RealSeries

Release 0.0.1

Wenbo Hu

Dec 07, 2021

GETTING STARTED

1 Installation **3**

2 Examples **5**
2.1 IsolationForest Example . . . 5
2.2 Pytorch basded Neural Network Example . . . 8
2.3 Series and Label Visualize Example . . . 11
2.4 Granger causality Example . . . 13
2.5 VAE for anomaly detection Example . . . 19

3 API Cheetsheet **23**
3.1 Model . . . 23
3.2 Data . . . 24
3.3 Visualize . . . 24

4 API Reference **25**
4.1 All Models . . . 25
4.2 Utility Functions . . . 49

5 Time Series Datasets **63**
5.1 Forecast Datasets . . . 63
5.2 Anomaly Detection Datasets . . . 65
5.3 Granger causality Datasets . . . 66

6 Anomaly Detection **67**
6.1 Problem Description . . . 67
6.2 Models . . . 68

7 Granger causality **69**
7.1 Problem Description of Granger causality . . . 69
7.2 Models of Granger causality . . . 70

8 Forecast with Uncertainty **71**
8.1 Problem Description . . . 71
8.2 Models . . . 72

9 Contribution **75**

10 How to Contribute **77**

11 Indices and tables **79**

Bibliography **81**

Python Module Index **83**

Index **85**

RealSeries is a comprehensive **out-of-the-box Python toolkit** for various tasks, including *Anomaly Detection*, *Granger causality* and *Forecast with Uncertainty*, of dealing with *Time Series Datasets*.

RealSeries has the following features:

- Unified APIs, detailed documentation, easy-to-follow examples and straightforward visualizations.
- All-levels of models, including simple thresholds, classification-based models, and deep (Bayesian) models.

Warning: RealSeries supports **Python 3 ONLY**. See here to check why.

API Demo:

RealSeries uses the sklearn-style API and is as easy as

```
# train the SR-CNN detector
from realseries.models.sr_2 import SR_CNN
sr_cnn = SR_CNN(model_path)
sr_cnn.fit(train_data)
score = sr_cnn.detect(test_data, test_label)
```

CHAPTER

ONE

INSTALLATION

RealSeries is still under development. Before the first stable release, you can install the RealSeries from source.

If you use RealSeries temporarily, you may clone the code from GitHub repository and add it to your Python path:

```
git clone https://github.com/xchuwenbo/realseries.git
cd RealSeries # change current work directory to ./RealSeries
python
>>> import sys,os
>>> sys.path.append(os.getcwd()) # add ./RealSeries to sys.path
>>> import realseries
>>> from realseries.models.iforest import IForest
```

Alternatively, you can install it:

```
git clone https://github.com/xchuwenbo/realseries.git # clone
cd RealSeries
pip install .
python
>>> import realseries
>>> from realseries.models.iforest import IForest
```

CHAPTER

TWO

EXAMPLES

2.1 IsolationForest Example

Full example: *notebooks/lumino_rrcf.ipynb*

1. Import models

```
import numpy as np
from realseries.models.iforest import IForest  # IsolationForest detector
from realseries.utils.evaluation import point_metrics, adjust_predicts
from realseries.utils.data import load_NAB
from realseries.utils.visualize import plot_anom
```

2. Generate sample data with `realseries.utils.data.load_NAB()`:

```
dirname = 'realKnownCause'
filename = 'nyc_taxi.csv'

# the fraction of used for test
fraction=0.5

train_set, test_set = load_NAB(dirname, filename, fraction=fraction)

# the last column is label; other columns are values
train_data,train_label = train_set.iloc[:, :-1],train_set.iloc[:, -1]
test_data,test_label = test_set.iloc[:, :-1],test_set.iloc[:, -1]

# visualize
test_data.plot()
```

3. Initialize a `realseries.models.iforest.IForest` detector, fit the model, and make the prediction.

```
# train the isolation  forest model
# number of trees
n_estimators=1000

# number of samples from the input array X for one estimator
max_samples="auto"

# the fraction of anomaly point in the total input sequence
contamination=0.01
```

(continues on next page)

(continued from previous page)

```
#random seed
random_state=0

#build model
IF = IForest(n_estimators=n_estimators,
            max_samples=max_samples,
            contamination=contamination,
            random_state=random_state)

#train model
IF.fit(train_data)

# detect
score = IF.detect(test_data)
```

4. Get anomaly label by setting threshold.

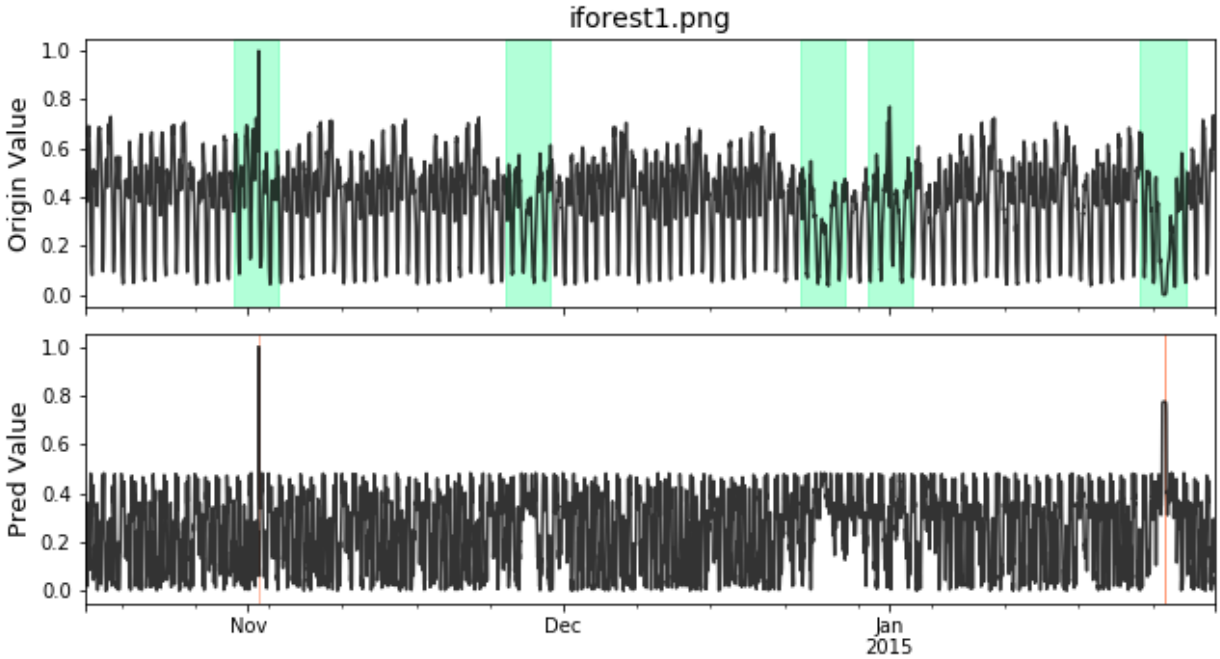
```
thres_percent=99.9
thres = np.percentile(score,thres_percent)
pred_label = (score>thres)
```

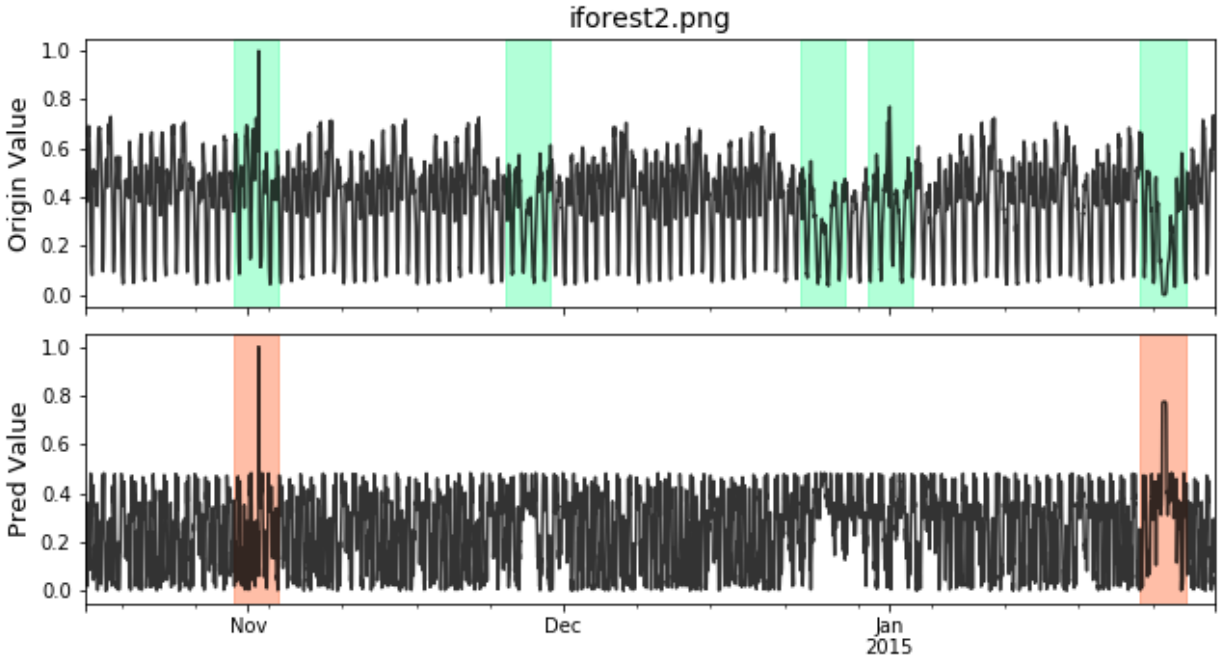
5. Visualize and Evaluate the prediction result point to point.

```
# visualize
plot_anom(
    test_set,
    pred_label,
    score)

# evaluate and print the results
precision, recall, f1, tp, tn, fp, fn = point_metrics(pred_label,
                                                      test_label)
print('precision:{}, recall:{}, f1:{}, tp:{}, tn:{}, fp:{}, fn:{}'.format(
                                precision, recall, f1, tp, tn, fp, fn))
```

6. Visualize the prediction that is adjusted. Evaluate the adjusted results.

```
# evaluate and print the results
delay = 200  # delay is the max number of delay points that allowed
             # when anomaly point occur.

adjust_pred_label = adjust_predicts(pred_label,test_label,delay=200)
plot_anom(
    test_set,
    adjust_pred_label,
    score)

precision, recall, f1, tp, tn, fp, fn = point_metrics(adjust_pred_label,
                                                      test_label)
print('precision:{}, recall:{}, f1:{}, tp:{}, tn:{}, fp:{}, fn:{}'.format(
                                precision, recall, f1, tp, tn, fp, fn))
```


iforest1.png
Origin Value
Pred Value
1.0
0.8
0.6
0.4
0.2
0.0
Nov
Dec
Jan
2015


iforest2.png
Origin Value
Pred Value
1.0
0.8
0.6
0.4
0.2
0.0
Nov
Dec
Jan
2015

2.2 Pytorch basded Neural Network Example

Full example: *notebooks/lstm_dym.ipynb*

1. Import models

```
import os
import numpy as np
from pathlib import Path
from realseries.models.lstm_dynamic import LSTM_dynamic
from realseries.utils.evaluation import point_metrics, adjust_predicts
from realseries.utils.data import load_NAB
from realseries.utils.visualize import plot_anom
os.environ["CUDA_VISIBLE_DEVICES"] = '0'  # set visible gpu
```

2. Generate sample data with *realseries.utils.data.load_NAB()*:

```
dirname = 'realKnownCause'
filename = 'nyc_taxi.csv'

# the fraction of used for test
fraction=0.5

train_set, test_set = load_NAB(dirname, filename, fraction=fraction)

# the last column is label; other columns are values
train_data,train_label = train_set.iloc[:, :-1],train_set.iloc[:, -1]
test_data,test_label = test_set.iloc[:, :-1],test_set.iloc[:, -1]

# visualize
test_data.plot()

from realseries.utils.preprocess import normalization
train_data,test_data = normalization(train_data),normalization(test_data)
```

3. Initialize parameters.

```
# LSTM parameters
# --------------------------
dropout = 0.3
lstm_batch_size = 64
hidden_size = 128
num_layers = 2
lr = 1e-3
epochs = 40

# data parameters
# --------------------------
# time_window length of input data
l_s = 50

# number of values to predict by input data
n_predictions =  5
```

(continues on next page)

(continued from previous page)

```
# error parameters
# -------------------------
# number of values to evaluate in each batch in the prediction stage
batch_size = 100

# window_size to use in error calculation
window_size = 30

# determines window size used in EWMA smoothing (percentage of total values␣
↪for channel)
smoothing_perc = 0.05

# number of values surrounding an error that are brought into the sequence␣
↪(promotes grouping on nearby sequences
error_buffer = 20

# minimum percent decrease between max errors in anomalous sequences (used␣
↪for pruning)
p = 0.13
```

4. Initialize a *realseries.models.lstm_dynamic.LSTM_dynamic* detector, fit the model, and make the prediction.

```
# build the model
# -------------------------
# path to save model in Realseries/snapshot/.....
model_path = Path('',f'../snapshot/lstm_dym/{filename[:-4]}')

# init the model class
lstm_dym = LSTM_dynamic(
    batch_size=batch_size,
    window_size=window_size,
    smoothing_perc=smoothing_perc,
    error_buffer=error_buffer,
    dropout=dropout,
    lstm_batch_size=lstm_batch_size,
    epochs=epochs,
    num_layers=num_layers,
    l_s=l_s,
    n_predictions=n_predictions,
    p=p,
    model_path=model_path,
    hidden_size=hidden_size,
    lr=lr)

lstm_dym.fit(train_data)

# detect
anomaly_list, score_list = lstm_dym.detect(test_data)

# create anomaly score array for ploting and evaluation
pred_label = np.zeros(len(test_label))
```

