numpy array or list, list or pandas Series, containing only class labels inside 'arr'

out_type
{'ndarray', 'dataframe', 'list'}, data type of the output

Outputs:

label
one hot encoded output data, can be ndarray, list or pandas DataFrame based on 'out_type'

class_label
names of class labels corresponding to each column of one hot encoded data.

It is provided only if ret_label is set as True.

class msdlib.msd.**paramOptimizer**(*params*, *mode='random'*, *find='min'*, *iteration=None*, *top=3*, *rand_it_perc=0.5*, *shuffle_queue=True*, *random_seed=1216*)

Bases: object

This class is used for hyper-parameter optimization for Machine Learning or other usage.

Inputs:

params
dict of lists, containing choices for each of the param keys

mode
str, search mode. avaialble modes {'random', 'grid'}, default is 'random'

find
str, available {'min', 'max'}, indicates whether minimizing or maximizing the score, default is 'min'

iteration
int, number of iterations to be done, default is None

top
int, number of top scores to be stored; default is 3

rand_it_perc
float (0 ~ 1), indirectly sets the total number of iteration in 'random' model if iteration is not given, default is .5

shuffle_queue
bool, whether to shuffle the parameter queue or not, default is True

random_seed
float, random seed to use for random actions, default is 1216

`best()`

this function provides the best parameter set in the end of optimization (can also be used at any time in stead of end)

`get_param()`

`get_queue()`

`set_score`(*score*, *element=''*)

This method takes in score value found for the current set of parameters and shows if the iteration is finished or not.

Inputs:

score
float, the score value

element
anything, any additional storage element like model or any other variable etc.

Outputs:

stop_iteration_flag
bool, if True, it means that the optimization is finished.

`msdlib.msd.plot_class_score`(*score*, *confmat*, *xrot=0*, *figure_dir=None*, *figtitle=None*, *figsize=(15, 5)*, *show=False*, *cmap='Blues'*)

This function generates figure showing the classification score and confusion matrix as tables side by side.

Inputs:

score
pandas DataFrame, contains score matrix values of all classes for all matrices

confmat
pandas DataFrame, contains confusion matrix values for all classes

xrot
float, rotation angle for x-axis labels (expected to be class names). Default is 0.

figure_dir
str or None, path to the directory where the figure will be saved. Default is None (means it wont be saved)

figsize
tuple of floats (xsize, ysize) sets the matplotlib figure size. Default is (15, 5)

show
bool, whether to show the figure or not, default is False

cmap
str, colormap name of the score and confusion matrix tables

msdlib.msd.**plot_diag**(*arr*, *bins*, *ax*, *diag*, *color*, *label*)

msdlib.msd.**plot_heatmap**(*data*, *keep='both'*, *rem_diag=False*, *cmap='gist_heat'*, *cbar=True*, *stdz=False*, *annotate=False*, *fmt=None*, *vmin=None*, *vmax=None*, *center=None*, *show=True*, *save=False*, *savepath=None*, *figsize=(30, 10)*, *fig_title=''*, *file_name=''*, *xrot=90*, *axobj=None*)

This function draws table like heatmap for a matrix data (should be pandas DataFrame)

Inputs:

data
pandas DataFrame, data to be plotted as heatmap

stdz
bool, whether to standardize the data or not, default is False

keep
{'both', 'up', 'down'}, which side of the heatmap matrix to plot, necessary for correlation heatmap plot, default is 'both'

rem_diag
bool, whether to remove diagoanl if keep_only is not 'both', default is False

cmap
str, matplotlib colormap, default is 'gist_heat'

cbar
bool, show the colorbar with the heatmap or not

annotate
bool, whether to show the values or not

fmt
str, value format for printing if annotate is True, default is None

show
bool, show the heatmap or not, default is True

save
bool, save the figure or not, default is False

savepath
str, path for saving the figure, default is None

figsize
tuple, figure size, default is (30, 10)_

fig_title
str, title of the heatmap, default is 'Heatmap of {data.columns.name}'

file_name
str, name of the image as will be saved in savepath, default is fig_title

xrot
float, rotation angle of x-axis labels, default is 0

axobj
matplotlib axes object, to draw time series on this axes object plot. Default is None

Outputs:

ax
if axobj is not None and axobj is provided as matplotlib axes object, then this function returns the axes object after drawing

msdlib.msd.**plot_regression_score**(*true*, *pred*, *figure_dir=None*, *figtitle=None*, *figsize=(10, 5)*, *show=False*, *point_color='darkcyan'*, *s=8*, *metrics={}*)

This function generates true-value VS predicted value scatter plot for regression problems.

Inputs:

true
1D list, numpy array etc. containing true values

pred
1D list, numpy array etc. containing predicted values

figure_dir
str or None, path to the directory where the figure will be saved. Default is None (means it wont be saved)

figsize
tuple of floats (xsize, ysize) sets the matplotlib figure size. Default is (10, 5)

show
bool, whether to show the figure or not, default is False

point_color
str, color of the scatter points, default is 'darkcyan'

s
float, scatter point size, default is 8

msdlib.msd.**plot_table**(*_data*, *cell_width=2.5*, *cell_height=0.625*, *font_size=14*, *header_color='#003333'*, *row_colors=['whitesmoke', 'w']*, *edge_color='w'*, *bbox=[0, 0, 1, 1]*, *header_cols=0*, *index_cols=0*, *fig_name='Table'*, *save=False*, *show=True*, *savepath=None*, *axobj=None*, ***kwargs*)

This function creates figure for a pandas dataframe table. It drops the dataframe index at first. So, if index is necessary to show, keep it as a dataframe column.

Inputs:

_data
pandas dataframe, which will be used to make table and save as a matplotlib figure. Index is removed in the begining.

So, if index is necessary to show, keep it as a dataframe column.

cell_width
float, each cell width, default is 2.5

cell_height
float, each cell height, default is 0.625

font_size
float, font size for table values, default is 14

header_color
str or rgb color code, column and index color, default is '#003333'

row_colors
str or rgb color code, alternating colors to separate consecutive rows, default is ['whitesmoke', 'w']

edge_color
str or rgb color code, cell border color, default is 'w'

bbox
list of four floats, edges of the table to cover figure, default is [0,0,1,1]

header_cols
int, number of initial rows to cover column names, default is 0

index_cols
int, number of columns to cover index of dataframe, default is 0

fig_name
str, figure name to save the figure when save is True, default is 'Table'

save
bool, whether to save the figure, default is False

show
bool, whether to show the plot, default is True

savepath
path to store the figure, default is current directory from where the code is being run ('.')

ax
matplotlib axis to include table in another figure, default is None

Outputs:

ax
matplotlib axes object used for plotting, if provided as axobj

msdlib.msd.**plot_time_series**(*same_srs*, *srs=[]*, *segs=None*, *same_srs_width=[]*, *spans=[]*, *lines=[]*, *linestyle=[]*, *linewidth=[]*, *fig_title=''*, *show=True*, *save=False*, *savepath=None*, *fname=''*, *spine_dist=0.035*, *spalpha=[]*, *ylims=[]*, *name_thres=50*, *fig_x=30*, *fig_y=4*, *marker=None*, *xlabel='Time'*, *ylabel='Data value'*, *xrot=0*, *x_ha='center'*, *axobj=None*)

This function generates plots for time series data along with much additional information if provided as argument. This function provides many flexibilities such as dividing a time series into multiple subplots to visualize easily. It can plot multiple time series with proper legends, backgrounds and grids. It can also plot span plots, veritcal lines, each with separate legends. It allows multiple axes for multiple time series data with separate value ranges.

Inputs:

same_srs
list of pandas series holding the variables which share same x-axis in matplotlib plot

srs
list of pandas series holding the variables which share different x axes, default is []

segs
int or pandas dataframe with two columns 'start' and 'stop' indicating the start and stop of each axis plot segment (subplot).

Providing int will split the time series signals into that number of segments and will show as separate row subplot.

Default is None

same_srs_width
list of flaots indicating each line width of corresponding same_srs time series data,

must be same size as length of same_srs, default is []

spans
list of pandas dataframe indicating the 'start' and 'stop' of the span plotting elements, default is []

lines
list of pandas series where the values will be datetime of each line position, keys will be just serials, default is []

linestyle
list of str, marker for each line, '' indicates continuous straight line, default is []

linewidth
list of constant, line width for each line, default is []

fig_title
title of the entire figure, default is ''

show
bool, indicating whether to show the figure or not, default is True

save
bool, indicating whether to save the figure or not, default is False

savepath
str, location of the directory where the figure will be saved, default is None

fname
str, figure name when the figure is saved, default is ''

spine_dist
constant indicating distance of the right side spines from one another if any, default is 0.035

spalpha
list of constants indicating alpha values for each of the span in spans, default is []

ylims
list of lists, indicating y axis limits in case we want to keep the limit same for all subplots, default is []

name_thres
int, maximum allowed characters in one line for plot labels, default is 50

fig_x
float, horizontal length of the figure, default is 30

fig_y
float, vertical length of each row of plot, default is 3

marker
str, marker for time series plots, default is None

xlabel
str, label name for x axis for each row, default is 'Time'

ylabel
str, label name for y axis for each row, default is 'Data value'

xrot
float, rotation angle of x-axis tick values, default is 0

x_ha
horizontal alignment of x axis tick values. It can be {'left', 'right', 'center'}. By default, it is 'center'.

axobj
matplotlib axes object, to draw time series on this axes object plot. Default is None.

To use this option, segs must be 1 or None or dataframe with 1 row.

Outputs:

axobj
Returns matplotlib axes object is axobj is provided in input.

Otherwise, this function shows or stores plots, doesnt return anything

msdlib.msd.**regression_result**(*y*, *pred*)

This function calculates R-square value (coefficient of determination) and root mean squared error value for regression evaluation.

Inputs:

y
numpy 1-d array, list or pandas Series, contains true regression labels

pred
numpy 1-d array, list or pandas Series, prediction values against y. Must be same size as y.

Outputs:

r_sq
float, calculated R-square value (coefficient of determination)

rmse
float, root mean square error value

corr
float, correlation coefficient between y and pred

msdlib.msd.**rsquare_rmse**(*y*, *pred*)

This function calculates R-square value (coefficient of determination) and root mean squared error value for regression evaluation.

Inputs:

y
numpy 1-d array, list or pandas Series, contains true regression labels

pred
numpy 1-d array, list or pandas Series, prediction values against y. Must be same size as y.

Outputs:

r_sq
calculated R-square value (coefficient of determination)

rmse
root mean square error value

msdlib.msd.**standardize**(*data*, *zero_std=1*)

This function applies z-standardization on the data

Inputs:

data
pandas series, dataframe, list or numpy ndarray, input data to be standardized

zero_std
float, value used to replace std values in case std is 0 for any column. Default is 1

Outputs:
standardized data

msdlib.msd.**word_length_error**(*msg*)

1.3 msd.processing

This is a sub-module containing functions and classes supporting data processing and data manipulation tasks. It contains Filter utilization (high, low, band pass etc.), Feature importance measurement, Spectrogram calculation and visualization, getting start-end timestamps from time series data for categorical and numerical variables and many more. Author : Abdullah Al Masud

email : abdullahalmasud.buet@gmail.com

LICENSE : MIT License

class msdlib.msd.processing.**Filters**(*T*, *N=1000*, *savepath=None*, *save=False*, *show=True*)

Bases: object

This class is used to apply FIR filters as high-pass, low-pass, band-pass and band-stop filters. It also shows the proper graphical representation of the filter response, signal before filtering and after filtering and filter spectram. The core purpose of this class is to make the filtering process super easy and check filter response comfortably.

Inputs:

T
float, indicating the sampling period of the signal, must be in seconds (doesnt have any default values)

n
int, indicating number of fft frequency bins, default is 1000

savepath
str, path to the directory to store the plot, default is None (doesnt save plots)

show
bool, whether to show the plot or not, default is True (Shows figures)

save
bool, whether to store the plot or not, default is False (doesnt save figures)

apply(*sr*, *filt_type*, *f_cut*, *order=10*, *response=False*, *plot=False*, *f_lim=[]*, *savepath=None*, *show=None*, *save=None*)

The purpose of this function is to apply an FIR filter on a time series signal 'sr' and get the output filtered signal y

Inputs:

sr
numpy ndarray/pandas Series/list, indicating the time series signal you want to apply the filter on

filt_type
str, indicating the type of filter you want to apply on sr.

{'lp', 'low_pass', 'low pass', 'lowpass'} for applying low pass filter

and similar for 'high pass', 'band pass' and 'band stop' filters

f_cut
float/list/numpy 1d array, n indicating cut off frequencies. Please follow the explanations bellow

for lowpass/highpass filters, it must be int/float

for bandpass/bandstop filters, it must be a list or numpy array of length divisible by 2

order
int, filter order, the more the values, the sharp edges at the expense of complexer computation. Default is 10.

response
bool, whether to check the frequency response of the filter or not, Default is False

plot
bool, whether to see the spectrum and time series plot of the filtered signal or not, Default is False

f_lim
list of length 2, frequency limit for the plot. Default is []

savepath
str, path to the directory to store the plot, default is None (follows Filters.savepath attribute)

show
bool, whether to show the plot or not, default is None (follows Filters.show attribute)

save
bool, whether to store the plot or not, default is None (follows Filters.save attribute)

Outputs:

y
pandas Series, filtered output signal

`filter_response`(*sos*, *f_lim=[]*, *savepath=None*, *show=None*, *save=None*)

This function plots the filter spectram in frequency domain.

Inputs:

sos
sos object found from butter function

flim
list/tuple as [lower_cutoff, higher_cutoff], default is []

savepath
str, path to the directory to store the plot, default is None (follows Filters.savepath attribute)

show
bool, whether to show the plot or not, default is None (follows Filters.show attribute)

save
bool, whether to store the plot or not, default is None (follows Filters.save attribute)

Outputs:
It doesnt return anything.

`plot_filter`(*y*, *filt_type*, *f_cut*, *f_lim=[]*, *savepath=None*, *show=None*, *save=None*)

This function plots filter frequency response, time domain signal etc.

Inputs:

y
pandas Series, time series data to filter

filt_type
str, indicating the type of filter you want to apply on y.

{'lp', 'low_pass', 'low pass', 'lowpass'} for applying low pass filter

and similar for 'high pass', 'band pass' and 'band stop' filters

f_cut
float/list/numpy 1d array, n indicating cut off frequencies. Please follow the explanations bellow

for lowpass/highpass filters, it must be int/float

for bandpass/bandstop filters, it must be a list or numpy array of length divisible by 2

f_lim
list of length 2, frequency limit for the plot. Default is []

savepath
str, path to the directory to store the plot, default is None (follows Filters.savepath attribute)

show
bool, whether to show the plot or not, default is None (follows Filters.show attribute)

save
bool, whether to store the plot or not, default is None (follows Filters.save attribute)

Outputs:
No output is returned. The function generates plot.

`raise_cutoff_error`(*msg*)

`raise_filter_error`(*msg*)

`vis_spectrum`(*sr*, *f_lim=[]*, *see_neg=False*, *show=None*, *save=None*, *savepath=None*, *figsize=(30, 3)*)

The purpose of this function is to produce spectrum of time series signal 'sr'.

Inputs:

sr
numpy ndarray or pandas Series, indicating the time series signal you want to check the spectrum for

f_lim
python list of len 2, indicating the limits of visualizing frequency spectrum. Default is []

see_neg
bool, flag indicating whether to check negative side of the spectrum or not. Default is False

show
bool, whether to show the plot or not, default is None (follows Filters.show attribute)

save
bool, whether to store the plot or not, default is None (follows Filters.save attribute)

savepath
str, path to the directory to store the plot, default is None (follows Filters.savepath attribute)

figsize
tuple, size of the figure plotted to show the fft version of the signal. Default is (30, 3)

Outputs:
doesnt return anything as the purpose is to generate plots

class msdlib.msd.processing.**SplitDataset**(*data*, *label*, *index=None*, *test_ratio=0*, *np_seed=1216*, *same_ratio=False*, *make_onehot=False*)

Bases: object

This class contains method to split data set into train, validation and test. It allows both k-fold cross validation and casual random splitting. Additionally it allows sequence splitting, necessary to model time series data using LSTM.

Inputs:

data
pandas DataFrame or Series or numpy ndarray of dimension #samples X other dimensions with rank 2 or higher in total

label
pandas dataframe or series or numpy ndarray with dimension #samples X #classes with rank 2 in total

index
numpy array, we can explicitly insert indices of the data set and labels, default is None

test_ratio
float, ratio of test data, default is 0

np_seed
int, numpy seed used for random shuffle, default is 1216

same_ratio
bool, keep the test and validation ratio same for all classes.

Should be only true if the labels are classification labels, default is False

make_onehot
bool, if True, the labels will be converted into one hot encoded format. Default is False

check_data_label()

cross_validation_split(*fold=5*, *only_index=False*)

This method applies cross validation splitting.

Inputs:

fold
int, number of folds being applied to the data, default is 5

only_index
bool, whether to return only index or data, label, index all. Default is False meaning the function will return data, label and index for all folds.

Outputs:

outdata
dict containing data, label and indices for train, validation and test data sets

`data_label_mismatch`(*msg*)

`prepare_class_index`(*same_ratio*)

`random_split`(*val_ratio=0.15*)

This method splits the data randomly into train, validation and test sets.

Inputs:

val_ratio
float, validation data set ratio, default is .15

Outputs:

outdata
dict containing data, label and indices for train, validation and test data sets

`sequence_split`(*seq_len*, *val_ratio=0.15*, *data_stride=1*, *label_shift=1*, *split_method='multiple_train_val'*, *sec=1*, *dist=0*, *inference=False*)

This method creates sequences of time series data and then splits those sequence data into train, validation and test sets. We can create mutiple pairs of train and validation data sets if we want. Test data set will be only one set.

Note : As this method creates sequential data, it may also be needed in inference time.

Inputs:

seq_len
int, length of each sequence

val_ratio
float, validation data set ratio, deafult is .15

data_stride
int, indicates the number of shift between two adjacent example data to prepare the data set, default is 1

label_shift
temporal difference between the label and the corresponding data's last time step, default is 1

split_method
{'train_val', 'multiple_train_val'}, default is 'multiple_train_val'

sec
int or pandas dataframe with columns naming 'start' and 'stop', number of sections for multiple_train_val method splitting (basically train_val splitting is multiple_train_val method with sec = 1), deafult is 1

dist
int, number of data to be leftover between two adjacent sec, default is 0

inference
bool, whether the method is called in inference or not. Default is False

Outputs:

outdata
dict containing data, label and indices for train, validation and test data sets

split_method_error(*msg*)

msdlib.msd.processing.**class_result**(*y*, *pred*, *out_confus=False*)

This function computes classification result with confusion matrix For classification result, it calculates precision, recall, f1_score, accuracy and specificity for each classes.

Inputs:

y
numpy 1-d array, list or pandas Series, true classification labels, a vector holding class names/indices for each sample

pred
numpy 1-d array, list or pandas Series, predicted values, a vector containing prediction opt for class indices similar to y.

Must be same length as y

out_confus
bool, returns confusion matrix if set True. Default is False

Outputs:

result
DataFrame containing precision, recall and f1 scores for all labels

con_mat
DataFrame containing confusion matrix, depends on out_confus

msdlib.msd.processing.**each_row_max**(*data*)

The purpose of this function is to get the maximum values and corresponding column names for each row of a matrix

Inputs:

data
list of lists/numpy ndarray or pandas dataframe, matrix data from where max values will be calculated

Outputs:
returns same data with two new columns with max values and corresponding column names

msdlib.msd.processing.**feature_evaluator**(*data*, *label_name*, *label_type_num*, *n_bin=40*, *is_all_num=False*, *is_all_cat=False*, *num_cols=[]*, *cat_cols=[]*, *cmap='gist_rainbow'*, *fig_width=30*, *fig_height=5*, *rotation=40*, *show=True*, *save=False*, *savepath=None*, *fname=None*, *fig_title=''*)

This function calculates feature importance by statistical measures. Here its okay to use labels before converting into one hot encoding, its even better not to convert into one hot.

The process is below-

——— feature evaluation criteria ———

for numerical feature evaluation:

group- a variable is divided into groups based on a constant interval of its values

group diversity- std of mean of every group

group uncertainty- mean of std of points for every group

for categorical feature evaluation:

group diversity- mean of std of percentage of categories inside a feature

group uncertainty- mean of std of percentage of groups for each category

group confidence (Importance)- group diversity / group uncertainty

Inputs:

data
must be a pandas dataframe containing feature and label data inside it

(it must contain only features and labels, no unnecessary columns are allowed)

label_name
list; containing names of the columns used as labels

label_type_num
list of bool, containing flag info if the labels are numerical or not (must be corresponding to label_name)

n_bin
int, expressing number of divisions of the label values.

If any label is categorical, n_bin for that label will be equal to the number of categories.

Default is 40 for numerical labels.

is_all_num
bool, is a simplification flag indicates whether all the features are numerical or not

is_all_cat
bool, is a simplification flag indicates whether all the features are categorical or not

num_cols
list, containing name of numerical columns if needed

cat_cols
list, containing categorical feature names if needed

cmap
str, is the matplotlib colormap name, default is 'gist_rainbow'

fig_width
float, is the width of figure, default is 30

fig_height
float, is the height of figure, default is 5

rotation
float, rotational angle of x axis labels of the figure, default is 40

show
bool, whether to show the graph or not, default is True

save
bool, whether to save the figure or not, default is False

savepath
str, path where the data will be saved, default is None

fname
str, filename which will be used for saving the figure, default is None

fig_title
str, title of the figure, default is 'Feature Confidence Evaluation for categorical Features'

Outputs:

num_lab_confids
dict, contains confidence values for each label

msdlib.msd.processing.**get_category_edges**(*dt*, *categories=None*, *names=None*)

This function calculates edges for categorical values and returns start, stop, duration and interval

Inputs:

dt
pandas Series, numpy array or list, time series signal containing categorical variable values.

categories
list or None. all possible values for the signal dt. If None, then unique values found in dt

are considered only possible values for dt.

names
dict, contains name of span DataFrame for each unique value of dt. Default is None means name of span DataFrame will be designated by its value

Outputs:

spans
list of pandas DataFrame, each DataFrame contains four columns 'start', 'stop', 'duration', 'interval'

which describes the edges of each categorical value in categories.

msdlib.msd.processing.**get_edges_from_ts**(*sr*, *th_method='median'*, *th_factor=0.5*, *th=None*, *del_side='up'*, *name=None*)

The purpose of this function is to find the edges specifically indices of start and end of certain regions by thresholding the desired time series data.