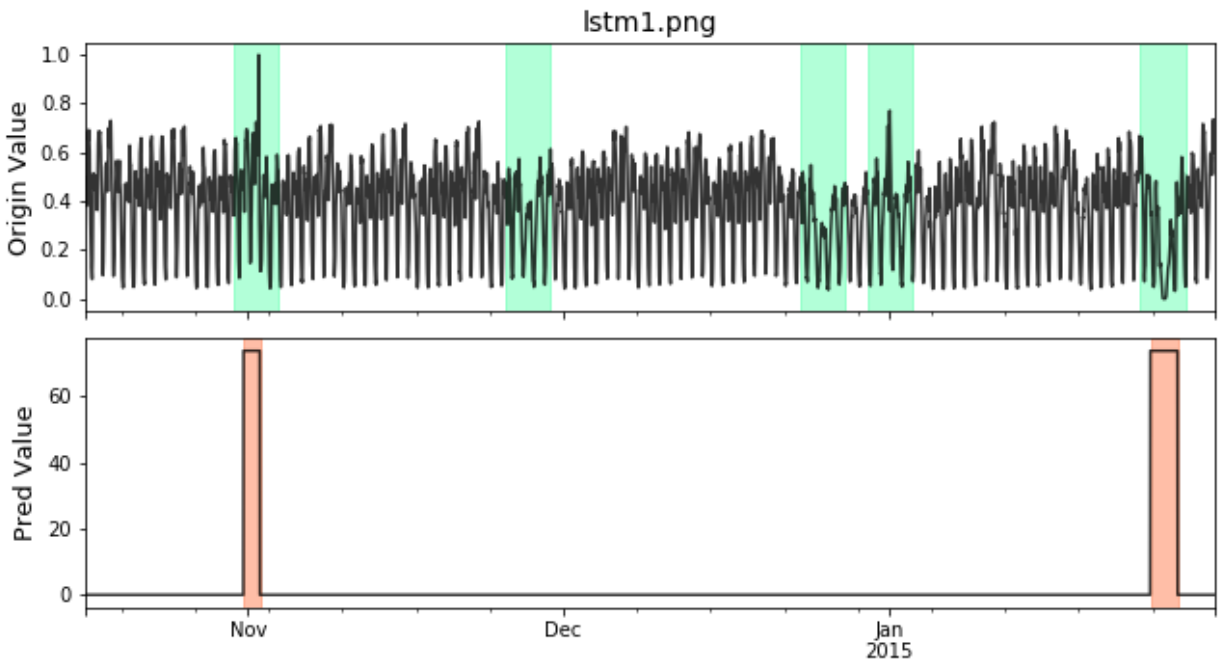
(continues on next page)

(continued from previous page)

```
score = np.zeros(len(test_label))
for (l, r), score_ in zip(anomaly_list, score_list):
    pred_label[l:r] = 1
    score[l:r] = score_
```

5. Visualize and Evaluate the prediction result point to point.

```
# visualize
plot_anom(
    test_set,
    pred_label,
    score)

# evaluate and print the results
precision, recall, f1, tp, tn, fp, fn = point_metrics(pred_label,
                                                      test_label)
print('precision:{}, recall:{}, f1:{}, tp:{}, tn:{}, fp:{}, fn:{}'.format(
                                precision, recall, f1, tp, tn, fp, fn))
```



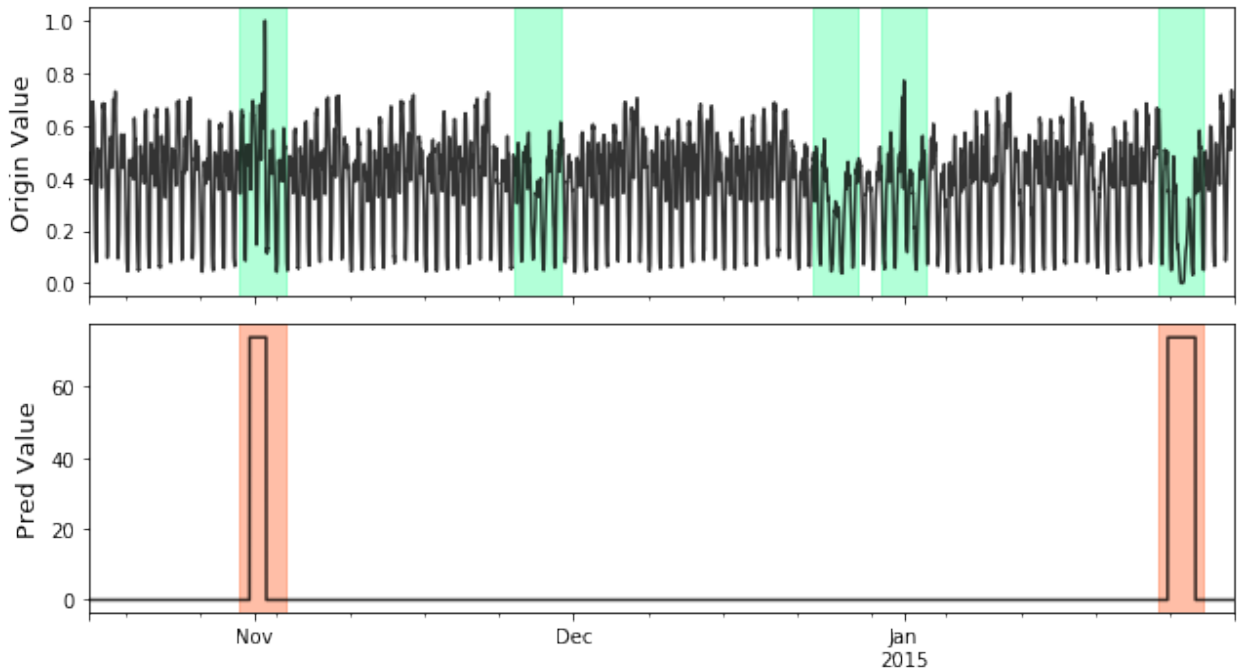
6. Visualize the prediction that is adjusted. Evaluate the adjusted results.

```
# evaluate and print the results
delay = 200  # delay is the max number of delay points that allowed
             # when anomaly point occur.

adjust_pred_label = adjust_predicts(pred_label,test_label,delay=200)
plot_anom(
    test_set,
    adjust_pred_label,
```

(continues on next page)

(continued from previous page)

```
    score)

precision, recall, f1, tp, tn, fp, fn = point_metrics(adjust_pred_label,
                                                      test_label)
print('precision:{}, recall:{}, f1:{}, tp:{}, tn:{}, fp:{}, fn:{}'.format(
                                  precision, recall, f1, tp, tn, fp, fn))
```



2.3 Series and Label Visualize Example

1. Import models

```
import numpy as np
from realseries.utils.data import load_NAB
from realseries.utils.visualize import plot_anom,plot_mne
```

2. Generate sample data with *realseries.utils.data.load_NAB()*:

```
dirname = 'realKnownCause'
filename = 'nyc_taxi.csv'

# the fraction of used for test
fraction=0.5

train_set, test_set = load_NAB(dirname, filename, fraction=fraction)

# the last column is label; other columns are values
```

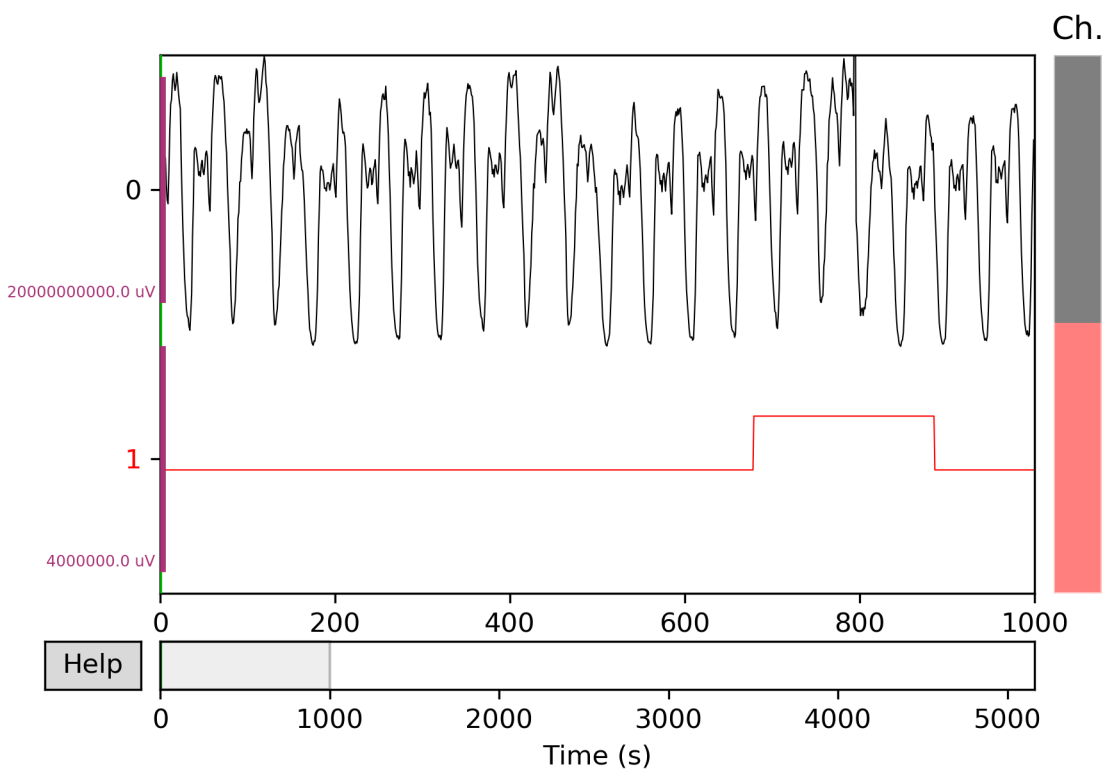
(continues on next page)

(continued from previous page)

```
train_data,train_label = train_set.iloc[:, :-1],train_set.iloc[:, -1]
test_data,test_label = test_set.iloc[:, :-1],test_set.iloc[:, -1]
```

3. Mne based visualize. The dataset `test_set` contains signals channel and label channel. The label is in the last channel. In order to make data and label channel shown in different colors, we set two kinds channle type by `ch_types=['eeg']*(num_chans-1) + ['ecg']`.The last channel `ecg` is different with others `eeg`. We also assign different colors as `eeg='k', ecg='r'`. e.g.

```
# the last column of test_set is label
num_chans = test_set.shape[1]

# modify scales according the shown figure, it can also
# be set to scalings='auto'
scalings = {'eeg': 1e4, 'ecg': 2}

# assign colors for different channel types.
color=dict(eeg='k', ecg='r')

# the last channle is ecg and others are eeg channle
ch_types=['eeg']*(num_chans-1) + ['ecg']

plot_mne(test_set,
    scalings=scalings,
    ch_types=ch_types,
    color=color)
```



More details in *`realseries.utils.visualize.plot_mne()`*.

2.4 Granger causality Example

Full example: *notebooks/DWGC.ipynb*, *notebooks/GC.ipynb*

2.4.1 Industrial demo

1. Import models

```
import sys,os


import realseries
import matplotlib as plt
import numpy as np
realseries.__file__


from realseries.models.base import BaseModel
from realseries.models.NAR import NAR_Network
from realseries.models.AR_new import AR_new
from realseries.models.DWGC import DWGC
from realseries.models.GC import GC

from matplotlib import pyplot as plt
from statsmodels.tsa.ar_model import AR
from sklearn.metrics import mean_squared_error
import pandas as pd


import scipy.special

import matplotlib.pyplot as plt
import numpy as np
from datetime import datetime
import matplotlib.dates as mdates

from scipy import stats
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import pandas as pd
import math
import numpy as np
```

2. Import data and DWGC algorithm, the F_test result shows the window level causality.

```
model = NAR_Network(20,10,1,0.9)
tempt1 = DWGC(1,model,0.8,'NAR',2,1)
tempt2 = GC(1,model,'NAR',1)
data = pd.read_csv("ensodata.csv",encoding = "ISO-8859-1", engine='python')
data = data.values
```

3. Visualize the window-level causality. We find that DWGC method is more consistent with two prior conclusions

than traditional GC method: 1. The causal relationship of ENSO to MKE/OLR is more obvious in autumn/winter than in spring/summer; 2 the causal relationship of MKE/OLR to ENSO exists in spring/summer.

```
fig = plt.figure()
fig = plt.figure(figsize=(10,8))

ax1 = fig.add_subplot(411)
tempt1.fit([data[0:50,0],data[0:50,2]])
F_test_win1 = tempt1.detect([data[0:50,0],data[0:50,2]])
F_test_win2 = tempt2.detect([data[0:50,0],data[0:50,2]])
x = np.linspace(4, 12,30)
l1,=ax1.plot(x,F_test_win1)
l2,=plt.plot((F_test_win2))
plt.xlim(4,12)
ax1.legend(handles=[l1,l2], labels=[r'DWGC ',r'GC'],  loc='upper left',␣
↪fontsize=14)
plt.title('causality ENSO to OLR',fontsize = 14)
plt.xlabel('month',fontsize = 14)
plt.ylabel(r'$F_{statistic}$',fontsize = 14)
ax1.yaxis.get_major_formatter().set_powerlimits((0,1))



ax2 = fig.add_subplot(412)
F_test_win1 = tempt1.detect([data[0:50,1],data[0:50,2]])
F_test_win2 = tempt2.detect([data[0:50,1],data[0:50,2]])
l1,=plt.plot(x,F_test_win1)
l2,=plt.plot(F_test_win2)
plt.xlim(4,12)
plt.legend(handles=[l1,l2], labels=[r'DWGC ',r'GC'],  loc='upper left',␣
↪fontsize=14)
plt.xlabel('month',fontsize = 14)
plt.ylabel(r'$F_{statistic}$',fontsize = 14)
ax2.yaxis.get_major_formatter().set_powerlimits((0,1))
plt.title('causality ENSO to MKE',fontsize = 14)



ax3 = fig.add_subplot(413)
F_test_win1 = tempt1.detect([data[0:50,2],data[0:50,0]])
F_test_win2 = tempt2.detect([data[0:50,2],data[0:50,0]])
l1,=plt.plot(x,F_test_win1)
l2,=plt.plot(F_test_win2)
plt.xlim(4,12)
plt.legend(handles=[l1,l2], labels=[r'DWGC ',r'GC'],  loc='upper left',␣
↪fontsize=14)
plt.xlabel('month', fontsize=14)
plt.ylabel(r'$F_{statistic}$', fontsize=14)
ax3.yaxis.get_major_formatter().set_powerlimits((0,1))
plt.title('causality OLR to ENSO', fontsize=14)
fig.tight_layout()
plt.subplots_adjust(wspace =1, hspace =1)
```

(continues on next page)

(continued from previous page)

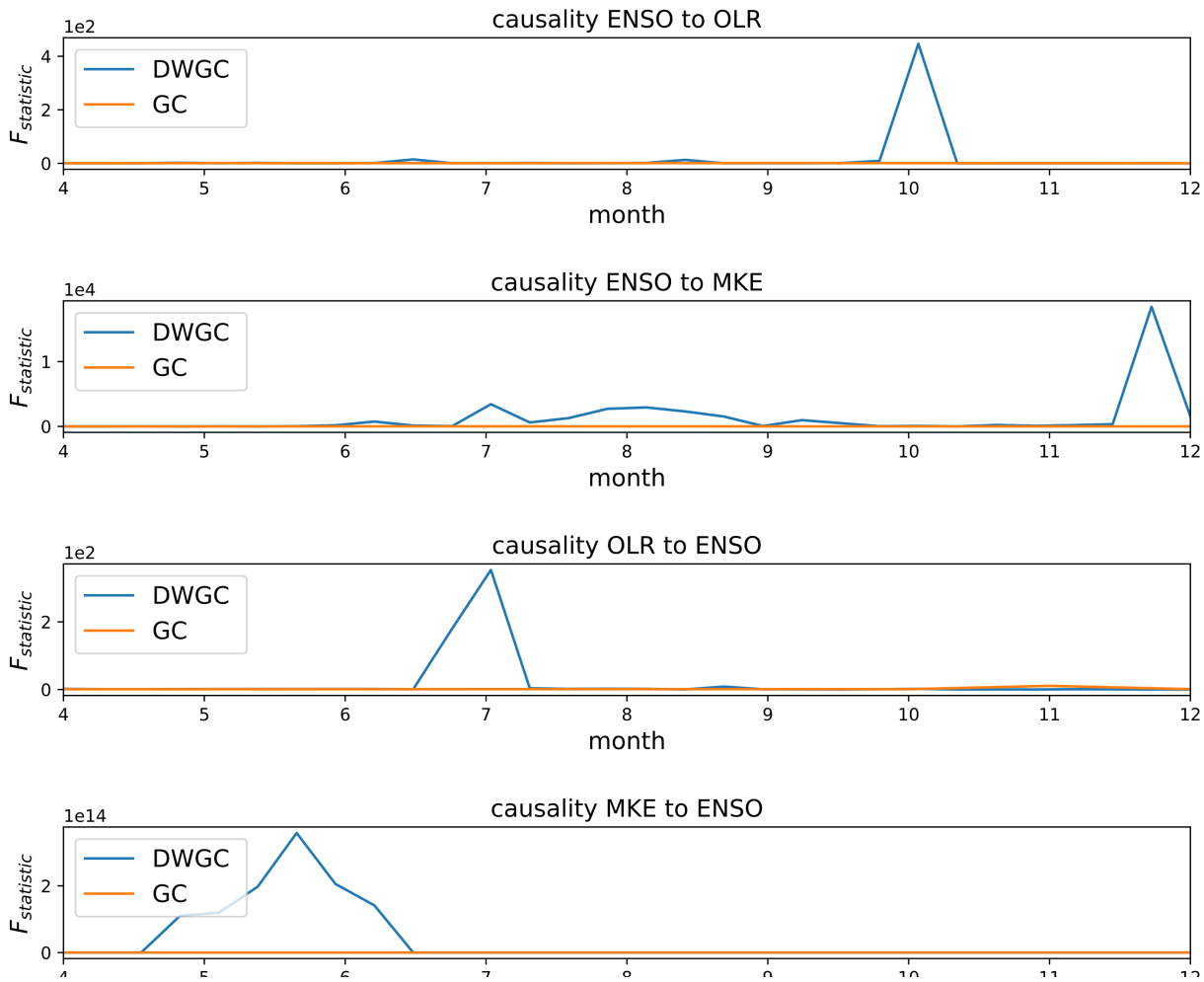
```
ax4 = fig.add_subplot(414)
F_test_win1 = tempt1.detect([data[0:50,2],data[0:50,1]])
F_test_win2 = tempt2.detect([data[0:50,2],data[0:50,1]])
plt.xlim(4,12)
l1,=plt.plot(x,F_test_win1)
l2,=plt.plot(F_test_win2)
plt.legend(handles=[l1,l2], labels=[r'DWGC ',r'GC'],  loc='upper left',␣
↪fontsize=14)
plt.xlabel('month', fontsize=14)
plt.ylabel(r'$F_{statistic}$', fontsize=14)
ax4.yaxis.get_major_formatter().set_powerlimits((0,1))
plt.title('causality MKE to ENSO', fontsize=14)
# Compared with the traditional GC method, DWGC method can better fit two␣
↪prior conclusions:
# 1 The causality from ENSO to MKE/OLR is more obvious in autumn/winter␣
↪than in spring/summer;
# 2 The causality from MKE/OLR to ENSO exists in spring/suummer.

plt.savefig('DWGC(GC)_ENSO.pdf')
```

2.4.2 Simulation demo

1. Preprocessing, import the data and models.

```
import sys,os

import realseries
import matplotlib as plt
import numpy as np
realseries.__file__
from realseries.models import iforest
from realseries.models.base import BaseModel



import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from realseries.models.DWGC import DWGC
from realseries.models.GC import GC

import scipy.special


data = pd.read_csv("AR_causal_point.csv",encoding = "ISO-8859-1", engine=
↪'python')
data = data.values
data = data[0:490,:]
```

(continues on next page)


causality ENSO to OLR
1e2
DWGC
GC
$F_{statistic}$
month
causality ENSO to MKE
1e4
DWGC
GC
$F_{statistic}$
month
causality OLR to ENSO
1e2
DWGC
GC
$F_{statistic}$
month
causality MKE to ENSO
1e14
DWGC
GC
$F_{statistic}$

(continued from previous page)

```
delay_input = 30


data2 = pd.read_csv("AR_series.csv",encoding = "ISO-8859-1", engine='python
↪')
data2 = data2.values
data2 = data2[0:500,:]

data_0 = scipy.special.expit(np.diff(data2[:,0]))
data_1 = scipy.special.expit(np.diff(data2[:,1]))
```

2. Create a class to detect the window-level causality on NAR simulation series.

```
def experiment(win_length):
    stat_count_detect_DWGC = []
    stat_count_detect_GC = []
    #repeat the experiment
    for repeat in range(50):

        model = NAR_Network(delay_input,10,1,0.95)
        tempt = DWGC(win_length,model,0.9,'NAR',2,0.1)
        Ftest_win_DWGC = tempt.detect([data_0,data_1])

        tempt = DWGC(win_length,model,1,'NAR',1,0.1)  #GC is the special␣
↪case of lr=1 in DWGC;


        Ftest_win_GC = tempt.detect([data_0,data_1])



        count_real = 0
        count_detect_DWGC = 0
        count_whole_DWGC = 0
        count_detect_GC = 0
        count_whole_GC = 0


        data_use =  int((len(data_0)-delay_input)/win_length) * win_length



        win_num = int((len(data_0)-delay_input)/win_length)
        label_real = [0] *win_num
        label_detect_DWGC = [0] *win_num
        label_whole_DWGC = [0] *win_num

        label_detect_GC = [0] *win_num
        label_whole_GC = [0] *win_num
```

(continues on next page)

(continued from previous page)

```
        for i in range(data_use-1):
            win_number = int((i)/win_length)
            if data[i,0] != 0.45:


                label_real[win_number] = 1
                if Ftest_win_DWGC[win_number] > 1 or Ftest_win_DWGC[win_
↪number] == 1 :
                    label_detect_DWGC[win_number] = 1
            if Ftest_win_DWGC[win_number]>1 or Ftest_win_DWGC[win_number]␣
↪== 1:

                label_whole_DWGC[win_number] = 1

        if np.mean(np.abs(np.array(Ftest_win_DWGC)-np.array([1]*len(Ftest_
↪win_DWGC)))) > 0.1:

            stat_count_detect_DWGC.append(np.sum(label_detect_DWGC))
        '''
        else:

            stat_count_detect_DWGC.append(0.5 * np.sum(label_real))
        '''


        for i in range(data_use-1):
            win_number = int((i)/win_length)
            if data[i,0] != 0.45:

                if Ftest_win_GC[win_number] > 1 or Ftest_win_DWGC[win_
↪number] == 1:

                    label_detect_GC[win_number] = 1
            if Ftest_win_GC[win_number]>1 or Ftest_win_DWGC[win_number] ==␣
↪1:

                label_whole_GC[win_number] = 1
        if np.mean(np.abs(np.array(Ftest_win_GC)-np.array([1]*len(Ftest_win_
↪GC)))) > 0.1:

            stat_count_detect_GC.append(np.sum(label_detect_GC))
        '''
        else:

            stat_count_detect_GC.append(0.5 * np.sum(label_real))
        '''

        if repeat % 2 ==0:

            print("repeat:",repeat)
            '''
```

(continues on next page)

(continued from previous page)

```
            print("count_real:", np.sum(label_real))
            print("count_detect_DWGC",np.sum(label_detect_DWGC))
            print("count_whole_DWGC",np.sum(label_whole_DWGC))


            print("count_detect_GC",np.sum(label_detect_GC))
            print("count_whole_GC",np.sum(label_whole_GC))
            '''

            print("mean_stat_Ftest_win_DWGC:",np.mean(stat_count_detect_
↪DWGC)/np.sum(label_real))
            print("mean_stat_Ftest_win_GC:",np.mean(stat_count_detect_GC)/
↪np.sum(label_real))
            print("var_stat_Ftest_win_DWGC:",(np.var(stat_count_detect_
↪DWGC))/(np.sum(label_real)*np.sum(label_real)))
            print("var_stat_Ftest_win_GC:",np.var(stat_count_detect_GC)/(np.
↪sum(label_real)*np.sum(label_real)))

    return [np.mean(stat_count_detect_DWGC)/np.sum(label_real),np.mean(stat_
↪count_detect_GC)/np.sum(label_real)]
```

3. Compute the accuracy/recall of DWGC/GC on different window length.

```
experiment(10)
experiment(20)
experiment(30)
experiment(100)
```

datasets		External	GC		DWGC	
			accuracy	recall	accuracy	recall
NAR simulation	window length=10	github	0.42	0.58	0.44	0.73
	window length=20		0.76	0.65	0.80	0.65
	window length=30		0.93	0.66	0.94	0.67
	window length=100		1	0.86	1	0.88

2.5 VAE for anomaly detection Example

Full example: *notebooks/donut_vae.ipynb*

1. Import models

```
import numpy as np
from realseries.models.vae_ad import VAE_AD  # VAE anomaly detector
from realseries.utils.evaluation import point_metrics, adjust_predicts
from realseries.utils.data import load_NAB
from realseries.utils.visualize import plot_anom
```

2. Generate sample data with `realseries.utils.data.load_NAB()` and standardize:

```
dirname = 'realKnownCause'
filename = 'nyc_taxi.csv'

# the fraction of used for test
fraction=0.5

train_data, test_data = load_NAB(dirname, filename, fraction=fraction)
mean_ = train_data['value'].mean()
std_ = train_data['value'].std()
train_data['value'] = train_data['value'].apply(lambda x: (x - mean_) / std_
↪)
test_data['value'] = test_data['value'].apply(lambda x: (x - mean_) / std_)
```

3. Initialize a *realseries.models.vae_ad.VAE_AD* detector, fit the model, and make the prediction.

```
# define the parameters
num_epochs=256
batch_size=256
lr=1e-3
lr_decay=0.8
clip_norm_value=12.0
weight_decay=1e-3
data_split_rate=0.5
window_size=120
window_step=1

# vae network parameters
h_dim=100
z_dim=5

#build model
vae = VAE_AD(name='VAE_AD',
             num_epochs=num_epochs,
             batch_size=batch_size,
             lr=lr,
             lr_decay=lr_decay,
             clip_norm_value=clip_norm_value,
             weight_decay=weight_decay,
             data_split_rate=data_split_rate,
             window_size=window_size,
             window_step=window_step,
             h_dim=h_dim,
             z_dim=z_dim)

#train model
vae.fit(train_data['value'].values)

# detect
res = vae.detect(test_data['value'].values)
ori_series = res['origin_series']
anomaly_score = res['score']
```

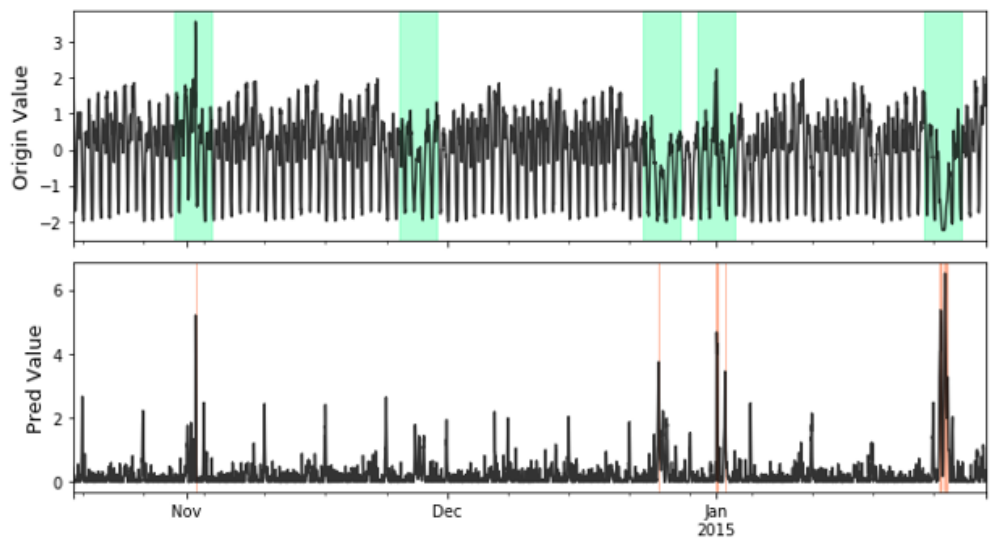
4. Get anomaly label by setting threshold.

```
k = 6
pred_label = (anomaly_score > np.std(anomaly_score) * k)
test_set = test_data[window_size - 1:]
test_label = test_set.iloc[:, -1]
```