Inputs:

sr
pandas Series, time series data with proper timestamp as indices of the series. Index timestamps must be timedelta type values. Index timestamps must be sorted from old to new time (ascending order).

th_method
{'mode', 'median', 'mean'}, method of calculating threshold, Default is 'median'

th_factor
flaot/int, multiplication factor to be multiplied with th_method value to calculate threshold. Default is .5

th
int/flaot, threshold value inserted as argument. Default is None

del_side
{'up', 'down'}, indicating which side to be removed to get edges. Default is 'up'

name
name of the events, default is None

Note: the algorithm starts recording when it exceeds the threshold value, so open interval system.

Outputs:

edges
pandas DataFrame, contains columns 'start', 'end', 'duration', 'interval' etc. related to the specific events found after thresholding

msdlib.msd.processing.**get_spectrogram**(*ts_sr*, *fs=None*, *win=('tukey', 0.25)*, *nperseg=None*, *noverlap=None*, *mode='psd'*, *figsize=None*, *vis_frac=1*, *ret_sxx=False*, *show=True*, *save=False*, *savepath=None*, *fname=None*)

The purpose of this function is to find the spectrogram of a time series signal. It computes spectrogram, returns the spectrogram output and also i able to show spectrogram plot as heatmap.

Inputs:

ts_sr
pandas.Series object containing the time series data, the series should contain its name as ts_sr.name

fs
int/flaot, sampling frequency of the signal, default is 1

win
tuple of (str, float), tuple of window parameters, default is (tukey, .25)

nperseg
int, number of data in each segment of the chunk taken for STFT, default is 256

noverlap
int, number of data in overlapping region, default is (nperseg // 8)

mode
str, type of the spectrogram output, default is power spectral density('psd')

figsize
tuple, figsize of the plot, default is (30, 6)

vis_frac
float or a list of length 2, fraction of the frequency from lower to higher, you want to visualize, default is 1(full range)

ret_sxx
bool, whehter to return spectrogram dataframe or not, default is False

show
bool, whether to show the plot or not, default is True

save
bool, whether to save the figure or not, default is False

savepath
str, path to save the figure, default is None

fname
str, name of the figure to save in the savepath, default is figure title

Outputs:

sxx
returns spectrogram matrix if ret_sxx flag is set to True

msdlib.msd.processing.**get_time_estimation**(*time_st*, *count_ratio=None*, *current_ep=None*, *current_batch=None*, *total_ep=None*, *total_batch=None*, *string_out=True*)

This function estimates remaining time inside any loop. he function is prepared for estimating remaining time in machine learning training with mini-batch. But it can be used for other purposes also by providing count_ratio input value.

Inputs:

time_st
time.time() instance indicating the starting time count

count_ratio
float, ratio of elapsed time at any moment.

Must be 0 ~ 1, where 1 will indicate that this is the last iteration of the loop

current_ep
current epoch count

current_batch
current batch count in mini-batch training

total_ep
total epoch in the training model

total_batch
total batch in the mini-batch training

string_out
bool, whether to output the elapsed and estimated time in string format or not. True will output time string

Outputs:

output time
the ouput can be a single string in format

'elapsed hour : elapsed minute : elapsed second < remaining hour : remaining minute : remaining second '

if string_out flag is True

or it can output 6 integer values in the above order for those 6 elements in the string format

msdlib.msd.processing.**get_weighted_scores**(*score*, *y_test*)

This function calculates weighted average of score values we get from class_result function

Inputs:

score
pandas DataFrame, contains classifiaction scores that we get from msd.class_result() function

y_test
numpy array, pandas Series or python list, contains true classification labels

Outputs:

_score
pandas DataFrame, output score DataFrame with an additional column containing weighted score values

msdlib.msd.processing.**grouped_mode**(*data*, *bins=None*, *neglect_values=[]*, *neglect_above=None*, *neglect_bellow=None*, *neglect_quan=0*)

The purpose of this function is to calculate mode (mode of mean-median-mode) value

Inputs:

data
pandas Series, list, numpy ndarray - must be 1-D

bins
int, list or ndarray, indicates bins to be tried to calculate mode value. Default is None

neglect_values
list, ndarray, the values inside the list will be removed from data. Default is []

neglect_above
float, values above this will be removed from the data. Default is None

neglect_beloow
float, values bellow this will be removed from the data. Default is None

neglect_quan
0 < float < 1 , percentile range which will be removed from both sides from data distribution. Default is 0

Outputs:
mode for the time series data

`msdlib.msd.processing.`**`invalid_bins`**`(`*msg*`)`

`msdlib.msd.processing.`**`moving_slope`**`(`*df*, *fs=None*, *win=60*, *take_abs=False*, *nan_valid=True*`)`
The purpose of this function is to calculate the slope inside a window for a variable in a rolling method

Inputs

df
pandas DataFrame or Series, contains time series columns

fs
str, the base sampling period of the time series, default is the mode value of the difference in time between two consecutive samples

win
int, window length, deafult is 60

take_abs
bool, indicates whether to take the absolute value of the slope or not. Default is False

returns the pandas DataFrame containing resultant windowed slope values. Default is True

Outputs:

new_df
pandas DataFrame, new dataframe with calculated rolling slope

`msdlib.msd.processing.`**`normalize`**`(`*data*, *zero_range=1*, *method='min_max_0_1'*`)`
The purpose of this function is to apply normalization of the input data

Inputs:

data
pandas series, dataframe, list or numpy ndarray, input data to be standardized

zero_range
float, value used to replace range values in case range is 0 for any column. Default is 1

method
{'zero_mean', 'min_max_0_1', '**min_max_**-1_1'}.

'zero_mean' : normalizes the data in a way that makes the data mean as 0

'min_max_0_1' : normalizes the data in a way that the data becomes confined within 0 and 1

'min_max_-1_1' : normalizes the data in a way that the data becomes confined within -1 and 1

Default is "min_max_0_1"

Outputs:

Normalized data

msdlib.msd.processing.**one_hot_encoding**(*arr*, *class_label=None*, *ret_label=False*, *out_type='ndarray'*)

This function converts categorical values into one hot encoded format.

Inputs:

arr
numpy 1-d array, list or pandas Series, containing all class names/indices

class_label
numpy array or list, list or pandas Series, containing only class labels inside 'arr'

out_type
{'ndarray', 'dataframe', 'list'}, data type of the output

Outputs:

label
one hot encoded output data, can be ndarray, list or pandas DataFrame based on 'out_type'

class_label
names of class labels corresponding to each column of one hot encoded data.

It is provided only if ret_label is set as True.

class msdlib.msd.processing.**paramOptimizer**(*params*, *mode='random'*, *find='min'*, *iteration=None*, *top=3*, *rand_it_perc=0.5*, *shuffle_queue=True*, *random_seed=1216*)

Bases: object

This class is used for hyper-parameter optimization for Machine Learning or other usage.

Inputs:

params
dict of lists, containing choices for each of the param keys

mode
str, search mode. avaialble modes {'random', 'grid'}, default is 'random'

find
str, available {'min', 'max'}, indicates whether minimizing or maximizing the score, default is 'min'

iteration
int, number of iterations to be done, default is None

top
int, number of top scores to be stored; default is 3

rand_it_perc
float (0 ~ 1), indirectly sets the total number of iteration in 'random' model if iteration is not given, default is .5

shuffle_queue
bool, whether to shuffle the parameter queue or not, default is True

random_seed
float, random seed to use for random actions, default is 1216

`best()`

this function provides the best parameter set in the end of optimization (can also be used at any time in stead of end)

`get_param()`

`get_queue()`

`set_score(score, element='')`

This method takes in score value found for the current set of parameters and shows if the iteration is finished or not.

Inputs:

score
float, the score value

element
anything, any additional storage element like model or any other variable etc.

Outputs:

stop_iteration_flag
bool, if True, it means that the optimization is finished.

`msdlib.msd.processing.regression_result(y, pred)`

This function calculates R-square value (coefficient of determination) and root mean squared error value for regression evaluation.

Inputs:

y
numpy 1-d array, list or pandas Series, contains true regression labels

pred
numpy 1-d array, list or pandas Series, prediction values against y. Must be same size as y.

Outputs:

r_sq
float, calculated R-square value (coefficient of determination)

rmse
float, root mean square error value

corr
float, correlation coefficient between y and pred

`msdlib.msd.processing.rsquare_rmse(y, pred)`

This function calculates R-square value (coefficient of determination) and root mean squared error value for regression evaluation.

Inputs:

y
numpy 1-d array, list or pandas Series, contains true regression labels

pred
numpy 1-d array, list or pandas Series, prediction values against y. Must be same size as y.

Outputs:

r_sq
calculated R-square value (coefficient of determination)

rmse
root mean square error value

`msdlib.msd.processing.standardize(data, zero_std=1)`

This function applies z-standardization on the data

Inputs:

data
pandas series, dataframe, list or numpy ndarray, input data to be standardized

zero_std
float, value used to replace std values in case std is 0 for any column. Default is 1

Outputs:
standardized data

1.4 msd.vis

This is a sub-module containing functions and classes supporting data visualization tasks. It contains time series plotting function, heatmap table, grid plot, custom table preparation, plotting classification and regression metrics etc.
Author : Abdullah Al Masud

email : abdullahalmasud.buet@gmail.com

LICENSE : MIT License

`class msdlib.msd.vis.ProgressBar(arr, desc='progress', barlen=40, front_space=20, tblink_max=0.3, tblink_min=0.18)`

Bases: `object`

Inputs:

arr
iterable, it is the array you will use to run the loop, it can be range(10) or any numpy array or python list or any other iterator

desc
str, description of the loop, default - 'progress'

barlen
int, length of the progress bar, default is 40

front space
int, allowed space for description, default is 20

tblink_max
float/int indicates maximum interval in seconds between two adjacent blinks, default is .4

tblink_min
float/int indicates minimum interval in seconds between two adjacent blinks, default is .18

Outputs:
there is no output. It creates a progress bar which shows the loop progress.

barprint(*end=''*)

blink_func()

calc_time()

conv_time()

inc()

set_barelap()

msdlib.msd.vis.**data_gridplot**(*data*, *idf=[]*, *idf_pref=''*, *idf_suff=''*, *diag='hist'*, *bins=25*, *figsize=(16, 12)*, *alpha=0.7*, *s=None*, *lg_font='x-small'*, *lg_loc=1*, *fig_title=''*, *show_corr=True*, *show_stat=True*, *show=True*, *save=False*, *savepath=None*, *fname=''*, *cmap=None*)

This function creates a grid on nxn subplots showing scatter plot for each columns of 'data', n being the number of columns in 'data'. This function also shows histogram/kde/line plot for each columns along grid diagonal. The additional advantage is that we can separate each subplot data by a classifier item 'idf' which shows the class names for each row of data. This functionality is specially helpful to understand class distribution of the data. It also shows the correlation values (non-diagonal subplots) and general statistics (mean, median etc in diagonal subplots) for each subplot.