5. Visualize and Evaluate the prediction result point to point.

```
# visualize
plot_anom(
    test_set,
    pred_label,
    anomaly_score)

# evaluate and print the results
precision, recall, f1, tp, tn, fp, fn = point_metrics(pred_label,
                                                      test_label)
print('precision:{}, recall:{}, f1:{}, tp:{}, tn:{}, fp:{}, fn:{}'.format(
                                precision, recall, f1, tp, tn, fp, fn))
```

```
precision:0.9999999972972973, recall:0.03574879226707741, f1:0.06902984406866151, tp:37, tn:4006, fp:0, fn:998
```



6. Visualize the prediction that is adjusted. Evaluate the adjusted results.

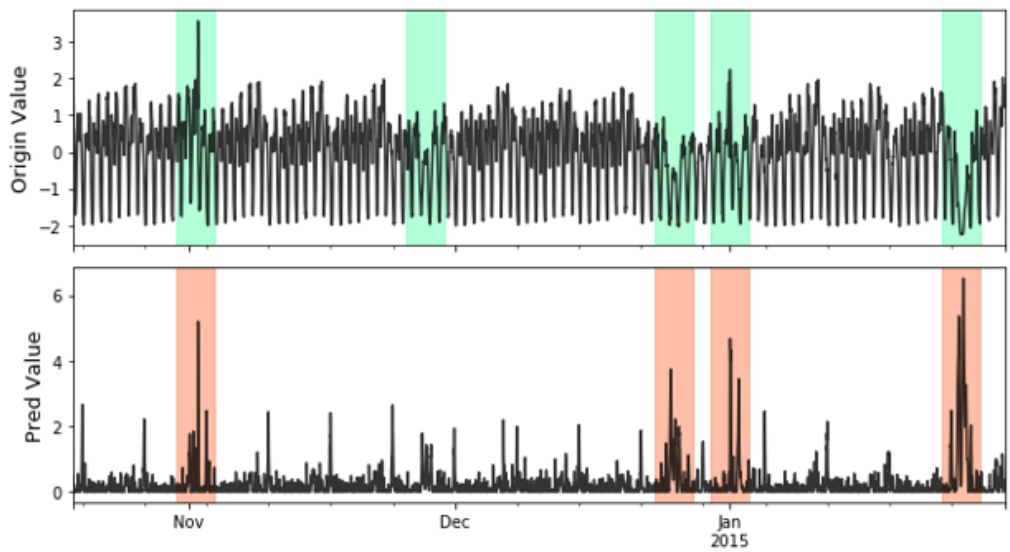
```
# evaluate and print the results
delay = 200  # delay is the max number of delay points that allowed
             # when anomaly point occur.

adjust_pred_label = adjust_predicts(pred_label,test_label,delay=delay)
plot_anom(
    test_set,
    adjust_pred_label,
    anomaly_score)

precision, recall, f1, tp, tn, fp, fn = point_metrics(adjust_pred_label,
```

(continues on next page)

(continued from previous page)

```
                                                        test_label)
print('precision:{}, recall:{}, f1:{}, tp:{}, tn:{}, fp:{}, fn:{}'.format(
                                precision, recall, f1, tp, tn, fp, fn))
```

```
precision:0.999999999879227, recall:0.7999999999227052, f1:0.8888888394107499, tp:828, tn:4006, fp:0, fn:207
```



CHAPTER

THREE

API CHEETSHEET

3.1 Model

- **IsolationForest**
  - *realseries.models.iforest.IForest.fit()*: Fit Isolation Forest. y is ignored.
  - *realseries.models.iforest.IForest.detect()*: Predict the score of a sample being anomaly by the detector. The anomaly score is returned.
- **LSTM_dynamic**
  - *realseries.models.lstm_dynamic.LSTM_dynamic.fit()*: Fit LSTM model. y is ignored.
  - *realseries.models.lstm_dynamic.LSTM_dynamic.detect()*: Predict the score of a sample being anomaly by the dynamic method. The anomaly sequence and score is returned.
- **Luminol**
  - *realseries.models.lumino.Lumino.detect()*: Predict the score of a sample being anomaly by the detector. The anomaly score is returned.
- **Random cut forest**
  - *realseries.models.rcforest.RCForest.detect()*: Predict the score of a sample being anomaly by the detector. The anomaly score is returned.
- **LSTM encoder decoder**
  - *realseries.models.rnn.LSTMED.fit()*: Fit LSTM. y is ignored.
  - *realseries.models.rnn.LSTMED.detect()*: Predict the score of a sample being anomaly by the LSTM. The anomaly score is returned.
- **SeqVL**
  - *realseries.models.seqvl.SeqVL.fit()*: Fit detector. y is ignored in unsupervised methods.
  - *realseries.models.seqvl.SeqVL.detect()*: Predict the score of a sample being anomaly by the detector. The anomaly score is returned.
- **SR_CNN**
  - *realseries.models.srcnn.SR_CNN.fit()*: Fit CNN model.
  - *realseries.models.srcnn.SR_CNN.detect()*: Predict the score of a sample being anomaly by the CNN. The anomaly score is returned.
- **VAE_AD**
  - *realseries.models.vae_ad.VAE_AD.fit()*: Fit detector. y is ignored in unsupervised methods.

  - *realseries.models.vae_ad.VAE_AD.detect()*: Predict the score of a sample being anomaly by the detector. The anomaly score is returned.

- **STL**
  - *realseries.models.stl.STL.fit()*: Fit STL model. y is ignored in unsupervised methods.
  - *realseries.models.stl.STL.forecast()*: Forecast the later value of a sequence. The array is returned.

- **Granger Causality**
  - *realseries.models.GC.GC.detect()*: Granger Causality detector, which is channel-level.
  - *realseries.models.DWGC.DWGC.detect()*: Dynamic Window-level Granger Causality detector.

See base class definition in *realseries.models.base*.

3.2 Data

The following functions are used for raw data loading easily.

- *realseries.utils.data.load_NAB()*: Load data in the *NAB_data* diary. Train DataFrame and Test DataFrame with labels are returned.
- *realseries.utils.data.load_Yahoo()*: Load data in the *Yahoo_data* diary. Train DataFrame and Test DataFrame with labels are returned.
- *realseries.utils.data.load_split_NASA()*: Load data in the *NASA* diary. Train DataFrame and Test DataFrame with labels are returned.

3.3 Visualize

The following functions are used plotting raw data and predicted result.

- *realseries.utils.visualize.plot_anom()*: The parameters mainly include `pd_data_label`, `pred_anom` and `pred_score`. `pd_data_label` is the `pandas.DataFrame()` with data and label, `pred_anom` is the array with predicted label, and `pred_score` is the corresponding anomaly score.
- *realseries.utils.visualize.plot_mne()*: The parameters mainly include `X, scalings, ch_types, color`. If `X` is the array and last column as label. We set label column to different `ch_type`, so it will show different color in the figure.

CHAPTER

FOUR

API REFERENCE

4.1 All Models

4.1.1 realseries.models.AR module

Created on Sun Apr 19 22:34:04 2020

@author: zhihengzhang

class `realseries.models.AR.`**`AR`**(*lag*)

Bases: *`realseries.models.base.BaseModel`*

Parameters **`lag`** – the time lag in AR model

`X_train`

training data in AR model.

`Y_train`

training label in AR model.

`detect`(*X*)

Parameters

- **`X`** – time series(dimension is 1 or 2)
- **`y`** – The default is None.

Returns the fitting result of training data.

Return type detection

`fit`(*X*, *y=None*)

Parameters

- **`X`** – time series(dimension is 1 or 2)
- **`y`** – The default is None.

Returns None.

4.1.2 realseries.models.DWGC module

Created on Fri Apr 17 11:57:41 2020

@author: zhihengzhang

class realseries.models.DWGC.**DWGC**(*win_len*, *model*, *index_lr*, *method*, *count*, *train_rate*)

Bases: *realseries.models.base.BaseModel*

Dynamic-window-level Granger Causality method, try to find the window-level causality on each channel pair.

Args: win_len: window length model:AR or NAR index_lr: leanring rate of causal-indexing coefficients method: option of fitting method, 'NAR'/'AR'

Attributes: causal_index : causal-indexing coefficients single_error1/single_error2 : the fitting error without other channel's dimension double_error: the fitting error with other channel's dimension

detect(*X*)

Parameters **X** – the pair of time series

Returns the causality on window-level

Return type Ftest_win

fit(*X*, *y=None*)

Fit the model

Parameters

- **x** (*array_like*) – The input sequence of shape (n_length, n_features) or (n_length,).
- **y** (*ndarray, optional*) – Ignored. Defaults to None.

4.1.3 realseries.models.GC module

Created on Thu Apr 16 11:07:05 2020

@author: zhihengzhang

class realseries.models.GC.**GC**(*win_len*, *model*, *method*, *train_rate*)

Bases: *realseries.models.base.BaseModel*

Parameters

- **win_len** – window length
- **model** – 'AR' or 'NAR-network'
- **method** – option of fitting method, 'NAR'/'AR'

-- single_error

the fitting error without other channel's dimension

-- double_error

the fitting error with other channel's dimension

detect(*X*)

Parameters **X** – time series pair

Returns window-level causality

Return type Ftest_win

fit(*X*, *y=None*)

Fit the model

Parameters

- **x** (*array_like*) – The input sequence of shape (n_length, n_features) or (n_length,).
- **y** (*ndarray, optional*) – Ignored. Defaults to None.

4.1.4 realseries.models.NAR module

Created on Tue Apr 14 16:36:45 2020

@author: studyzzh

class realseries.models.NAR.**NAR_Network**(*inputnodes*, *hiddennodes*, *outputnodes*, *learningrate*)

Bases: *realseries.models.base.BaseModel*

Parameters

- **inputnodes** (--) – the number of nodes in input layer.
- **hiddennodes** (--) – the number of nodes in hidden layer.
- **outputnodes** (--) – the number of nodes in outout layer.
- **rate** (-- *learning*) – the learning rate of NAR model.

-- fit_X

the fitting results on training data.

detect(*X*)

Parameters

- **X** – the input time series in shape of (,1) or (,2)
- **y** – The default is None.

Returns the fitting errors on training data

Return type output_errors

fit(*X*, *y=None*)

Parameters

- **X** – the input time series in shape of (,1) or (,2)
- **y** – The default is None.

4.1.5 realseries.models.base module

Base class for all time series analysis methods. It includes the methods like fit, detect and predict etc.

class realseries.models.base.**BaseModel**(*contamination=0.1*)

Bases: object

BaseModel class for all RealSeries predict/detect algorithms.

Parameters **contamination** (*float, optional*) – The amount of contamination of the data set, i.e. the proportion of outliers in the data set. Used when fitting to define the threshold on the decision function.. Defaults to 0.1.

Raises **ValueError** – Contamination must be in (0, 0.5].

abstract detect(*x*)

Predict using the trained detector.

Parameters **x** (*array_like*) – The input sequence of shape (n_length, n_features) or (n_length,).

Returns Outlier labels of shape (n_length,).For each sample of time series, whether or not it is an outlier. 0 for inliers and 1 for outliers.

Return type ndarray

abstract fit(*x*, *y=None*)

Fit the model

Parameters

- **x** (*array_like*) – The input sequence of shape (n_length, n_features) or (n_length,).
- **y** (*ndarray, optional*) – Ignored. Defaults to None.

forecast(*x*, *t*)

Forecast the input.

Parameters

- **x** (*array_like*) – The input sequence of shape (n_length, n_features) or (n_length,).
- **t** (*int*) – time index of to-be-forecast samples.

Returns Forecast samples of shape (n_length, n_features)

Return type X_1 (ndarray)

impute(*x*, *t*)

Impute the input data X at time index t.

Parameters

- **x** (*array_like*) – The input sequence of shape (n_length, n_features) or (n_length,).
- **t** (*int*) – time index of to-be-forecast samples.

Returns Impute samples of shape (n_length, n_features)

Return type X_1 (ndarray)

load(*path*)

Load the model from path

Parameters **path** (*string*) – model load path

save(*path*)
Save the model to path

Parameters **path** (*string*) – model save path

4.1.6 realseries.models.iforest module

The implementation of isolation forest method based on sklearn.

class realseries.models.iforest.**IForest**(*n_estimators=100*, *max_samples='auto'*, *contamination='auto'*, *max_features=1.0*, *bootstrap=False*, *n_jobs=1*, *random_state=None*, *verbose=0*)

Bases: *realseries.models.base.BaseModel*

Isolation forest algorithm.

The IsolationForest 'isolates' observations by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature.

Parameters

- **n_estimators** (*int, optional*) – The number of base estimators in the ensemble. Defaults to 100.
- **max_samples** (*int or float, optional*) – The number of samples to draw from X to train each base estimator. Defaults to "auto".
  - If int, then draw *max_samples* samples.
  - If float, then draw *max_samples * X.shape[0]* samples.
  - If "auto", then *max_samples=min(256, n_samples)*.

  If max_samples is larger than the number of samples provided, all samples will be used for all trees (no sampling).
- **contamination** (*'auto' or float, optional*) – The amount of contamination of the data set. Defaults to 'auto'.
- **max_features** (*int or float, optional*) – The number of features to draw from X to train each base estimator. Defaults to 1.
- **bootstrap** (*bool, optional*) – If True, individual trees are fit on random subsets of the training data sampled with replacement. If False, sampling without replacement is performed. Defaults to False.
- **n_jobs** (*int, optional*) – The number of jobs to run in parallel. Defaults to 1.
- **random_state** (*int, optional*) – If RandomState instance, random_state is the random number generator. If None, the random number generator is the RandomState instance used by *np.random*. Defaults to 0.
- **verbose** (*int, optional*) – Controls the verbosity of the tree building process. Defaults to None.

anomaly_score
Array of anomaly score.