Inputs:

data
pandas dataframe, list or numpy ndarray of rank-2, columns are considered as features

idf
pandas series with length equal to total number of samples in the data, works as clussifier of each sample data,

specially useful in clustering, default is []

idf_pref
str, idf prefix, default is ''

idf_suff
str, idf suffix, default is ''

diag
{'hish', 'kde', 'plot'}, selection of diagonal plot, default is 'hist'

bins
int, number of bins for histogram plot along diagonal, default is 25

figsize
tuple, size of the whole figure, default is (16, 12)

alpha
float (0~1), transparency parameter for scatter plot, deafult is .7

s
float, point size for scatter plot, deafult is None

lg_font
float or {'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'}, legend font size, default is 'x-small'

lg_loc
int, location parameter for plot legend, deafult is 1

fig_title
str, titile of the whole figure, default is 'All Columns Grid Plot'

show_corr
bool, whether to show correlation value for each scatter plot, default is True

show_stat
bool, whether to show mean and std for each columns od data along diagonal plots, default is True

show
bool, whether to show the figure or not, default is True

save
bool, whether to save the graph or not, default is False

savepath
str, path to store the graph, default is None

fname
str, name used to store the graph in savepath, defualt is fig_title

cmap
str, matplotlib color map, default is None ('jet')

Outputs:
This function doesnt return anything.

msdlib.msd.vis.**get_color_from_cmap**(*n*, *cmap='jet'*, *rng=[0.05, 0.95]*)

this function gets color from matplotlib colormap

Inputs:

n
int, number of colors you want to create

cmap
str, matplotlib colormap name, default is 'jet'

rng
array, list or tuple of length 2, edges of colormap, default is [.05, .95]

Outputs:
It returns a list of colors from the selected colormap

msdlib.msd.vis.**get_named_colors**()

This function returns 36 custom selected CSS4 named colors available in matplotlib

Outputs:
list of colors/color_names available in matplotlib

msdlib.msd.vis.**input_variable_error**(*msg*)

msdlib.msd.vis.**name_separation**(*string*, *maxlen*)

This function separates a string based on its length and creates multiple lines and adds them finally to form new string with multiple lines

Inputs:

string
str, input string

maxlen
int, maximum allowable length for each line

Outputs:

newstr
str, output string after dividing the inserted string

msdlib.msd.vis.**plot_class_score**(*score*, *confmat*, *xrot=0*, *figure_dir=None*, *figtitle=None*, *figsize=(15, 5)*, *show=False*, *cmap='Blues'*)

This function generates figure showing the classification score and confusion matrix as tables side by side.

Inputs:

score
pandas DataFrame, contains score matrix values of all classes for all matrices

confmat
pandas DataFrame, contains confusion matrix values for all classes

xrot
float, rotation angle for x-axis labels (expected to be class names). Default is 0.

figure_dir
str or None, path to the directory where the figure will be saved. Default is None (means it wont be saved)

figsize
tuple of floats (xsize, ysize) sets the matplotlib figure size. Default is (15, 5)

show
bool, whether to show the figure or not, default is False

cmap
str, colormap name of the score and confusion matrix tables

msdlib.msd.vis.**plot_diag**(*arr*, *bins*, *ax*, *diag*, *color*, *label*)

msdlib.msd.vis.**plot_heatmap**(*data*, *keep='both'*, *rem_diag=False*, *cmap='gist_heat'*, *cbar=True*, *stdz=False*, *annotate=False*, *fmt=None*, *vmin=None*, *vmax=None*, *center=None*, *show=True*, *save=False*, *savepath=None*, *figsize=(30, 10)*, *fig_title=''*, *file_name=''*, *xrot=90*, *axobj=None*)

This function draws table like heatmap for a matrix data (should be pandas DataFrame)

Inputs:

data
pandas DataFrame, data to be plotted as heatmap

stdz
bool, whether to standardize the data or not, default is False

keep
{'both', 'up', 'down'}, which side of the heatmap matrix to plot, necessary for correlation heatmap plot, default is 'both'

rem_diag
bool, whether to remove diagoanl if keep_only is not 'both', default is False

cmap
str, matplotlib colormap, default is 'gist_heat'

cbar
bool, show the colorbar with the heatmap or not

annotate
bool, whether to show the values or not

fmt
str, value format for printing if annotate is True, default is None

show
bool, show the heatmap or not, default is True

save
bool, save the figure or not, default is False

savepath
str, path for saving the figure, default is None

figsize
tuple, figure size, default is (30, 10)_

fig_title
str, title of the heatmap, default is 'Heatmap of {data.columns.name}'

file_name
str, name of the image as will be saved in savepath, default is fig_title

xrot
float, rotation angle of x-axis labels, default is 0

axobj
matplotlib axes object, to draw time series on this axes object plot. Default is None

Outputs:

ax
if axobj is not None and axobj is provided as matplotlib axes object, then this function returns the axes object after drawing

`msdlib.msd.vis.`**`plot_regression_score`**`(`*true*, *pred*, *figure_dir=None*, *figtitle=None*, *figsize=(10, 5)*, *show=False*, *point_color='darkcyan'*, *s=8*, *metrics={}*`)`

This function generates true-value VS predicted value scatter plot for regression problems.

Inputs:

true
1D list, numpy array etc. containing true values

pred
1D list, numpy array etc. containing predicted values

figure_dir
str or None, path to the directory where the figure will be saved. Default is None (means it wont be saved)

figsize
tuple of floats (xsize, ysize) sets the matplotlib figure size. Default is (10, 5)

show
bool, whether to show the figure or not, default is False

point_color
str, color of the scatter points, default is 'darkcyan'

s
float, scatter point size, default is 8

msdlib.msd.vis.**plot_table**(*_data*, *cell_width=2.5*, *cell_height=0.625*, *font_size=14*, *header_color='#003333'*, *row_colors=['whitesmoke', 'w']*, *edge_color='w'*, *bbox=[0, 0, 1, 1]*, *header_cols=0*, *index_cols=0*, *fig_name='Table'*, *save=False*, *show=True*, *savepath=None*, *axobj=None*, ***kwargs*)

This function creates figure for a pandas dataframe table. It drops the dataframe index at first. So, if index is necessary to show, keep it as a dataframe column.

Inputs:

_data
pandas dataframe, which will be used to make table and save as a matplotlib figure. Index is removed in the begining.

So, if index is necessary to show, keep it as a dataframe column.

cell_width
float, each cell width, default is 2.5

cell_height
float, each cell height, default is 0.625

font_size
float, font size for table values, default is 14

header_color
str or rgb color code, column and index color, default is '#003333'

row_colors
str or rgb color code, alternating colors to separate consecutive rows, default is ['whitesmoke', 'w']

edge_color
str or rgb color code, cell border color, default is 'w'

bbox
list of four floats, edges of the table to cover figure, default is [0,0,1,1]

header_cols
int, number of initial rows to cover column names, default is 0

index_cols
int, number of columns to cover index of dataframe, default is 0

fig_name
str, figure name to save the figure when save is True, default is 'Table'

save
bool, whether to save the figure, default is False

show
bool, whether to show the plot, default is True

savepath
path to store the figure, default is current directory from where the code is being run ('.')

ax
matplotlib axis to include table in another figure, default is None

Outputs:

ax
matplotlib axes object used for plotting, if provided as axobj

msdlib.msd.vis.**plot_time_series**(*same_srs*, *srs=[]*, *segs=None*, *same_srs_width=[]*, *spans=[]*, *lines=[]*, *linestyle=[]*, *linewidth=[]*, *fig_title=''*, *show=True*, *save=False*, *savepath=None*, *fname=''*, *spine_dist=0.035*, *spalpha=[]*, *ylims=[]*, *name_thres=50*, *fig_x=30*, *fig_y=4*, *marker=None*, *xlabel='Time'*, *ylabel='Data value'*, *xrot=0*, *x_ha='center'*, *axobj=None*)

This function generates plots for time series data along with much additional information if provided as argument. This function provides many flexibilities such as dividing a time series into multiple subplots to visualize easily. It can plot multiple time series with proper legends, backgrounds and grids. It can also plot span plots, veritcal lines, each with separate legends. It allows multiple axes for multiple time series data with separate value ranges.

Inputs:

same_srs
list of pandas series holding the variables which share same x-axis in matplotlib plot

srs
list of pandas series holding the variables which share different x axes, default is []

segs
int or pandas dataframe with two columns 'start' and 'stop' indicating the start and stop of each axis plot segment (subplot).

Providing int will split the time series signals into that number of segments and will show as separate row subplot.

Default is None

same_srs_width
list of flaots indicating each line width of corresponding same_srs time series data,

must be same size as length of same_srs, default is []

spans
list of pandas dataframe indicating the 'start' and 'stop' of the span plotting elements, default is []

lines
list of pandas series where the values will be datetime of each line position, keys will be just serials, default is []

linestyle
list of str, marker for each line, '' indicates continuous straight line, default is []

linewidth
list of constant, line width for each line, default is []

fig_title
title of the entire figure, default is ''

show
bool, indicating whether to show the figure or not, default is True

save
bool, indicating whether to save the figure or not, default is False

savepath
str, location of the directory where the figure will be saved, default is None

fname
str, figure name when the figure is saved, default is ''

spine_dist
constant indicating distance of the right side spines from one another if any, default is 0.035

spalpha
list of constants indicating alpha values for each of the span in spans, default is []

ylims
list of lists, indicating y axis limits in case we want to keep the limit same for all subplots, default is []

name_thres
int, maximum allowed characters in one line for plot labels, default is 50

fig_x
float, horizontal length of the figure, default is 30

fig_y
float, vertical length of each row of plot, default is 3

marker
str, marker for time series plots, default is None

xlabel
str, label name for x axis for each row, default is 'Time'

ylabel
str, label name for y axis for each row, default is 'Data value'

xrot
float, rotation angle of x-axis tick values, default is 0

x_ha
horizontal alignment of x axis tick values. It can be {'left', 'right', 'center'}. By default, it is 'center'.

axobj
matplotlib axes object, to draw time series on this axes object plot. Default is None.

To use this option, segs must be 1 or None or dataframe with 1 row.

Outputs:

axobj
Returns matplotlib axes object is axobj is provided in input.

Otherwise, this function shows or stores plots, doesnt return anything

`msdlib.msd.vis.`**`word_length_error`**`(`*`msg`*`)`

1.5 mlutils

This module is intended to provide support for easy usage of pytorch models. It provides support for using Pytorch models in a few lines of code, has similar (not exactly same) functionalities as scikit-learn models. We can train the model using simple 'fit' method, can generate predictions using 'predict' method and can evaluate model performance using 'evaluate' method.

It can also enable us to build Deep Neural Network models easily with some other helper functions. Author : Abdullah Al Masud

email : abdullahalmasud.buet@gmail.com

LICENSE : MIT License

class msdlib.mlutils.**AutoEncoderModel**(*enc_layers*, *dec_layers*, *seed_value=1216*)

Bases: Module

This class creates an auto-encoder model.

Inputs:

enc_layers
python list, containing the encoder layers (torch.nn.Module class objects) sequentially

dec_layers
python list, containing the decoder layers (torch.nn.Module class objects) sequentially

seed_value
float/int, random seed for reproducibility

decode(*x*)

Decoder part in Autoencoder model

Inputs:

x
input tensor for decoder part

Outputs:
decoder output

encode(*x*)

Encoder part in Autoencoder model

Inputs:

x
input tensor for encoder part

Outputs:
encoder output

forward(*x*)

pytorch forward function for forward propagation, applies encoder and then decoder sequentially on the input data

Inputs:

x
input tensor for autoencoder model

Outputs:

x
final output of the whole auto-encoder model

training: bool

class msdlib.mlutils.**DataSet**(*data*, *label=None*, *dtype=torch.float32*, *model_type='regressor'*)

Bases: Dataset

This is a customized Data set object which can build a torch.utils.data.Dataset object given the data and labels. This is only usable when we have complete data and labels (data and label lengths must be equal)

Inputs:

data
ideally should be torch tensor. Contains feature data tor model training. Can be python list or python set too but not much appreciated as it will be mostly used for training pytorch model.

label
ideally should be numpy ndarray, pandas Series/DataFrame or torch tensor. Contains true labels tor model training. Can be python list or python set too but not much appreciated as it will be mostly used for training pytorch model.

dtype
data type of the data and labels. Default is torch.float32. It also depends on model_type parameter for label data.

model_type
{'regressor', 'binary-classifier' or 'multi-classifier'}. Default is 'multi-classifier'. It is used to confirm that for multi-class classification, label data type is torch.long.

class msdlib.mlutils.**NNmodel**(*layer_funcs*, *seed_value=1216*)

Bases: Module

This class constructs a deep neural network model upon providing the layers we intend to build the model with.

Inputs:

layer_funcs
list, contains sequential layer classes (nn.Module).

For example-

[nn.Linear(50), nn.ReLU(), nn.Linear(3), nn.Softmax(dim=-1)]

seed_value
float/int, random seed for reproducibility, default is 1216

forward(*x*)

pytorch forward function for forward propagation

training: bool

msdlib.mlutils.**define_layers**(*input_units*, *output_units*, *unit_factors*, *dropout_rate=None*, *model_type='regressor'*, *actual_units=False*, *apply_bn=False*, *activation=None*, *final_activation=None*)

This function takes a common formation/sequence of functions to construct one hidden layer and then replicates this sequence for multiple hidden layers. Hidden layer units are decided based on 'unit_factors' paramter. The common formation of one hidden layer is this-

-Linear -BatchNorm1d -ReLU -Dropout

Dropout ratio is same in all layers

Finally the output layer is constructed depending on 'output_units' and 'model_type' parameter including final activation function. Output activation function is provided

Inputs:

input_units
int, number of units in input layer / number of features (not first hidden layer)

output_units
int, number of units in output layer / number of output nodes / number of classes (not last hidden layer)

unit_factors
array of ints or floats, multipliers to calculate number of units in each hidden layer from input_units, or actual number of units for each hidden layer

dropout_rate
dropout ratio, must be 0 ~ 1. Default is None (no dropout layer)

model_type
{'binary-classifier', 'multi-classifier, 'regressor'}, controls use of softmax/sigmoid at the output layer. Use 'regressor' if you dont intend to use any activation at output. Default is 'regressor'

actual_units
bool, whether actual units are placed in unit_factors or not, default is False (not actual units, instead unit_factos is containing ratios)

apply_bn
bool, whether to use batch normalization or not, default is False (does not use batch normalization)

activation
nn.Module object or None. Pytorch activation layer that will be used as activation function after each hidden layer. Default is None (No activation)

final_activation
torch.sigmoid / torch.Softmax(dim=1) / torch.tanh etc. for output layer, default is None. If None, the final activation will be below: - modey_type == 'regressor' –> No activation - model_type == 'binary-classifier' –> torch.sigmoid - model_type == 'multi-clussifier' –> torch.Softmax(dim=1)

Outputs:

layers
list of Deep Learning model layers which can be fed as NNModel layer_funcs input or torchModel layers input to build DNN model

msdlib.mlutils.**evaluate_with_data**(*outdata*, *models*, *figure_dir=None*, *model_type='multi-classifier'*, *figsize=(15, 5)*, *xrot=0*)

This function will be used to train models. We can use both scikit-models and torchModel objects for model training.

Inputs:

outdata
dict, contains dicts with structure like : outdata = {'train': {'data': <numpy-array>, 'label': <numpy-array>, 'index': <numpy-array>}} 'train' dict is mandatory. 'validation' dict is mandatory for torchModel objects inside "models" argument. 'data', 'label' and 'index' must be of same length.

models
dict, contains scikit-models, xgboost, lightgbm etc. scikit-like models and torchModel objects. If its a torchModel object, the key must contain 'pytorch' in it.

figure_dir
string/None, path to the directory where figures will be saved for feature improtances and evaluation figures.

model_type
string, can be either 'regressor', 'multi-classifier' or 'binary-classifier'. It controls the evaluation process.

figsize
tuple of horiaontal and vertical size of the result figure (regression score and classification score both). Default is (15, 5).

xrot
float, rotation angle of the x axis labels in classification score matrix (where class label names are written). Default is 0 (no rotation).

Outputs:

predictions
dict, contains detailed predictions on all sets in outdata argument, evaluation results etc.

msdlib.mlutils.**get_factors**(*n_layers*, *base_factor=5*, *max_factor=10*, *offset_factor=2*)

This function calculates factors/multipliers to calculate number of units inside define_layers() function

Inputs:

n_layers
number of hidden layers

max_factor
multiplier for mid layer (largest layer)

base_factor
multiplier for first layer

offset_factor
makes assymetric structure in output with factor (base - offset). For symmetric model (size in first and last hidden layer is same), offset will be 0.

Outputs:

factors
multipliers to calculate number of units in each layers based on input shape

msdlib.mlutils.**instantiate_models**()

This is not a real function, this is a dummy function which shows how a "models" dict looks like. This models dict can be used for input arguments for the function "train_with_data"

msdlib.mlutils.**load_models**(*models*, *folder_path*)

This function loads different scikit-learn models from .pickle format and Pytorch model from .pt formatted state_dict (only weights)

Inputs:

models
dict, containing model classes or None (for torch model, torch.nn.Module object is necessary as trained model to load the state variables. For other types of models like xgboost etc. None is fine.); For pytorch models, the key must contain 'pytorch' phrase (Case insensitive) key name must be like this :

stored model file name: xgboost_model.pickle

corresponding key for the dict: 'xgboost'

stored model file name: pytorch-1_model.pickle

corresponding key for the dict: 'pytorch-1'

for pytorch model, the model must not be a DataParallel model. You can add parallelism after loading the weights

folder_path
str, directory path from where the stored models will be loaded

Outputs:

models
dict, containing model classes like {<model_name> : <model_class>}

msdlib.mlutils.**store_models**(*models*, *folder_path*)

This function stores different types of scikit-learn models in .pickle format and also Pytorch model in .pt format

Inputs:

models
dict, containing only trained model class; {<model name>: <model class>} For pytorch models, the key must contain 'pytorch' phrase (Case insensitive).

Note: Pytorch model must not be a DataParallel model.

If its a DataParallel model, then take module attribute of your model like this- {'pytorch': <your_model>.module}

folder_path
str, the folder path where the models will be stores, if doesnt exist, it will be created

class msdlib.mlutils.**torchModel**(*layers=[]*, *loss_func=None*, *optimizer=None*, *learning_rate=0.0001*, *epoch=2*, *batch_size=32*, *lr_reduce=1*, *loss_reduction='mean'*, *model_type='regressor'*, *use_gpu=True*, *gpu_devices=None*, *model_name='pytorch'*, *dtype=torch.float32*, *plot_loss=True*, *quant_perc=0.98*, *loss_roll_period=1*, *model=None*, *savepath=None*, *shuffle=True*, *lr_scheduler=None*, *tensorboard_path=None*, *tb_metrics=None*, *interval=None*, *class_xrot=0*, ***kwargs*)

Bases: object

This class controls the deep neural network model training, inference, prediction and evaluation. It can provide the training loss curves and evaluation results very nicely. It is capable of using multiple GPUs.

Note: For classification problems (both binary and multi-class), there is no need to convert labels into one hot encoded format. In stead, the class label values should be indices like 0, 1, 2…

Inputs:

layers
a list of torch.nn.Module objects indicating layers/activation functions. The list should contain all elements sequentially. Default is []. Each element of this list must be callable function object. For Sigmoid function, we can use torch.nn.Sigmoid() or torch.sigmoid (use the brackets this way).

loss_func
loss function for the ML model. default is torch.nn.MSELoss [without brackets ()]. It can also be a custom loss function, but should be equivalent to the default

optimizer
optimizer for the ML model. default is torch.optim.Adam

learning_rate
learning rate of the training steps, default is .0001

epoch
number of epoch for training, default is 2

batch_size
int or None, mini-batch size for training, default is 32. Batch size is neglected while utilizing torch dataloader object for training and prediction.

lr_reduce
learning rate reduction base for lambda reduction scheduler from pytorch (follows torch.optim.lr_scheduler.LambdaLR). Must be 0 ~ 1. Default is 1 (No reduction of learning rate during training)

loss_reduction
loss reduction parameter for loss calculation, default is 'mean'

model_type
type of the model depending on the objective, should be any of {'regressor', 'binary-classifier', 'multi-classifier'}, default is 'regressor'. - 'binary-classifier' indicates binary classifier with 1 output unit. The output values must be 0 ~ 1 (sigmoid like activations should be used at model output). - 'multi-classifier' indicates classifier with more than one output unit

use_gpu
bool, whether to use gpu or not, default is True.

gpu_devices
list, a list of cuda ids starting from 0. Default is None which will try to use all available gpus. If you have 4 gpus in your machine, and want to use first 3 of them, the list should be [0, 1, 2] It can also be used to select a particular GPU device. For example to use GPU device 1, make it [1]

model_name
str, name of the model, default is 'pytorch'

dtype
dtype of processing inside the model, default is torch.float32

plot_loss
bool, whether to plot loss curves after training or not, default is True

quant_perc
float, quantile value to limit the loss values for loss curves, default is .98

loss_roll_preiod
int, rolling/moving average period for loss curve

model
torch.nn.Module class (ML model class), so that we are able to write the model ourselves and use fit, predict etc methods from here.

savepath
str, path to store the learning curve and evaluation results

shuffle
bool, whether the training dataset should be shuffled during training or not. Default is True (data set will be shuffles).

tensorboard_path
str or None, tensorboard will show the progress of training using loss values and training metrics. - If model_type is 'regressor', metrics will be rmse, rsquare - If model_type is 'binary-classifier' or 'multi-classifier', metrics will be 'accuracy'

tb_metrics
None or list of functions. Each function will take two inputs, first is true labels and second is

predicted values. If None, the metrics will be selected based on model type according to the description of parameter 'tensorboard_path'.

interval
int or None, indicates number of epoch after which the model weights will be stored each time during training. Default is None means model will not be stored during training. Note: "model_name" parameter must include 'pytorch' phrase in it.

class_xrot
float, rotation angle of the x-axis label for classification score plot (both score matrix and confusion matrix). Only applicable if evaluation is done.