IF
The isolation model.

estimators_
List of DecisionTreeClassifier.The collection of fitted sub-estimators.

estimators_samples_
List of arrays.The subset of drawn samples (i.e., the in-bag samples) for each base estimator.

max_samples_
The actual number of samples

detect(*X*)
Detect the test data by trained model.

Parameters **X** (*array_like*) – The input sequence with shape (n_sample, n_features).

Returns The predicted anomaly score.

Return type ndarray

property estimators_
The collection of fitted sub-estimators. Decorator for scikit-learn Isolation Forest attributes.

property estimators_samples_
The subset of drawn samples (i.e., the in-bag samples) for each base estimator. Decorator for scikit-learn Isolation Forest attributes.

fit(*X*, *y=None*)
Train the model.

Parameters

- **X** (*array_like*) – The input sequence with shape (n_sample, n_features).
- **y** (*ndarray, optional*) – The label. Defaults to None.

property max_samples_
The actual number of samples. Decorator for scikit-learn Isolation Forest attributes.

4.1.7 realseries.models.lstm_dynamic module

The lstm danamic threshold method is the implentation of paper 'Detecting Spacecraft Anomalies Using LSTMs and-Nonparametric Dynamic Thresholding'

class realseries.models.lstm_dynamic.**LSTM_dynamic**(*hidden_size=128*, *model_path='./model'*, *dropout=0.3*, *lr=0.001*, *lstm_batch_size=100*, *epochs=50*, *num_layers=2*, *l_s=120*, *n_predictions=10*, *batch_size=32*, *window_size=50*, *smoothing_perc=0.2*, *error_buffer=50*, *p=0.1*)

Bases: *realseries.models.base.BaseModel*

LSTM Dynamic method.

Parameters

- **hidden_size** (*int, optional*) – Hidden size of LSTM. Defaults to 128.
- **model_path** (*str, optional*) – Path for saving and loading model. Defaults to './model'.
- **dropout** (*float, optional*) – Dropout rate. Defaults to 0.3.
- **lr** (*float, optional*) – Learning rate. Defaults to 1e-3.
- **lstm_batch_size** (*int, optional*) – Batch size of training LSTM. Defaults to 100.
- **epochs** (*int, optional*) – Epochs of training. Defaults to 50.
- **num_layers** (*int, optional*) – Number of LSTM layer. Defaults to 2.

- **l_s** (*int, optional*) – Length of the input sequence for LSTM. Defaults to 120.
- **n_predictions** (*int, optional*) – Number of values to predict by input sequence. Defaults to 10.
- **batch_size** (*int, optional*) – Number of values to evaluate in each batch in the prediction stage. Defaults to 32.
- **window_size** (*int, optional*) – Window_size to use in error calculation. Defaults to 50.
- **smoothing_perc** (*float, optional*) – Percentage of total values used in EWMA smoothing. Defaults to 0.2.
- **error_buffer** (*int, optional*) – Number of values surrounding an error that are brought into the sequence. Defaults to 50.
- **p** (*float, optional*) – Minimum percent decrease between max errors in anomalous sequences (used for pruning). Defaults to 0.1.

model
The LSTM model.

y_test
The origin data for calculate error.

y_hat
The predicted data.

detect(*X*, *smoothed=True*)
Get anomaly score of input sequence.

Parameters

- **X** (*array_like*) – Input sequence.
- **smoothed** (*bool, optional*) – Whether to smooth the errors by EWMA. Defaults to True.

Returns (error_seq, error_seq_scores).The error_seq is list that stand the anomaly duration. The error_seq_scores is the corresponding anomaly score.

Return type tuple

fit(*X*, *split=0.25*, *monitor='val_loss'*, *patience=10*, *delta=0*, *verbose=True*)
Train the LSTM model.

Parameters

- **X** (*arrar_like*) – The 2-D input sequence with shape (n_samples, n_features)
- **split** (*float, optional*) – Fration to split for validation set. Defaults to 0.25.
- **monitor** (*str, optional*) – Monitor the validation loss by setting the monitor argument to 'val_loss'. Defaults to 'val_loss'.
- **patience** (*int, optional*) – Patience argument represents the number of epochs before stopping once your loss starts to increase (stops improving). Defaults to 10.
- **delta** (*int, optional*) – A threshold to whether quantify a loss at some epoch as improvement or not. If the difference of loss is below delta, it is quantified as no improvement. Better to leave it as 0 since we're interested in when loss becomes worse. Defaults to 0.
- **verbose** (*bool, optional*) – Verbose decides what to print. Defaults to True.

static obtain_anomaly(*y_test*, *y_hat*, *batch_size*, *window_size*, *smoothing_perc*, *p*, *l_s*, *error_buffer*, *smoothed=True*)

Obtain anomaly from the origin sequence and reconstructed sequence y_hat.

Parameters

- **y_test** (*ndarray*) – The origin 1-D signals array of test targets corresponding to true values to be predicted at end of each window.
- **y_hat** (*ndarray*) – The predicted 1-D sequence y_hat for each timestep in y_test
- **batch_size** (*int, optional*) – Number of values to evaluate in each batch in the prediction stage. Defaults to 32.
- **window_size** (*int, optional*) – Window_size to use in error calculation. Defaults to 50.
- **smoothing_perc** (*float, optional*) – Percentage of total values used in EWMA smoothing. Defaults to 0.2.
- **error_buffer** (*int, optional*) – Number of values surrounding an error that are brought into the sequence. Defaults to 50.
- **p** (*float, optional*) – Minimum percent decrease between max errors in anomalous sequences (used for pruning). Defaults to 0.1.
- **l_s** (*int, optional*) – Length of the input sequence for LSTM. Defaults to 120.
- **smoothed** (*bool, optional*) – Whether to smooth the errors by EWMA. Defaults to True.

Returns (error_seq, error_seq_scores)

Return type tuple

predict(*X*)

Predict the reconstructed output array y_hat.

Parameters **X** (*array_like*) – The input 2-D array.

Raises **ValueError** – Num_batches less than 0.

Returns The predicted array of lstm_encoder_decoder.

Return type ndarray

4.1.8 realseries.models.lumino module

The implementation of luminol method. Reference: https://github.com/linkedin/luminol

class realseries.models.lumino.**Lumino**

Bases: *realseries.models.base.BaseModel*

detect(*X*, *algorithm_name=None*, *algorithm_params=None*)

Detect the input sequence and return anomaly socre.

Parameters

- **X** (*array_like*) – 1-D time series with shape (n_samples,)
- **algorithm_name** (*str, optional*) – Algorithm_name. Defaults to None.
- **algorithm_params** (*dict, optional*) – Algorithm_params. Defaults to None. The algorithm_name and the corresponding algorithm_params are:

```
1.  'bitmap_detector': # behaves well for huge data sets, and it is␣
↪the default detector.
    {
    'precision'(4): # how many sections to categorize values,
    'lag_window_size'(2% of the series length): # lagging window␣
↪size,
    'future_window_size'(2% of the series length): # future window␣
↪size,
    'chunk_size'(2): # chunk size.
    }
2.  'default_detector': # used when other algorithms fails, not␣
↪meant to be explicitly used.
3.  'derivative_detector': # meant to be used when abrupt changes␣
↪of value are of main interest.
    {
    'smoothing factor'(0.2): # smoothing factor used to compute␣
↪exponential moving averages
                                # of derivatives.
    }
4.  'exp_avg_detector': # meant to be used when values are in a␣
↪roughly stationary range.
                        # and it is the default refine algorithm.
    {
    'smoothing factor'(0.2): # smoothing factor used to compute␣
↪exponential moving averages.
    'lag_window_size'(20% of the series length): # lagging window␣
↪size.
    'use_lag_window'(False): # if asserted, a lagging window of␣
↪size lag_window_size will be used.
    }
```

Returns Normalized anomaly score in [0,1].

Return type ndarray

fit()

Fit the model

Parameters

- **x** (*array_like*) – The input sequence of shape (n_length, n_features) or (n_length,).
- **y** (*ndarray, optional*) – Ignored. Defaults to None.

4.1.9 realseries.models.rcforest module

The implementation of random cur forest method. Reference: S. Guha, N. Mishra, G. Roy, & O. Schrijvers, Robust random cut forest based anomaly detection on streams, in Proceedings of the 33rd International conference on machine learning, New York, NY, 2016 (pp. 2712-2721). https://github.com/kLabUM/rrcf

class realseries.models.rcforest.**RCForest**(*shingle_size=32, num_trees=100, tree_size=50, random_state=0*)

Bases: *realseries.models.base.BaseModel*

Random cut forest.The Robust Random Cut Forest (RRCF) algorithm is an ensemble method for detecting outliers in streaming data. RRCF offers a number of features that many competing anomaly detection algorithms

lack. Specifically, RRCF:

- Is designed to handle streaming data.
- Performs well on high-dimensional data.
- Reduces the influence of irrelevant dimensions.
- Gracefully handles duplicates and near-duplicates that could otherwise mask the presence of outliers.
- Features an anomaly-scoring algorithm with a clear underlying statistical meaning.

Parameters

- **shingle_size** (*int, optional*) – Window size. Defaults to 32.
- **num_trees** (*int, optional*) – Number of estimators. Defaults to 100.
- **tree_size** (*int, optional*) – Number of leaf. Defaults to 50.
- **random_state** (*int, optional*) – Random state seed. Defaults to None.

detect(*X*)

Detect the input.

Parameters **X** (*array_like*) – Input sequence.

Returns Anomaly score.

Return type ndarray

fit(*X*, *y=None*)

Fit the model

Parameters

- **x** (*array_like*) – The input sequence of shape (n_length, n_features) or (n_length,).
- **y** (*ndarray, optional*) – Ignored. Defaults to None.

4.1.10 realseries.models.rnn module

RNN encoder decoder model. Reference 'LSTM-based Encoder-Decoder for Multi-sensor Anomaly Detection'

class realseries.models.rnn.**LSTMED**(*rnn_type='LSTM'*, *emsize=128*, *nhid=128*, *epochs=200*, *nlayers=2*, *batch_size=64*, *window_size=50*, *dropout=0.2*, *lr=0.0002*, *weight_decay=0.0001*, *clip=10*, *res_connection=False*, *prediction_window_size=10*, *model_path=None*, *seed=1111*)

Bases: *realseries.models.base.BaseModel*

RNN(LSTM) encoder decoder model for anomaly detection.

Parameters

- **rnn_type** (*str, optional*) – Type of recurrent net (RNN_TANH, RNN_RELU, LSTM, GRU, SRU). Defaults to 'LSTM'.
- **emsize** (*int, optional*) – Size of rnn input features. Defaults to 128.
- **nhid** (*int, optional*) – Number of hidden units per layer. Defaults to 128.
- **epochs** (*int, optional*) – Upper epoch limit. Defaults to 200.
- **nlayers** (*int, optional*) – Number of LSTM layers. Defaults to 2.
- **batch_size** (*int, optional*) – Batch size. Defaults to 64.

- **window_size** (*int, optional*) – LSTM input sequence length. Defaults to 50.
- **dropout** (*float, optional*) – Defaults to 0.2.
- **lr** (*float, optional*) – Learning rate. Defaults to 0.0002.
- **weight_decay** (*float, optional*) – Weight decay. Defaults to 1e-4.
- **clip** (*int, optional*) – Gradient clipping. Defaults to 10.
- **res_connection** (*bool, optional*) – Residual connection. This parameters has not been tested when setting *True*. Defaults to False.
- **prediction_window_size** (*int, optional*) – Prediction window size. Defaults to 10.
- **model_path** (*str, optional*) – The path to save or load model. Defaults to None.
- **seed** (*int, optional*) – Seed. Defaults to 1111.

model

LSTM model.

detect(*X*, *channel_idx=0*)

If X is an array of shape (n_samples, n_features), it need to be detected one by one channel.

Parameters

- **X** (*array_like*) – Input sequence.
- **channel_idx** (*int, optional*) – The index of feature cahnnel to detect.
- **0.** (*Defaults to*) –

Returns Anomaly score

Return type ndarray

fit(*X*, *y=None*, *augment_length=None*, *split=0.25*, *monitor='val_loss'*, *patience=10*, *delta=0*, *verbose=True*)

Train the detector.

Parameters

- **X** (*array_like*) – The input sequence of shape (n_length,).
- **y** (*array_like, optional*) – Ignored. Defaults to None.
- **augment_length** (*int, optional*) – The total number of samples after augmented. Defaults to None.
- **split** (*float, optional*) – Fration to split for validation set. Defaults to 0.25.
- **monitor** (*str, optional*) – Monitor the validation loss by setting the monitor argument to 'val_loss'. Defaults to 'val_loss'.
- **patience** (*int, optional*) – Patience argument represents the number of epochs before stopping once your loss starts to increase (stops improving). Defaults to 10.
- **delta** (*int, optional*) – A threshold to whether quantify a loss at some epoch as improvement or not. If the difference of loss is below delta, it is quantified as no improvement. Better to leave it as 0 since we're interested in when loss becomes worse. Defaults to 0.
- **verbose** (*bool, optional*) – Verbose decides what to print. Defaults to True.

4.1.11 realseries.models.seqvl module

Introduction of seqvl.

class realseries.models.seqvl.**SeqVL**(*contamination=0.1*, *name='SeqVL'*, *num_epochs=250*, *batch_size=1*, *lr=0.001*, *lr_decay=0.8*, *lamb=10*, *clip_norm_value=12.0*, *data_split_rate=0.5*, *window_size=30*, *window_count=300*, *h_dim=24*, *z_dim=5*, *l_h_dim=24*)

Bases: *realseries.models.base.BaseModel*

detect(*X*, *thres*)

Detect the data by trained model.

Parameters

- **X** (*array_like*) – 1-D time series with length L.
- **thres** (*float*) – Threshold.

Returns

Dict containing results. 0-1 sequence indicates whether the last point of a window is an anomaly. length: L - window_size + 1

Return type dict

fit(*X*)

Train the model.

Parameters **X** (*array_like*) – Input sequence.

reshape_for_test(*X*)

Reshape the data gor test.

Parameters **X** (*array_like*) – Input data.

Returns Reshaped data.