`add_tb_params`(*ep*, *tr_mean_loss*, *val_loss*, *true*, *pred*, *batch_data*)

This function adds tensorboard parameters for training session records.

Inputs:

ep
int, current epoch number.

tr_mean_loss
float, average training loss

val_loss
float, average validation loss

true
torch.Tensor, true labels of validation data

pred
torch.Tensor, predicted labels from the model for validation data

batch_data
torch.Tensor, last minibatch data.

`evaluate`(*data_sets*, *label_sets=None*, *set_names=[]*, *figsize=(18, 4)*, *savepath=None*, *index2label=None*)

This is a customized function to evaluate model performance in regression and classification type tasks

Inputs:

data_sets
list of data, data can be pytorch dataloader object, numpy ndarray, torch tensor or Pandas DataFrame/Series. If its a list of dataloaders, the dataloader must return data and label as pair through __getitem__().

label_sets
list of labels corresponding to each data, label must be numpy ndarray, torch tensor or Pandas DataFrame/Series. Default is None. While using data-loader for 'data_sets' argument, label_sets doesnt have any effect.

set_names
names of the data sets, default is []

figsize
figure size for the evaluation plots, default is (18, 4)

savepath
path where the evaluation tables/figures will be stored. Default is None (by default, if this path is None, savepath will be replaced by torchModel.savepath)

index2label
python dict or None. It is only application for classification task to replace the class indices

with their corresponding labels. So, the keys will be index values and dictionary values will be actual class labels.

Outputs:

summary_result
pandas DataFrame, result summary accumulated in a variable

all_results
dict, complete results for all datasets

`fit`(*data=None*, *label=None*, *val_data=None*, *val_label=None*, *validation_ratio=0.15*, *evaluate=True*, *figsize=(18, 4)*, *train_loader=None*, *val_loader=None*)

scikit like wrapper for training DNN pytorch model

Inputs:

data
input train data, must be torch tensor, numpy ndarray or pandas DataFrame/Series. Default value in None

label
supervised labels for data, must be torch tensor, numpy ndarray or pandas DataFrame/Series. Default value in None

val_data
validation data, must be torch tensor, numpy ndarray or pandas DataFrame/Series, default is None

val_label
supervised labels for val_data, must be torch tensor, numpy ndarray or pandas DataFrame/Series, default is None

validation_ratio
ratio of 'data' that will be used for validation during training. It will be used only when val_data or val_label or both are None. Default is 0.15

evaluate
bool, whether to evaluate model performance after training ends or not. evaluate performance if set True. Default is True.

figsize
tuple of (width, height) of the figure, size of the figure for loss curve plot and evaluation plots. Default is (18, 4)

train_loader
torch.utils.data.DataLaoder instance for handling training data set. Default is None

val_loader
torch.utils.data.DataLaoder instance for handling validation data set. Default is None

Outputs:

self
torchModel object, returns the self object variable just like scikit-learn model objects

`predict`(*data*, *return_label=False*)

A wrapper function that generates prediction from pytorch model

Inputs:

data
input data to predict on, can be a torch tensor, numpy ndarray, pandas DataFrame or even torch DataLoader object

return_label
bool, whether to return label data if 'data' is a pytorch DataLoader object. Default is False

Outputs:

preds
torch Tensor, predicted values against the inserted data

labels
torch Tensor, only found if 'data' is pytorch DataLoader and return_label is True.

`prepare_data_loader`(*data*, *label*, *val_data*, *val_label*, *validation_ratio*, *train_loader*, *val_loader*)

This function prepares the data format and creates training and validation data loader from training and validation data. If no validation data is provided, it uses validation_ratio to split training data into train and validation and creates separate data loader for each set. If direct train and validation data loaders are provided, then no additional processing is used.

Inputs:

train_data
input training data, must be torch tensor, numpy ndarray or pandas DataFrame/Series. Default value in None

train_label
supervised labels for train_data, must be torch tensor, numpy ndarray or pandas DataFrame/Series. Default value in None

val_data
validation data, must be torch tensor, numpy ndarray or pandas DataFrame/Series, default is None. Note: If val_data is not provided, validation data will be taken from training data set using validation_ratio.

val_label
supervised labels for val_data, must be torch tensor, numpy ndarray or pandas DataFrame/Series, default is None

validation_ratio
ratio of 'data' that will be used for validation during training. It will be used only when val_data or val_label or both are None. Default is 0.15

train_loader
torch.utils.data.DataLaoder instance for handling training data set or None. Default is None. Note: If train_loader is provided, its expected that val_loader will also be provided.

val_loader
torch.utils.data.DataLaoder instance for handling validation data set or None. Default is None

Outputs:

train_loader
torch.utils.data.DataLaoder instance for handling training data set.

val_loader
torch.utils.data.DataLaoder instance for handling validation data set.

`relieve_parallel`()

This function reverts the DataParallel model to simple torch.nn.Module model

set_parallel()

This method sets multiple GPUs to use for training depending on the machine's cuda availability and use_gpu parameter.

msdlib.mlutils.**train_with_data**(*outdata*, *feature_columns*, *models*, *featimp_models=[]*, *figure_dir=None*, *model_type='multi-classifier'*, *evaluate=True*, *featimp_figsize=(30, 5)*, *figsize=(15, 5)*, *xrot=0*)

This function will be used to train models. We can use both scikit-models and torchModel objects for model training.

Inputs:

outdata
dict, contains dicts with structure like : outdata = {'train': {'data': <numpy-array>, 'label': <numpy-array>, 'index': <numpy-array>}} 'train' dict is mandatory. 'validation' dict is mandatory for torchModel objects inside "models" argument. 'data', 'label' and 'index' must be of same length.

feature_columns
list/numpy array, contains feature names of 'data' inside outdata. seature_columns length must be equal to number of columns in 'data'

models
dict, contains scikit-models, xgboost, lightgbm etc. scikit-like models and torchModel objects. If its a torchModel object, the key must contain 'pytorch' in it.

featimp_models
list of strings, contains model names which belong to models dict keys and which can provide feature importances

figure_dir
string/None, path to the directory where figures will be saved for feature improtances and evaluation figures.

model_type
string, can be either 'regressor', 'multi-classifier' or 'binary-classifier'. It controls the evaluation process.

evaluate
boolean, If True, model evaluation will be executed and results of evaluation will be stored inside figure_dir. Default is True.

featimp_figsize
tuple of horiaontal and vertical size of the feature importance plot. Default is (30, 5).

figsize
tuple of horiaontal and vertical size of the result figure (regression score and classification score both). Default is (15, 5). Only effective when evaluate is True.

xrot
float, rotation angle of the x axis labels in classification score matrix (where class label names are written). Default is 0 (no rotation). Only effective when evaluate is True.

Outputs:

models
dict, contains trained models instances, same as 'models' argument

predictions
dict, contains detailed predictions on all sets in outdata argument, evaluation results etc.

1.6 mlutils.modeling

This is a sub-module containing torchModel class and related items for easy pytorch implementation and testing. torchModel class contains several functions whih can be safely used for pytorch model training (using .fit() method), prediction (using .predict() method) and evaluation (using .evaluate() method). Author : Abdullah Al Masud

email : abdullahalmasud.buet@gmail.com

LICENSE : MIT License

class msdlib.mlutils.modeling.**AutoEncoderModel**(*enc_layers*, *dec_layers*, *seed_value=1216*)

Bases: Module

This class creates an auto-encoder model.

Inputs:

enc_layers
python list, containing the encoder layers (torch.nn.Module class objects) sequentially

dec_layers
python list, containing the decoder layers (torch.nn.Module class objects) sequentially

seed_value
float/int, random seed for reproducibility

decode(*x*)

Decoder part in Autoencoder model

Inputs:

x
input tensor for decoder part

Outputs:
decoder output

encode(*x*)

Encoder part in Autoencoder model

Inputs:

x
input tensor for encoder part

Outputs:
encoder output

forward(*x*)

pytorch forward function for forward propagation, applies encoder and then decoder sequentially on the input data

Inputs:

x
input tensor for autoencoder model

Outputs:

x
final output of the whole auto-encoder model

training: bool

class msdlib.mlutils.modeling.**NNmodel**(*layer_funcs*, *seed_value=1216*)

Bases: Module

This class constructs a deep neural network model upon providing the layers we intend to build the model with.

Inputs:

layer_funcs
list, contains sequential layer classes (nn.Module).

For example-

[nn.Linear(50), nn.ReLU(), nn.Linear(3), nn.Softmax(dim=-1)]

seed_value
float/int, random seed for reproducibility, default is 1216

forward(*x*)

pytorch forward function for forward propagation

training: bool

msdlib.mlutils.modeling.**instantiate_models**()

This is not a real function, this is a dummy function which shows how a "models" dict looks like. This models dict can be used for input arguments for the function "train_with_data"

class msdlib.mlutils.modeling.**torchModel**(*layers=[]*, *loss_func=None*, *optimizer=None*, *learning_rate=0.0001*, *epoch=2*, *batch_size=32*, *lr_reduce=1*, *loss_reduction='mean'*, *model_type='regressor'*, *use_gpu=True*, *gpu_devices=None*, *model_name='pytorch'*, *dtype=torch.float32*, *plot_loss=True*, *quant_perc=0.98*, *loss_roll_period=1*, *model=None*, *savepath=None*, *shuffle=True*, *lr_scheduler=None*, *tensorboard_path=None*, *tb_metrics=None*, *interval=None*, *class_xrot=0*, ***kwargs*)

Bases: object

This class controls the deep neural network model training, inference, prediction and evaluation. It can provide the training loss curves and evaluation results very nicely. It is capable of using multiple GPUs.

Note: For classification problems (both binary and multi-class), there is no need to convert labels into one hot encoded format. In stead, the class label values should be indices like 0, 1, 2…

Inputs:

layers
a list of torch.nn.Module objects indicating layers/activation functions. The list should contain all elements sequentially. Default is []. Each element of this list must be callable function object. For Sigmoid function, we can use torch.nn.Sigmoid() or torch.sigmoid (use the brackets this way).

loss_func
loss function for the ML model. default is torch.nn.MSELoss [without brackets ()]. It can also be a custom loss function, but should be equivalent to the default

optimizer
optimizer for the ML model. default is torch.optim.Adam

learning_rate
learning rate of the training steps, default is .0001

epoch
number of epoch for training, default is 2

batch_size
int or None, mini-batch size for training, default is 32. Batch size is neglected while utilizing torch dataloader object for training and prediction.

lr_reduce
learning rate reduction base for lambda reduction scheduler from pytorch (follows torch.optim.lr_scheduler.LambdaLR). Must be 0 ~ 1. Default is 1 (No reduction of learning rate during training)

loss_reduction
loss reduction parameter for loss calculation, default is 'mean'

model_type
type of the model depending on the objective, should be any of {'regressor', 'binary-classifier', 'multi-classifier'}, default is 'regressor'. - 'binary-classifier' indicates binary classifier with 1 output unit. The output values must be 0 ~ 1 (sigmoid like activations should be used at model output). - 'multi-classifier' indicates classifier with more than one output unit

use_gpu
bool, whether to use gpu or not, default is True.

gpu_devices
list, a list of cuda ids starting from 0. Default is None which will try to use all available gpus. If you have 4 gpus in your machine, and want to use first 3 of them, the list should be [0, 1, 2] It can also be used to select a particular GPU device. For example to use GPU device 1, make it [1]

model_name
str, name of the model, default is 'pytorch'

dtype
dtype of processing inside the model, default is torch.float32

plot_loss
bool, whether to plot loss curves after training or not, default is True

quant_perc
float, quantile value to limit the loss values for loss curves, default is .98

loss_roll_preiod
int, rolling/moving average period for loss curve

model
torch.nn.Module class (ML model class), so that we are able to write the model ourselves and use fit, predict etc methods from here.

savepath
str, path to store the learning curve and evaluation results

shuffle
bool, whether the training dataset should be shuffled during training or not. Default is True (data set will be shuffles).

tensorboard_path
str or None, tensorboard will show the progress of training using loss values and training metrics. - If model_type is 'regressor', metrics will be rmse, rsquare - If model_type is 'binary-classifier' or 'multi-classifier', metrics will be 'accuracy'

tb_metrics
None or list of functions. Each function will take two inputs, first is true labels and second is predicted values. If None, the metrics will be selected based on model type according to the description of parameter 'tensorboard_path'.

interval
int or None, indicates number of epoch after which the model weights will be stored each time during training. Default is None means model will not be stored during training. Note: "model_name" parameter must include 'pytorch' phrase in it.

class_xrot
float, rotation angle of the x-axis label for classification score plot (both score matrix and confusion matrix). Only applicable if evaluation is done.

`add_tb_params`(*ep*, *tr_mean_loss*, *val_loss*, *true*, *pred*, *batch_data*)

This function adds tensorboard parameters for training session records.

Inputs:

ep
int, current epoch number.

tr_mean_loss
float, average training loss

val_loss
float, average validation loss

true
torch.Tensor, true labels of validation data

pred
torch.Tensor, predicted labels from the model for validation data

batch_data
torch.Tensor, last minibatch data.

`evaluate`(*data_sets*, *label_sets=None*, *set_names=[]*, *figsize=(18, 4)*, *savepath=None*, *index2label=None*)

This is a customized function to evaluate model performance in regression and classification type tasks

Inputs:

data_sets
list of data, data can be pytorch dataloader object, numpy ndarray, torch tensor or Pandas DataFrame/Series. If its a list of dataloaders, the dataloader must return data and label as pair through __getitem__().

label_sets
list of labels corresponding to each data, label must be numpy ndarray, torch tensor or Pandas DataFrame/Series. Default is None. While using data-loader for 'data_sets' argument, label_sets doesnt have any effect.

set_names
names of the data sets, default is []

figsize
figure size for the evaluation plots, default is (18, 4)

savepath
path where the evaluation tables/figures will be stored. Default is None (by default, if this path is None, savepath will be replaced by torchModel.savepath)

index2label
python dict or None. It is only application for classification task to replace the class indices with their corresponding labels. So, the keys will be index values and dictionary values will be actual class labels.

Outputs:

summary_result
pandas DataFrame, result summary accumulated in a variable

all_results
dict, complete results for all datasets

`fit`(*data=None*, *label=None*, *val_data=None*, *val_label=None*, *validation_ratio=0.15*, *evaluate=True*, *figsize=(18, 4)*, *train_loader=None*, *val_loader=None*)

scikit like wrapper for training DNN pytorch model

Inputs:

data
input train data, must be torch tensor, numpy ndarray or pandas DataFrame/Series. Default value in None

label
supervised labels for data, must be torch tensor, numpy ndarray or pandas DataFrame/Series. Default value in None

val_data
validation data, must be torch tensor, numpy ndarray or pandas DataFrame/Series, default is None

val_label
supervised labels for val_data, must be torch tensor, numpy ndarray or pandas DataFrame/Series, default is None

validation_ratio
ratio of 'data' that will be used for validation during training. It will be used only when val_data or val_label or both are None. Default is 0.15

evaluate
bool, whether to evaluate model performance after training ends or not. evaluate performance if set True. Default is True.

figsize
tuple of (width, height) of the figure, size of the figure for loss curve plot and evaluation plots. Default is (18, 4)

train_loader
torch.utils.data.DataLaoder instance for handling training data set. Default is None

val_loader
torch.utils.data.DataLaoder instance for handling validation data set. Default is None

Outputs:

self
torchModel object, returns the self object variable just like scikit-learn model objects

`predict`(*data*, *return_label=False*)

A wrapper function that generates prediction from pytorch model

Inputs:

data
input data to predict on, can be a torch tensor, numpy ndarray, pandas DataFrame or even torch DataLoader object

return_label
bool, whether to return label data if 'data' is a pytorch DataLoader object. Default is False

Outputs:

preds
torch Tensor, predicted values against the inserted data

labels
torch Tensor, only found if 'data' is pytorch DataLoader and return_label is True.

`prepare_data_loader`(*data*, *label*, *val_data*, *val_label*, *validation_ratio*, *train_loader*, *val_loader*)

This function prepares the data format and creates training and validation data loader from training and validation data. If no validation data is provided, it uses validation_ratio to split training data into train and validation and creates separate data loader for each set. If direct train and validation data loaders are provided, then no additional processing is used.

Inputs:

train_data
input training data, must be torch tensor, numpy ndarray or pandas DataFrame/Series. Default value in None

train_label
supervised labels for train_data, must be torch tensor, numpy ndarray or pandas DataFrame/Series. Default value in None

val_data
validation data, must be torch tensor, numpy ndarray or pandas DataFrame/Series, default is None. Note: If val_data is not provided, validation data will be taken from training data set using validation_ratio.

val_label
supervised labels for val_data, must be torch tensor, numpy ndarray or pandas DataFrame/Series, default is None

validation_ratio
ratio of 'data' that will be used for validation during training. It will be used only when val_data or val_label or both are None. Default is 0.15

train_loader
torch.utils.data.DataLaoder instance for handling training data set or None. Default is None. Note: If train_loader is provided, its expected that val_loader will also be provided.

val_loader
torch.utils.data.DataLaoder instance for handling validation data set or None. Default is None

Outputs:

train_loader
torch.utils.data.DataLaoder instance for handling training data set.

val_loader
torch.utils.data.DataLaoder instance for handling validation data set.

relieve_parallel()

This function reverts the DataParallel model to simple torch.nn.Module model

set_parallel()

This method sets multiple GPUs to use for training depending on the machine's cuda availability and use_gpu parameter.

1.7 mlutils.utils

This is a sub-module containing utility functions for multiple model training including scikit-learn models. It also contains several other utility for model storing and loading, Neural Network model building, custom DataSet for pytorch etc. Author : Abdullah Al Masud

email : abdullahalmasud.buet@gmail.com

LICENSE : MIT License

class msdlib.mlutils.utils.**DataSet**(*data*, *label=None*, *dtype=torch.float32*, *model_type='regressor'*)

Bases: Dataset

This is a customized Data set object which can build a torch.utils.data.Dataset object given the data and labels. This is only usable when we have complete data and labels (data and label lengths must be equal)

Inputs:

data
ideally should be torch tensor. Contains feature data tor model training. Can be python list or python set too but not much appreciated as it will be mostly used for training pytorch model.

label
ideally should be numpy ndarray, pandas Series/DataFrame or torch tensor. Contains true labels tor model training. Can be python list or python set too but not much appreciated as it will be mostly used for training pytorch model.

dtype
data type of the data and labels. Default is torch.float32. It also depends on model_type parameter for label data.

model_type
{'regressor', 'binary-classifier' or 'multi-classifier'}. Default is 'multi-classifier'. It is used to confirm that for multi-class classification, label data type is torch.long.

msdlib.mlutils.utils.**define_layers**(*input_units*, *output_units*, *unit_factors*, *dropout_rate=None*, *model_type='regressor'*, *actual_units=False*, *apply_bn=False*, *activation=None*, *final_activation=None*)

This function takes a common formation/sequence of functions to construct one hidden layer and then replicates this sequence for multiple hidden layers. Hidden layer units are decided based on 'unit_factors' paramter. The common formation of one hidden layer is this-

-Linear -BatchNorm1d -ReLU -Dropout

Dropout ratio is same in all layers

Finally the output layer is constructed depending on 'output_units' and 'model_type' parameter including final activation function. Output activation function is provided

Inputs:

input_units
int, number of units in input layer / number of features (not first hidden layer)

output_units
int, number of units in output layer / number of output nodes / number of classes (not last hidden layer)

unit_factors
array of ints or floats, multipliers to calculate number of units in each hidden layer from input_units, or actual number of units for each hidden layer

dropout_rate
dropout ratio, must be 0 ~ 1. Default is None (no dropout layer)

model_type
{'binary-classifier', 'multi-classifier, 'regressor'}, controls use of softmax/sigmoid at the output layer. Use 'regressor' if you dont intend to use any activation at output. Default is 'regressor'

actual_units
bool, whether actual units are placed in unit_factors or not, default is False (not actual units, instead unit_factos is containing ratios)

apply_bn
bool, whether to use batch normalization or not, default is False (does not use batch normalization)

activation
nn.Module object or None. Pytorch activation layer that will be used as activation function after each hidden layer. Default is None (No activation)

final_activation
torch.sigmoid / torch.Softmax(dim=1) / torch.tanh etc. for output layer, default is None. If None, the final activation will be below: - modey_type == 'regressor' –> No activation - model_type == 'binary-classifier' –> torch.sigmoid - model_type == 'multi-clussifier' –> torch.Softmax(dim=1)

Outputs:

layers
list of Deep Learning model layers which can be fed as NNModel layer_funcs input or torchModel layers input to build DNN model

msdlib.mlutils.utils.**evaluate_with_data**(*outdata*, *models*, *figure_dir=None*, *model_type='multi-classifier'*, *figsize=(15, 5)*, *xrot=0*)

This function will be used to train models. We can use both scikit-models and torchModel objects for model training.

Inputs:

outdata
dict, contains dicts with structure like : outdata = {'train': {'data': <numpy-array>, 'label': <numpy-array>, 'index': <numpy-array>}} 'train' dict is mandatory. 'validation' dict is mandatory for torchModel objects inside "models" argument. 'data', 'label' and 'index' must be of same length.

models
dict, contains scikit-models, xgboost, lightgbm etc. scikit-like models and torchModel objects. If its a torchModel object, the key must contain 'pytorch' in it.

figure_dir
string/None, path to the directory where figures will be saved for feature improtances and evaluation figures.

model_type
string, can be either 'regressor', 'multi-classifier' or 'binary-classifier'. It controls the evaluation process.

figsize
tuple of horiaontal and vertical size of the result figure (regression score and classification score both). Default is (15, 5).

xrot
float, rotation angle of the x axis labels in classification score matrix (where class label names are written). Default is 0 (no rotation).