Return type ndarray

reshape_for_training(*X*)

Reshape the data for training.

Parameters **X** (*ndarray*) – 1-D time series

Returns

input with shape [-1, window_count, window_size], label with shape [-1, window_count]

Return type tuple

4.1.12 realseries.models.sr module

class realseries.models.sr.**SpectralResidual**(*series*, *threshold*, *mag_window*, *score_window*)

Bases: *realseries.models.base.BaseModel*

SpectralResidual calss.

Parameters

- **series** – input time series with shape (n_sample,)
- **threshold** – the threshold that apply anomaly score
- **mag_window** – the window of avarage filter when calculating spectral mag

- **score_window** – the window of average filter when calculating score

detect()

Predict using the trained detector.

Parameters **x** (*array_like*) – The input sequence of shape (n_length, n_features) or (n_length,).

Returns Outlier labels of shape (n_length,).For each sample of time series, whether or not it is an outlier. 0 for inliers and 1 for outliers.

Return type ndarray

static extend_series(*values*, *extend_num=5*, *look_ahead=5*)

extend the array data by the predicted next value

Parameters

- **values** (*ndarray*) – array of float numbers.
- **extend_num** (*int, optional*) – number of values added to the back of data. Defaults to 5.
- **look_ahead** (*int, optional*) – number of previous values used in prediction. Defaults to 5.

Raises **ValueError** – the parameter 'look_ahead' must be at least 1

Returns The result array.

Return type ndarray

fit()

Fit the model

Parameters

- **x** (*array_like*) – The input sequence of shape (n_length, n_features) or (n_length,).
- **y** (*ndarray, optional*) – Ignored. Defaults to None.

generate_spectral_score(*series*)

static predict_next(*values*)

Predicts the next value by sum up the slope of the last value with previous values.

Mathematically, $g = 1/m * \sum_{i=1}^{m} g(x_n, x_{n-i})$, $x_{n+1} = x_{n-m+1} + g * m$, where $g(x_i, x_j) = (x_i - x_j)/(i - j)$.

Parameters **values** (*list*) – a list of float numbers.

Raises **ValueError** – Length lsit should at least 2.

Returns The predicted next value.

Return type float

spectral_residual_transform(*values*)

Transform a time series into spectral residual series by FFT.

Parameters **values** (*ndarray*) – Array of values.

Returns Spectral residual values.

Return type ndarray

4.1.13 realseries.models.srcnn module

class realseries.models.srcnn.**SR_CNN**(*model_path*, *window=128*, *lr=1e-06*, *seed=0*, *epochs=20*, *batch_size=64*, *dropout=0.2*, *num_worker=0*)

Bases: *realseries.models.base.BaseModel*

The sali_map method for anomaly detection.

Parameters

- **model_path** (*str, optional*) – Path for saving and loading model.
- **window** (*int, optional*) – Length of each sample for input. Defaults to 128.
- **lr** (*float, optional*) – Learning rate. Defaults to 1e-6.
- **seed** (*int, optional*) – Random seed. Defaults to 0.
- **epochs** (*int, optional*) – Defaults to 20.
- **batch_size** (*int, optional*) – Defaults to 64.
- **dropout** (*float, optional*) – Defaults to 0.2.
- **num_worker** (*int, optional*) – Defaults to 0.

model

CNN model built by torch.

detect(*X*, *y*, *back_k=0*, *backaddnum=5*, *step=1*)

Get anomaly score of input sequence.

Parameters

- **X** (*array_like*) – Input sequence.
- **y** – Ignored.
- **back_k** (*int, optional*) – Not test. Defaults to 0.
- **backaddnum** (*int, optional*) – Not test. Defaults to 5.
- **step** (*int, optional*) – Stride of sliding window in detecing stage. Defaults to 1.

Returns Anomaly score.

Return type ndarray

fit(*X*, *step=64*, *num=10*, *back_k=0*)

Train the model

Parameters

- **X** (*array_like*) – The input 1-D array.
- **step** (*int, optional*) – Stride of sliding window. Defaults to 64.
- **num** (*int, optional*) – Number of added anomaly points to each window. Defaults to 10.
- **back_k** (*int, optional*) – Defaults to 0.

4.1.14 realseries.models.stl module

class realseries.models.stl.**STL**

Bases: *realseries.models.base.BaseModel*

static calc_seasonal(*detrended*, *period*)

Calculate seasonal from detrended data.

Parameters

- **detrended** (*ndarray*) – Input detrended data.
- **period** (*float or int*) – The period of data.

Returns The seasonal and the period_averages.

Return type (ndarray, ndarray)

static calc_trend(*observed*, *lo_frac=0.6*, *lo_delta=0.01*)

calculate trend from observed data.

Parameters

- **observed** (*ndarray*) – Input array.
- **lo_frac** (*float, optional*) – Defaults to 0.6.
- **lo_delta** (*float, optional*) – Defaults to 0.01.

Returns The trend.

Return type ndarray

detect()

Predict using the trained detector.

Parameters **x** (*array_like*) – The input sequence of shape (n_length, n_features) or (n_length,).

Returns Outlier labels of shape (n_length,).For each sample of time series, whether or not it is an outlier. 0 for inliers and 1 for outliers.

Return type ndarray

static drift(*data*, *n=3*)

The drift forecast for the next point is a linear extrapolation from the previous n points in the series.

Parameters

- **data** (*ndrray*) – Observed data, presumed to be ordered in time.
- **n** (*int*) – period over which to calculate linear model for extrapolation.

Returns a single-valued forecast for the next value in the series.

Return type float

fit(*df*, *period=365*, *lo_frac=0.6*, *lo_delta=0.01*)

Train the STL decompose model.Y[t] = T[t] + S[t] + e[t]

Parameters

- **df** (*DataFrame*) – Input data.
- **period** (*int, optional*) – Defaults to 365.

- **lo_frac** (*float, optional*) – Defaults to 0.6.
- **lo_delta** (*float, optional*) – Defaults to 0.01.

Returns Dict results.

Return type dict

forecast(*stl*, *forecast_func='drift'*, *steps=10*, *seasonal=False*)

Forecast the given decomposition `stl` forward by `steps`

Parameters

- **stl** (*object*) – STL object.
- **forecast_func** (*str, optional*) – Defaults to 'drift'.
- **steps** (*int, optional*) – Defaults to 10.
- **seasonal** (*bool, optional*) – Defaults to False.

Returns forecast dataframe

Return type DataFrame

static mean(*data*, *n=3*)

static naive(*data*, *n=7*)

4.1.15 realseries.models.vae_ad module

class realseries.models.vae_ad.**VAE_AD**(*name='VAE_AD'*, *num_epochs=256*, *batch_size=256*, *lr=0.001*, *lr_decay=0.8*, *clip_norm_value=12.0*, *weight_decay=0.001*, *data_split_rate=0.5*, *window_size=120*, *window_step=1*, *h_dim=100*, *z_dim=5*)

Bases: *realseries.models.base.BaseModel*

The Donut-VAE version for anomaly detection

Parameters

- **name** (*str, optional*) – Model name. Defaults to 'VAE_AD'.
- **num_epochs** (*int, optional*) – Epochs for model training. Defaults to 256.
- **batch_size** (*int, optional*) – Batch size for model training. Defaults to 256.
- **lr** (*[type], optional*) – Learning rate. Defaults to 1e-3.
- **lr_decay** (*float, optional*) – Learning rate decay. Defaults to 0.8.
- **clip_norm_value** (*float, optional*) – Gradient clip value. Defaults to 12.0.
- **weight_decay** (*[type], optional*) – L2 regularization. Defaults to 1e-3.
- **data_split_rate** (*float, optional*) – Defaults to 0.5.
- **window_size** (*int, optional*) – Defaults to 120.
- **window_step** (*int, optional*) – Defaults to 1.
- **h_dim** (*int, optional*) – Hidden dim between x and z for VAE's encoder and decoder Defaults to 100.
- **z_dim** (*int, optional*) – Defaults to 5.

model
VAE model built by torch.

detect(*X*)
Get anomaly score of input sequence.

Parameters **X** (*array_like*) – Input sequence.

Returns origin_series: ndarray, Origin time series score: ndarray, Corresponding anomaly score.

Return type A dict with attributes

fit(*X*)
Train the model

Parameters **X** (*array_like*) – The input 1-D array.

forecast(*X*)
Forecast the input.

Parameters

- **x** (*array_like*) – The input sequence of shape (n_length, n_features) or (n_length,).
- **t** (*int*) – time index of to-be-forecast samples.

Returns Forecast samples of shape (n_length, n_features)

Return type X_1 (ndarray)

load(*path*)
Load the model from path

Parameters **path** (*string*) – model load path

predict(*X*)

save(*path*)
Save the model to path

Parameters **path** (*string*) – model save path

4.1.16 realseries.models.vae_dense module

class realseries.models.vae_dense.**VAE_Dense**(*window_size*, *channels*, *name='VAE_Dense'*, *num_epochs=256*, *batch_size=64*, *lr=0.001*, *lr_decay=0.8*, *clip_norm_value=12.0*, *weight_decay=0.001*, *h_dim=200*, *z_dim=20*)

Bases: *realseries.models.base.BaseModel*

The Donut-VAE version for anomaly detection

Parameters

- **window_size** (*int*) –
- **channels** (*int*) – Channel count of the input signals.
- **name** (*str, optional*) – Model name. Defaults to 'VAE_Dense'.
- **num_epochs** (*int, optional*) – Epochs for model training. Defaults to 256.
- **batch_size** (*int, optional*) – Batch size for model training. Defaults to 256.
- **lr** (*[type], optional*) – Learning rate. Defaults to 1e-3.

- **lr_decay** (*float, optional*) – Learning rate decay. Defaults to 0.8.
- **clip_norm_value** (*float, optional*) – Gradient clip value. Defaults to 12.0.
- **weight_decay** (*[type], optional*) – L2 regularization. Defaults to 1e-3.
- **h_dim** (*int, optional*) – Hidden dim between x and z for VAE's encoder and decoder Defaults to 200.
- **z_dim** (*int, optional*) – Defaults to 20.

model

VAE model built by torch.

detect(*X*)

Get anomaly score of input sequence.

Parameters **X** (*array_like*) – Input sequence.

Returns origin_series: ndarray [timesteps, channels], Origin time series recon_series: ndarray [timesteps, channels], Reconstruct time series score: ndarray [timesteps, channels], Corresponding anomaly score.

Return type A dict with attributes

fit(*X*)

Train the model

Parameters **X** (*array_like*) – The input 2-D array. The first dimension denotes timesteps. The second dimension denotes the signal channels.

flatten(*x*)

forecast(*X*)

Forecast the input.

Parameters

- **x** (*array_like*) – The input sequence of shape (n_length, n_features) or (n_length,).
- **t** (*int*) – time index of to-be-forecast samples.

Returns Forecast samples of shape (n_length, n_features)

Return type X_1 (ndarray)

load(*path*)

Load the model from path

Parameters **path** (*string*) – model load path

predict(*X*)

reform(*x*)

save(*path*)

Save the model to path

Parameters **path** (*string*) – model save path

4.1.17 realseries.models.crmmd module

The crmmd is the implentation of paper 'Calibrated Reliable Regression using Maximum Mean Discrepancy' https://arxiv.org/abs/2006.10255

class realseries.models.crmmd.**CRMMD**(*kernel_type='LSTM', input_size=128, hidden_sizes=[128, 64], prediction_window_size=1, activation='tanh', dropout_rate=0.2, variance=True, lr=0.0002, weight_decay=0.001, grad_clip=10, epochs_hnn=400, epochs_mmd=100, batch_size=1024, window_size=15, model_path='./model', seed=1111*)

Bases: *realseries.models.base.BaseModel*

HNN forecaster for uncertainty prediction.

Parameters

- **kernel_type** (*str, optional*) – Type of recurrent net (RNN, LSTM, GRU). Defaults to 'LSTM'.
- **input_size** (*int, optional*) – Size of rnn input features. Defaults to 128.
- **hidden_sizes** (*list, optional*) – Number of hidden units per layer. Defaults to [128,64].
- **prediction_window_size** (*int, optional*) – Prediction window size. Defaults to 1.
- **activation** (*str,optional*) – The activation func to use. Can be either `'tanh'` or `'relu'`. Default: `'relu'`
- **dropout_rate** (*float, optional*) – Defaults to 0.2.
- **variance** (*bool, optional*) – Whether to add a variance item at the last layer to indicate uncertainty. Default to True
- **lr** (*float, optional*) – Learning rate. Defaults to 0.0002.
- **weight_decay** (*float, optional*) – Weight decay. Defaults to 1e-4.
- **grad_clip** (*int, optional*) – Gradient clipping. Defaults to 10.
- **epochs_hnn** (*int, optional*) – Upper epoch limit for the first training phase (HNN). Defaults to 200.
- **epochs_mmd** (*int, optional*) – Upper epoch limit for the second training phase (MMD). Defaults to 200.
- **batch_size** (*int, optional*) – Batch size. Defaults to 1024.
- **window_size** (*int, optional*) – LSTM input sequence length. Defaults to 15.
- **model_path** (*str, optional*) – The path to save or load model. Defaults to './model'.
- **seed** (*int, optional*) – Seed. Defaults to 1111.

model

HNN model.

evaluation_model(*scaler, test_data, test_label, t=1, confidence=95*)

Get predictive intervals and evaluation scores.

Parameters

- **scaler** – receive the scaler of data loader
- **test_data** (*numpy array*) – The 3-D input sequence (n_samples,window_size,n_features)

- **test_label** (*numpy array*) – The 2-D input sequence (n_samples,prediction_window_size)
- **t** (*optional, int*) – the forecasting horizon, default to 1
- **confidence** (*optional, int*) – the confidence of predictive intervals. Default to 95, output 95% predictive intervals.

Returns the lower bound and the upper bound arrays of the predictive intervals for test data. rmse (float): the rmse score calibration error (float): the uncertainty evaluation score for test data.

Return type PIs (two numpy arrays)

fit(*train_data*, *train_label*, *val_data*, *val_label*, *patience=50*, *delta=0*, *verbose=True*)

Train the LSTM model.