Outputs:

predictions
dict, contains detailed predictions on all sets in outdata argument, evaluation results etc.

`msdlib.mlutils.utils.`**`get_factors`**(*n_layers*, *base_factor=5*, *max_factor=10*, *offset_factor=2*)

This function calculates factors/multipliers to calculate number of units inside define_layers() function

Inputs:

n_layers
number of hidden layers

max_factor
multiplier for mid layer (largest layer)

base_factor
multiplier for first layer

offset_factor
makes assymetric structure in output with factor (base - offset). For symmetric model (size in first and last hidden layer is same), offset will be 0.

Outputs:

factors
multipliers to calculate number of units in each layers based on input shape

`msdlib.mlutils.utils.`**`load_models`**(*models*, *folder_path*)

This function loads different scikit-learn models from .pickle format and Pytorch model from .pt formatted state_dict (only weights)

Inputs:

models
dict, containing model classes or None (for torch model, torch.nn.Module object is necessary as trained model to load the state variables. For other types of models like xgboost etc. None is fine.); For pytorch models, the key must contain 'pytorch' phrase (Case insensitive) key name must be like this :

stored model file name: xgboost_model.pickle

corresponding key for the dict: 'xgboost'

stored model file name: pytorch-1_model.pickle

corresponding key for the dict: 'pytorch-1'

for pytorch model, the model must not be a DataParallel model. You can add parallelism after loading the weights

folder_path
str, directory path from where the stored models will be loaded

Outputs:

models
dict, containing model classes like {<model_name> : <model_class>}

msdlib.mlutils.utils.**store_models**(*models*, *folder_path*)

This function stores different types of scikit-learn models in .pickle format and also Pytorch model in .pt format

Inputs:

models
dict, containing only trained model class; {<model name>: <model class>} For pytorch models, the key must contain 'pytorch' phrase (Case insensitive).

Note: Pytorch model must not be a DataParallel model.

If its a DataParallel model, then take module attribute of your model like this- {'pytorch': <your_model>.module}

folder_path
str, the folder path where the models will be stores, if doesnt exist, it will be created

msdlib.mlutils.utils.**train_with_data**(*outdata*, *feature_columns*, *models*, *featimp_models=[]*, *figure_dir=None*, *model_type='multi-classifier'*, *evaluate=True*, *featimp_figsize=(30, 5)*, *figsize=(15, 5)*, *xrot=0*)

This function will be used to train models. We can use both scikit-models and torchModel objects for model training.

Inputs:

outdata
dict, contains dicts with structure like : outdata = {'train': {'data': <numpy-array>, 'label': <numpy-array>, 'index': <numpy-array>}} 'train' dict is mandatory. 'validation' dict is mandatory for torchModel objects inside "models" argument. 'data', 'label' and 'index' must be of same length.

feature_columns
list/numpy array, contains feature names of 'data' inside outdata. seature_columns length must be equal to number of columns in 'data'

models
dict, contains scikit-models, xgboost, lightgbm etc. scikit-like models and torchModel objects. If its a torchModel object, the key must contain 'pytorch' in it.

featimp_models
list of strings, contains model names which belong to models dict keys and which can provide feature importances

figure_dir
string/None, path to the directory where figures will be saved for feature improtances and evaluation figures.

model_type
string, can be either 'regressor', 'multi-classifier' or 'binary-classifier'. It controls the evaluation process.

evaluate
boolean, If True, model evaluation will be executed and results of evaluation will be stored inside figure_dir. Default is True.

featimp_figsize
tuple of horiaontal and vertical size of the feature importance plot. Default is (30, 5).

figsize
tuple of horiaontal and vertical size of the result figure (regression score and classification score both). Default is (15, 5). Only effective when evaluate is True.

xrot
float, rotation angle of the x axis labels in classification score matrix (where class label names are written). Default is 0 (no rotation). Only effective when evaluate is True.

Outputs:

models
dict, contains trained models instances, same as 'models' argument

predictions
dict, contains detailed predictions on all sets in outdata argument, evaluation results etc.

1.8 dataset

Author : Abdullah Al Masud

email : abdullahalmasud.buet@gmail.com

LICENSE : MIT License

`msdlib.dataset.`**`solenoid_data`**(*data_len=10000*, *ys=3*, *delay=0.3*, *z_spread=[0, 50]*)

This function generates 3d data to form solenoidal shape

Inputs:

data_len
int, number of points, default is 10000

ys
int, number of solenoids, default is 3

delay
float, phase shift parameter for cosine function, default is 0.3

z_spread
tuple/array of length 2, containing z-axis starting and ending values stretching all data points. Default is [0, 50]

Outputs:

data_source
pandas DataFrame, contains 'x', 'z' and ys number of y columns (default is 3 for ys, so in total 5 columns to represent 3 solenoidal curves)

1.9 msdbacktest

msdbacktest intends to support for fintech. Its still under development. It plans to provide-

- back-testing program
- strategy implementation
- evaluating strategy performance using different measures (drawdown, calmar, sharpe etc.)
- some miscellaneous supportive functions

Author : Abdullah Al Masud

email : abdullahalmasud.buet@gmail.com

msdlib.msdbacktest.**get_annual_return**(*sr*)

This function calculates annual rate of return

Inputs:

sr
pandas Series, balance curve

Outputs:

annual_return
float, rate of return value

msdlib.msdbacktest.**get_balance**(*ts_index*, *closings*, *starting_cap*, *sample_per='D'*)

This function calcualtes total available balance for a portfolio.

Inputs:

ts_index
index of time series pandas data

closings
pandas dataframe, having columns 'trigger_point', 'rated_return'

'trigger_point' column contains time stamp when there is buy or sell signal found

'rated_return' column contains gain(+) or loss(-) values corresponding to each trigger_point

starting_cap
float, starting capital value

sample_per
Literals like 'D', 'M', 'H' etc., resampling period, default is 'D'

Outputs:

balance: pandas Series, containing balance value for a portfolio or one symbol for each time instance (balance curve)

msdlib.msdbacktest.**get_calmar**(*sr*)

This function calculates calmar ratio value for a balance curve

Inputs:

sr
pandas Series, balance time series data (balance curve)

Outputs:

calmar
float, calmar ratio value

msdlib.msdbacktest.**get_crossover**(*sr_slow*, *sr_fast*)

This functions calculates crossover points between a faster time series (sr_fast) and a bit slower version of the same time series data.

Inputs:

sr_slow
pandas Series, slow time series data (rolling average with bigger window size of a time series data)

sr_fast
pandas Series, faster time series data (rolling average with bigger smaller size of a time series data)

Outputs:

cross_sig
pandas Series, containing crossover points 1 (fast crosses slow signal from down to up) or -1 (slow crosses fast signal from up to down)

msdlib.msdbacktest.**get_drawdown**(*sr*)

This function calculates drawdown value from a balance curve

Inputs:

sr
padnas Series, time series balance at each time instance (balance curve)

Outputs:

dds
pandas DataFrame, contains these columns ['start_point', 'start_value', 'low_point', 'low_value', 'end_point', 'end_value', 'duration']

msdlib.msdbacktest.**load_data**(*root_path*, *file*, *start=None*, *stop=None*, *usables=['open', 'high', 'low', 'close', 'volume']*)

This function loads csv or txt formatted data.

Inputs:

root_path
path to the directory where the csv or txt file is kept

file
file name of the csv or txt file with extension

start
starting time for the data, default is None

stop
ending time for the data, default is None

usables
column names in lower case letter which will be used from csv or txt data. Default columns are [open, high, low, close, volume]

Outputs:

data
loaded data

msdlib.msdbacktest.**next_dates**(*datetimes*, *dates*)

gets next date of a particular series data from pandas series index.

Inputs:

datetimes
list of datetimes, to find the next dates of

dates
list of indices of true time series data

Outputs:

nextdates
list of next dates corresponding to datetimes

msdlib.msdbacktest.**prev_dates**(*datetimes*, *dates*)

gets predious date of a particular series data from pandas series index.

Inputs:

datetimes
list of datetimes, to find the previous dates of

dates
list of indices of true time series data

Outputs:

prevdates
list of next dates corresponding to datetimes

1.10 stats

This module is intended to use for statistical computations such as calculating likelihoods and probabilities etc. related to data science and machine learning. Author : Abdullah Al Masud

email : abdullahalmasud.buet@gmail.com

LICENSE : MIT License

msdlib.stats.**likelihood_for_categorics**(*x*, *uniques=None*)

This function calculates likelihood for variable x for each categorical value in it.

Here, x is Discrete Random Variable. For example, we get x from Titanic Dataset for gender variable if we take gender values of Class Survived=1 (so, subset of all gender values). So, this function will provide likelihood of Class Survived=1 (people who survived) for gender variable.

If x doesnt contain all possible values of this variable, then we must provide all possible values in uniques variable.

Inputs:

x
Pandas Series or numpy 1-D array, contains discrete random variable for a particular class.

uniques
list, pandas Series or numpy array, containing all possible values of broader/parent set of x. Default if None which assumes that x contains all possible values of parent set.

Outputs:

probs
pandas Series, it contains likelihood estimations of x for each discrete value. The indices of probs show discrete values of x.

msdlib.stats.**likelihood_for_numerics**(*x*, *_min=None*, *_max=None*, *bins=100*)

This function converts continues distribution into discrete distribution and calculates likelihood for each distribution bin.

Here, x is Continuous Random Variable. For example, we get x from Iris Dataset for petal length variable if we take petal lengths of Class 'Setosa'. So, this function will provide likelihood of 'Setosa' for Petal length.

If the minimum and maximum values of complete distrubition (all petal lengths) is out of bound of the distribution of x, then _min and _max must be provided.

Inputs:

x
Pandas Series or numpy 1-D array, contains random variable for a particular class.

_min
float, minimum value of the whole distribution where all classes are included. Default is None which assumes the minimum value is minimum of x.

_max
float, maximum value of the whole distribution where all classes are included. Default is None which assumes the maximum value is maximum of x.

bins
int, number of distribution bins, used to convert continuous distribution into discrete distribution. Default is 100.

Outputs:

probs
pandas Series, it contains likelihood estimations of x for each bin. The indices of probs show the upper range/cut of each bin.

1.11 msdExceptions

It contains some custom made exception classes. Author : Abdullah Al Masud

email : abdullahalmasud.buet@gmail.com

LICENSE : MIT License

exception msdlib.msdExceptions.**CutoffError**

Bases: Exception

exception msdlib.msdExceptions.**DataLabelMismatchError**

Bases: Exception

exception msdlib.msdExceptions.**FilterTypeError**

Bases: Exception

exception msdlib.msdExceptions.**InputVariableError**

Bases: Exception

exception msdlib.msdExceptions.**InvalidBins**

Bases: Exception

exception msdlib.msdExceptions.**SplitMethodError**

Bases: Exception

exception msdlib.msdExceptions.**WordLengthError**

Bases: Exception

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

INDEX

G

I

L

M

N

O

P

R

S

T

V

W