Parameters

- **train_data** (*numpy array*) – The 3-D input sequence (n_samples,window_size,n_features)
- **train_label** (*numpy array*) – The 2-D input sequence (n_samples,prediction_window_size)
- **val_data** (*numpy array*) – The 3-D input sequence (n_samples,window_size,n_features)
- **val_label** (*numpy array*) – The 2-D input sequence (n_samples,prediction_window_size)
- **patience** (*int, optional*) – Patience argument represents the number of epochs before stopping once your loss starts to increase (stops improving). Defaults to 10.
- **delta** (*int, optional*) – A threshold to whether quantify a loss at some epoch as improvement or not. If the difference of loss is below delta, it is quantified as no improvement. Better to leave it as 0 since we're interested in when loss becomes worse. Defaults to 0.
- **verbose** (*bool, optional*) – Verbose decides what to print. Defaults to True.

model

a trained model to save to model_path/checkpoint.pt.

forecast(*scaler*, *test_data*, *t=1*, *confidence=95*, *is_uncertainty=True*)

Get predictive intervals and evaluation scores.

Parameters

- **scaler** – receive the scaler of data loader
- **test_data** (*numpy array*) – The 3-D input sequence (n_samples,window_size,n_features)
- **t** (*optional, int*) – the forecasting horizon, default to 1
- **confidence** (*optional, int*) – the confidence of predictive intervals. Default to 95, output 95% predictive intervals.
- **is_uncertainty** (*optional, bool*) – whether to get uncertainty, if true, outputing PIs, if false, outputing means. Defaults to True.

Returns the lower bound and the upper bound arrays of the predictive intervals for test data.

Return type PIs (two numpy arrays)

load_model(*path=PosixPath('model/checkpoint_crmmd.pt')*)

Load Pytorch model.

Parameters **model_path** (*string or path*) – Path for loading model.

save_model(*model_path=PosixPath('model/checkpoint_crmmd.pt')*)
Save pytorch model.

Parameters **model_path** (*string or path*) – Path for saving model.

4.1.18 realseries.models.hnn module

The models(HNN, Deep-ensemble, MC-dropout, CRMMD…) for time series forcasting and uncertainty prediction.

class realseries.models.hnn.**HNN**(*kernel_type='LSTM'*, *input_size=128*, *hidden_sizes=[128, 64]*, *prediction_window_size=1*, *activation='tanh'*, *dropout_rate=0.2*, *variance=True*, *lr=0.0002*, *weight_decay=0.001*, *grad_clip=10*, *epochs=200*, *batch_size=1024*, *window_size=15*, *model_path='./model'*, *seed=1111*)

Bases: *realseries.models.base.BaseModel*

HNN forecaster for uncertainty prediction.

Parameters

- **kernel_type** (*str, optional*) – Type of recurrent net (RNN, LSTM, GRU). Defaults to 'LSTM'.
- **input_size** (*int, optional*) – Size of rnn input features. Defaults to 128.
- **hidden_sizes** (*list, optional*) – Number of hidden units per layer. Defaults to [128,64].
- **prediction_window_size** (*int, optional*) – Prediction window size. Defaults to 1.
- **activation** (*str,optional*) – The activation func to use. Can be either 'tanh' or 'relu'. Default: 'relu'
- **dropout_rate** (*float, optional*) – Defaults to 0.2.
- **variance** (*bool, optional*) – Whether to add a variance item at the last layer to indicate uncertainty. Default to True
- **lr** (*float, optional*) – Learning rate. Defaults to 0.0002.
- **weight_decay** (*float, optional*) – Weight decay. Defaults to 1e-4.
- **grad_clip** (*int, optional*) – Gradient clipping. Defaults to 10.
- **epochs** (*int, optional*) – Upper epoch limit. Defaults to 200.
- **batch_size** (*int, optional*) – Batch size. Defaults to 1024.
- **window_size** (*int, optional*) – LSTM input sequence length. Defaults to 15.
- **model_path** (*str, optional*) – The path to save or load model. Defaults to './model'.
- **seed** (*int, optional*) – Seed. Defaults to 1111.

model
HNN model.

evaluation_model(*scaler*, *test_data*, *test_label*, *t=1*, *confidence=95*)
Get predictive intervals and evaluation scores.

Parameters

- **scaler** – receive the scaler of data loader

- **test_data** (*numpy array*) – The 3-D input sequence (n_samples,window_size,n_features)
- **test_label** (*numpy array*) – The 2-D input sequence (n_samples,prediction_window_size)
- **t** (*optional, int*) – the forecasting horizon, default to 1
- **confidence** (*optional, int*) – the confidence of predictive intervals. Default to 95, output 95% predictive intervals.

Returns the lower bound and the upper bound arrays of the predictive intervals for test data. rmse (float): the rmse score calibration error (float): the uncertainty evaluation score for test data.

Return type PIs (two numpy arrays)

fit(*train_data*, *train_label*, *val_data*, *val_label*, *patience=50*, *delta=0*, *verbose=True*)

Train the LSTM model.

Parameters

- **train_data** (*numpy array*) – The 3-D input sequence (n_samples,window_size,n_features)
- **train_label** (*numpy array*) – The 2-D input sequence (n_samples,prediction_window_size)
- **val_data** (*numpy array*) – The 3-D input sequence (n_samples,window_size,n_features)
- **val_label** (*numpy array*) – The 2-D input sequence (n_samples,prediction_window_size)
- **patience** (*int, optional*) – Patience argument represents the number of epochs before stopping once your loss starts to increase (stops improving). Defaults to 10.
- **delta** (*int, optional*) – A threshold to whether quantify a loss at some epoch as improvement or not. If the difference of loss is below delta, it is quantified as no improvement. Better to leave it as 0 since we're interested in when loss becomes worse. Defaults to 0.
- **verbose** (*bool, optional*) – Verbose decides what to print. Defaults to True.

model

a trained model to save to model_path/checkpoint_hnn.pt.

forecast(*scaler*, *test_data*, *t=1*, *confidence=95*, *is_uncertainty=True*)

Get predictive intervals and evaluation scores.

Parameters

- **scaler** – receive the scaler of data loader
- **test_data** (*numpy array*) – The 3-D input sequence (n_samples,window_size,n_features)
- **t** (*optional, int*) – the forecasting horizon, default to 1
- **confidence** (*optional, int*) – the confidence of predictive intervals. Default to 95, output 95% predictive intervals.
- **is_uncertainty** (*optional, bool*) – whether to get uncertainty, if true, outputing PIs, if false, outputing means. Defaults to True.

Returns the lower bound and the upper bound arrays of the predictive intervals for test data.

Return type PIs (two numpy arrays)

load_model(*path=PosixPath('model/checkpoint_hnn.pt')*)
Load Pytorch model.

Parameters **model_path** (*string or path*) – Path for loading model.

save_model(*model_path=PosixPath('model/checkpoint_hnn.pt')*)
Save pytorch model.

Parameters **model_path** (*string or path*) – Path for saving model.

4.1.19 realseries.models.mc_dropout module

The models(HNN, Deep-ensemble, MC-dropout, CRMMD…) for time series forcasting and uncertainty prediction.

class realseries.models.mc_dropout.**MC_dropout**(*kernel_type='LSTM', input_size=128, hidden_sizes=[128, 64], prediction_window_size=1, activation='tanh', dropout_rate=0.2, variance=True, lr=0.0002, weight_decay=0.001, grad_clip=10, epochs=200, batch_size=1024, window_size=15, model_path='./model', seed=1111*)

Bases: *realseries.models.base.BaseModel*

MC-dropout forecaster for uncertainty prediction.

Parameters

- **kernel_type** (*str, optional*) – Type of recurrent net (RNN, LSTM, GRU). Defaults to 'LSTM'.
- **input_size** (*int, optional*) – Size of rnn input features. Defaults to 128.
- **hidden_sizes** (*list, optional*) – Number of hidden units per layer. Defaults to [128,64].
- **prediction_window_size** (*int, optional*) – Prediction window size. Defaults to 1.
- **activation** (*str,optional*) – The activation func to use. Can be either `'tanh'` or `'relu'`. Default: `'relu'`
- **dropout_rate** (*float, optional*) – Defaults to 0.2.
- **variance** (*bool, optional*) – Whether to add a variance item at the last layer to indicate uncertainty. Default to True
- **lr** (*float, optional*) – Learning rate. Defaults to 0.0002.
- **weight_decay** (*float, optional*) – Weight decay. Defaults to 1e-4.
- **grad_clip** (*int, optional*) – Gradient clipping. Defaults to 10.
- **epochs** (*int, optional*) – Upper epoch limit. Defaults to 200.
- **batch_size** (*int, optional*) – Batch size. Defaults to 1024.
- **window_size** (*int, optional*) – LSTM input sequence length. Defaults to 15.
- **model_path** (*str, optional*) – The path to save or load model. Defaults to './model'.
- **seed** (*int, optional*) – Seed. Defaults to 1111.

model
regular MC-dropout model.

evaluation_model(*scaler, test_data, test_label, t=1, confidence=95, mc_times=400*)
Get predictive intervals and evaluation scores.

Parameters

- **scaler** – receive the scaler of data loader
- **test_data** (*numpy array*) – The 3-D input sequence (n_samples,window_size,n_features)
- **test_label** (*numpy array*) – The 2-D input sequence (n_samples,prediction_window_size)
- **t** (*optional, int*) – the forecasting horizon, default to 1
- **confidence** (*optional, int*) – the confidence of predictive intervals. Default to 95, output 95% predictive intervals.
- **mc_times** (*optional, int*) – the sampling times of MC dropout, Default to 400

Returns the lower bound and the upper bound arrays of the predictive intervals for test data. rmse (float): the rmse score calibration error (float): the uncertainty evaluation score for test data.

Return type PIs (two numpy arrays)

fit(*train_data*, *train_label*, *val_data*, *val_label*, *monitor='val_loss'*, *patience=50*, *delta=0*, *verbose=True*)

Train the LSTM model.

Parameters

- **train_data** (*numpy array*) – The 3-D input sequence (n_samples,window_size,n_features)
- **train_label** (*numpy array*) – The 2-D input sequence (n_samples,prediction_window_size)
- **val_data** (*numpy array*) – The 3-D input sequence (n_samples,window_size,n_features)
- **val_label** (*numpy array*) – The 2-D input sequence (n_samples,prediction_window_size)
- **patience** (*int, optional*) – Patience argument represents the number of epochs before stopping once your loss starts to increase (stops improving). Defaults to 10.
- **delta** (*int, optional*) – A threshold to whether quantify a loss at some epoch as improvement or not. If the difference of loss is below delta, it is quantified as no improvement. Better to leave it as 0 since we're interested in when loss becomes worse. Defaults to 0.
- **verbose** (*bool, optional*) – Verbose decides what to print. Defaults to True.

model

a trained model to save to model_path/checkpoint_mc.pt.

forecast(*scaler*, *test_data*, *t=1*, *confidence=95*, *mc_times=400*, *is_uncertainty=True*)

Get predictive intervals and evaluation scores.

Parameters

- **scaler** – receive the scaler of data loader
- **test_data** (*numpy array*) – The 3-D input sequence (n_samples,window_size,n_features)
- **t** (*optional, int*) – the forecasting horizon, default to 1
- **confidence** (*optional, int*) – the confidence of predictive intervals. Default to 95, output 95% predictive intervals.

- **mc_times** (*optional, int*) – the sampling times of MC dropout, Default to 400
- **is_uncertainty** (*optional, bool*) – whether to get uncertainty, if true, outputing PIs, if false, outputing means. Defaults to True.

Returns the lower bound and the upper bound arrays of the predictive intervals for test data.

Return type PIs (two numpy arrays)

load_model(*path=PosixPath('model/checkpoint_mc.pt')*)
Load Pytorch model.

Parameters **model_path** (*string or path*) – Path for loading model.

save_model(*model_path=PosixPath('model/checkpoint_mc.pt')*)
Save pytorch model.

Parameters **model_path** (*string or path*) – Path for saving model.

4.2 Utility Functions

4.2.1 realseries.utils.data module

The function for load and process data.

realseries.utils.data.**generate_arma_data**(*n=1000, ar=None, ma=None, contamination_rate=0.05, contamination_variance=20, random_seed=None*)
Generate synthetic data. Utility function for generate synthetic data for time series data

- raw data for forecasting.
- with contamination for anomaly detection.

Generate using linear method.

Parameters

- **n** (*int, optional*) – The length of training time series to generate. Defaults to 1000.
- **ar** (*float array, optional*) – Parameter of AR model. Defaults to None.
- **ma** (*float array, optional*) – Parameter of MA model. Defaults to None.
- **contamination_rate** (*float, optional*) – The amount of contamination of the dataset in (0., 0.1). Defaults to 0.05.
- **contamination_variance** (*float, optional*) – Variance of contamination. Defaults to 20.
- **random_seed** (*int, optional*) – Specify a random seed if need. Defaults to None.

realseries.utils.data.**load_NAB**(*dirname='realKnownCause', filename='nyc_taxi.csv', fraction=0.5*)
Load data from NAB dataset.

Parameters

- **dirname** (*str, optional*) – Dirname in `examples/data/NAB_data` . Defaults to 'realKnownCause'.
- **filename** (*str, optional*) – The name of csv file. Defaults to 'nyc_taxi.csv'.
- **fraction** (*float, optional*) – The amount of data used for test set. Defaults to 0.5.

Returns The pd.DataFrame instance of train and test set.

Return type (DataFrame, DataFrame)

`realseries.utils.data.load_SMD(data_name='machine-1-1')`

Load SMD dataset.

Parameters **data_name** (*str, optional*) – The filename of txt. Defaults to 'machine-1-1'.

Returns Train_data, test_data and test_label

Return type pd.DataFrame

`realseries.utils.data.load_Yahoo(dirname='A1Benchmark', filename='real_1.csv', fraction=0.5, use_norm=False, detail=True)`

Load Yahoo dataset.

Parameters

- **dirname** (*str, optional*) – Directory name. Defaults to 'A1Benchmark'.
- **filename** (*str, optional*) – File name. Defaults to 'real_1.csv'.
- **fraction** (*float, optional*) – Data split rate. Defaults to 0.5.
- **use_norm** (*bool, optional*) – Whether to use data normalize.

Returns train and test DataFrame.

Return type pd.DataFrame

`realseries.utils.data.load_exp_data(dataname='pm25', window_szie=15, prediction_window_size=1, fractions=[0.6, 0.2, 0.2], isshuffle=True, isscaler=True)`

reading data and pro-processing, get training data, validation data and test data for model.

Parameters

- **dataname** (*str, optional*) – the name of dataset, eg: 'pm25', 'bike_sharing', 'air_quality', 'metro_traffic'.
- **window_size** (*int, optional*) – Number of lag observations as input. Defaults to 15.
- **prediction_window_size** (*int, optional*) – Prediction window size. Defaults to 10.
- **fractions** – (list, optional): the training data, test data and validation data ratio, Defaults to [0.6,0.2,0.2].
- **is_shuffle** (*bool, optional*) – whether to shuffle the raw data. Defaults to True.
- **is_scaler** – (bool, optional): whether to scale the raw data. Defaults to True.

Returns train_data, train_label, test_data, test_label, validation_data, validation_label

Return type a splitted dataset(NumPy array)

`realseries.utils.data.load_split_NASA(chan_id='T-9')`

Load NASA data for lstm dynamic method.

Parameters **chan_id** (*str, optional*) – The name of file. Defaults to 'T-9'.

Returns A tuple contains train_set and test_set.

Return type pd.DataFrame

`realseries.utils.data.load_splitted_RNN(dirname='power_demand', filename='power_data.csv')`

Load data from RNN dataset.

Parameters

- **dirname** (*str, optional*) – Dirname in `examples/data/RNN_data` . Defaults to 'power_demand'.
- **filename** (*str, optional*) – The name of csv file. Defaults to 'power_data.csv'.

Returns The pd.DataFrame instance of train and test set.

Return type (DataFrame, DataFrame)

4.2.2 realseries.utils.dataset module

Load Data.

class `realseries.utils.dataset.`**Data**

Bases: `object`

data2supervised(*infer_length*, *pred_length*, *column*)

[summary]

Parameters

- **infer_length** (*[type]*) – [description]
- **pred_length** (*[type]*) – [description]
- **column** (*[type]*) – [description]

data_iterator(*batchsize*)

[summary]

Parameters **batchsize** (*[type]*) – [description]

Returns [description]

Return type [type]

data_to_seqvl_format(*window_size*, *window_count*, *split_rate*)

[summary]

Parameters

- **window_size** (*[type]*) – [description]
- **window_count** (*[type]*) – [description]
- **split_rate** (*[type]*) – [description]

Returns [description]

Return type [type]

load_data(*path*)

[summary]

Parameters **path** (*[type]*) – [description]

Returns [description]

Return type [type]

load_yahoo(*path*)

[summary]

Parameters **path** (*[type]*) – [description]

normalize(*normalize_type=None*)

[summary]

Parameters **normalize_type** (*[type], optional*) – [description]. Defaults to None.

Raises **NameError** – [description]

4.2.3 realseries.utils.errors module

"The function in lstm dynamic method.

`realseries.utils.errors.get_errors(batch_size, window_size, smoothing_perc, y_test, y_hat, smoothed=True)`

Calculate the difference between predicted telemetry values and actual values, then smooth residuals using ewma to encourage identification of sustained errors/anomalies.

Parameters

- **batch_size** (*int*) – Number of values to evaluate in each batch in the prediction stage.
- **window_size** (*int*) – Window_size to use in error calculation.
- **smoothing_perc** (*float*) – Percentage of total values used in EWMA smoothing.
- **y_test** (*ndarray*) – Array of test targets corresponding to true values to be predicted at end of each sequence
- **y_hat** (*ndarray*) – predicted test values for each timestep in y_test
- **smoothed** (*bool, optional*) – If False, return unsmooothed errors (used for assessing quality of predictions)

Returns unsmoothed errors (residuals) e_s (list): smoothed errors (residuals)

Return type e (list)

`realseries.utils.errors.process_errors(p, l_s, batch_size, window_size, error_buffer, y_test, y_hat, e_s)`

Using windows of historical errors (h = batch size * window size), calculate the anomaly threshold (epsilon) and group any anomalous error values into continuos sequences. Calculate score for each sequence using the max distance from epsilon.

Parameters

- **p** (*float, optional*) – Minimum percent decrease between max errors in anomalous sequences (used for pruning).
- **l_s** (*int, optional*) – Length of the input sequence for LSTM.
- **batch_size** (*int*) – Number of values to evaluate in each batch in the prediction stage.
- **window_size** (*int*) – Window_size to use in error calculation.
- **error_buffer** (*int, optional*) – Number of values surrounding an error that are brought into the sequence.
- **y_test** (*np array*) – test targets corresponding to true telemetry values at each timestep *t*.
- **y_hat** (*np array*) – test target predictions at each timestep *t*.
- **e_s** (*list*) – smoothed errors (residuals) between `y_test` and `y_hat`.

Returns Start and end indices for each anomaloues sequence. anom_scores (list): Score for each anomalous sequence.

Return type E_seq (list of tuples)

4.2.4 realseries.utils.evaluation module

Evaluation function.

`realseries.utils.evaluation.`**`adjust_metrics`**(*pred*, *label*, *delay=7*, *beta=1.0*)

Calculating the precison and recall etc. using adjusted label.

Parameters

- **pred** (*ndarray*) – The predicted y.
- **label** (*ndarray*) – The true y label.
- **delay** (*int, optional*) – The max allowed delay of the anomaly occuring. Defaults to 7.
- **beta** (*float, optional*) – The balance between presicion and recall for`` f score``. Defaults to 1.0.

Returns Tuple contains `precision, recall, f1, tp, tn, fp, fn`.

Return type tuple

`realseries.utils.evaluation.`**`adjust_predicts`**(*predict*, *label*, *delay=7*)

Adjust the predicted results.

Parameters

- **predict** (*ndarray*) – The predicted y.
- **label** (*ndarray*) – The true y label.
- **delay** (*int, optional*) – The max allowed delay of the anomaly occuring. Defaults to 7.

Returns The adjusted predicted array y.

Return type naarray

`realseries.utils.evaluation.`**`baseline_oneday`**(*y_true*)

Use the previous value as the predicted value

Parameters **y_true** (*1-D arral_like*) – Auto-regressive inputs.

Returns Evaluation result of one-day ahead baselinesss

Return type dict

`realseries.utils.evaluation.`**`baseline_threeday`**(*y_true*)

Use the average of 3 previous value as the predicted value.

Parameters **y_true** (*aray_like*) – Auto-regressive inputs.

Returns Evaluation result of three-day-ahead-average baseline.

Return type dcit

`realseries.utils.evaluation.`**`evaluate`**(*y_true*, *y_pred*)

Eval metrics. Here 1 stand for anomaly label and 0 is normal samples.

Parameters

- **y_true** (*1-D array_like*) – The actual value.
- **y_pred** (*1-D array_like*) – The predictive value.

Returns a dictionary which includes mse, rmse, mae and r2.

Return type dict

`realseries.utils.evaluation.`**`point_metrics`**(*y_pred*, *y_true*, *beta=1.0*)

Calculate precison recall f1 bny point to point comparison.

Parameters

- **y_pred** (*ndarray*) – The predicted y.
- **y_true** (*ndarray*) – The true y.
- **beta** (*float*) – The balance for calculating *f score*.

Returns Tuple contains `precision, recall, f1, tp, tn, fp, fn`.

Return type tuple

`realseries.utils.evaluation.`**`thres_search`**(*score*, *label*, *num_samples=1000*, *beta=1.0*, *sampling='log'*, *adjust=True*, *delay=7*)

Find the best-f1 score by searching best *threshold*

Parameters

- **score** (*ndarray*) – The anomaly score.
- **label** (*ndarray*) – The true label.
- **num_samples** (*int*) – The number of sample points between `[min_score, max_score]`.
- **beta** (*float, optional*) – The balance between precison and recall in `f score`. Defaults to 1.0.
- **sampling** (*str, optional*) – The sampling method including 'log' and 'linear'. Defaults to 'log'.

Returns Results in best threshold `precison, recall, f1, best_thres, predicted labele`.

Return type tuple

4.2.5 realseries.utils.preprocess module

Preprocess function

`realseries.utils.preprocess.`**`augmentation`**(*data*, *label*, *noise_ratio=0.05*, *noise_interval=0.0005*, *max_length=100000*)

Data augmentation by add anomaly points to origin data.

Parameters

- **data** (*array_like*) – The origin data.
- **label** (*array_like*) – The origin label.
- **noise_ratio** (*float, optional*) – The ratio of adding noise to data. Defaults to 0.05.
- **noise_interval** (*float, optional*) – Noise_interval. Defaults to 0.0005.
- **max_length** (*int, optional*) – The max length of data after augmentation. Defaults to 100000.

`realseries.utils.preprocess.`**`bandpass_cnt`**(*data*, *low_cut_hz*, *high_cut_hz*, *fs*, *filt_order=3*, *axis=0*, *filtfilt=False*)

Bandpass signal applying **causal** butterworth filter of given order.

Parameters

- **data** (*2d-array*) – Time x channels.

- **low_cut_hz** (*float*) – Low cut hz.
- **high_cut_hz** (*float*) – High cut hz.
- **fs** (*float*) – Sample frequency.
- **filt_order** (*int, optional*) – Defaults to 3.
- **axis** (*int, optional*) – Time axis. Defaults to 0.
- **filtfilt** (*bool, optional*) – Whether to use filtfilt instead of lfilter. Defaults to False.

Returns Data after applying bandpass filter.

Return type 2d-array

`realseries.utils.preprocess.`**`exponential_running_demean`**(*data*, *factor_new=0.001*, *init_block_size=None*)

Perform exponential running demeanining. Compute the exponental running mean m_t at time *t* as $m_t = \text{factornew} \cdot mean(x_t) + (1 - \text{factornew}) \cdot m_{t-1}$. Deman the data point x_t at time *t* as: $x'_t = (x_t - m_t)$.

Parameters

- **data** (*2darray*) – Shape is (time, channels)
- **factor_new** (*float, optional*) – Defaults to 0.001.
- **init_block_size** (*int, optional*) – Demean data before to this index with regular demeaning. Defaults to None.

Returns Demeaned data (time, channels).

Return type 2darray

`realseries.utils.preprocess.`**`exponential_running_standardize`**(*data*, *factor_new=0.001*, *init_block_size=None*, *eps=0.0001*)

Perform exponential running standardization.

Compute the exponental running mean m_t at time *t* as $m_t = \text{factornew} \cdot mean(x_t) + (1 - \text{factornew}) \cdot m_{t-1}$.

Then, compute exponential running variance v_t at time *t* as $v_t = \text{factornew} \cdot (m_t - x_t)^2 + (1 - \text{factornew}) \cdot v_{t-1}$.

Finally, standardize the data point x_t at time *t* as: $x'_t = (x_t - m_t)/max(\sqrt{v_t}, eps)$.

Parameters

- **data** (*2darray*) – The shape is (time, channels)
- **factor_new** (*float, optional*) – Defaults to 0.001.
- **init_block_size** (*int, optional*) – Standardize data before to this index with regular standardization. Defaults to None.
- **eps** (*float, optional*) – Stabilizer for division by zero variance.. Defaults to 1e-4.

Returns Standardized data (time, channels).

Return type 2darray

`realseries.utils.preprocess.`**`filter_is_stable`**(*a*)

Check if filter coefficients of IIR filter are stable.

Parameters **a** (*list*) – list or 1darray of number. Denominator filter coefficients a.

Returns Filter is stable or not.

Return type bool

Notes

Filter is stable if absolute value of all roots is smaller than 1, see http://stackoverflow.com/a/8812737/1469195.

`realseries.utils.preprocess.highpass_cnt(data, low_cut_hz, fs, filt_order=3, axis=0)`

signal applying **causal** butterworth filter of given order.

Parameters

- **data** (*2d-array*) – Time x channels.
- **low_cut_hz** (*float*) – Low cut frequency HZ.
- **fs** (*float*) – Sample frequency.
- **filt_order** (*int*) – Defaults to 3.
- **axis** (*int, optional*) – Time axis. Defaults to 0.

Returns Data after applying highpass filter.

Return type highpassed_data (2d-array)

`realseries.utils.preprocess.lowpass_cnt(data, high_cut_hz, fs, filt_order=3, axis=0)`

Lowpass signal applying **causal** butterworth filter of given order.

Parameters

- **data** (*2d-array*) – Time x channels.
- **high_cut_hz** (*[type]*) – High cut frequency.
- **fs** (*[type]*) – Sample frequency.
- **filt_order** (*int, optional*) – Defaults to 3.

Returns Data after applying lowpass filter.

Return type 2d-array

`realseries.utils.preprocess.normalization(X)`

Normalization to [0, 1] on each column data of input array.

Parameters **X** (*array_like*) – The input array for formalization.

Returns Normalized array in [0, 1].

Return type ndarray

`realseries.utils.preprocess.standardization(X)`

Standardization each column data by reduce mean and divide std.

Parameters **X** (*array_like*) – The input array for standardization.

Returns Standardized array with 0 mean and 1 std.

Return type ndarray

4.2.6 realseries.utils.segment module

Segment function.

class realseries.utils.segment.**BatchSegment**(*series_length*, *window_size*, *batch_size*, *shuffle=False*, *discard_last_batch=False*)

Bases: object

[summary]

Parameters

- **series_length** (*int*) – Series length.
- **window_size** (*int*) – Window size.
- **batch_size** (*int*) – Batch size.
- **shuffle** (*bool, optional*) – Defaults to False.
- **discard_last_batch** (*bool, optional*) – If the last batch not complete, ignore it. Defaults to False.

Raises

- **ValueError** – Window_size must larger than 1.
- **ValueError** – Window_size must smaller than series_length

get_iterator(*arrays*)

Get data iterator for input sequences.

Parameters **arrays** (*list*) – Contain the data to be iterated, which with the same length.

Yields *tuple* – Contain the sliding window data, which has the same order as param: arrays.

realseries.utils.segment.**slice_generator**(*series_length*, *batch_size*, *discard_last_batch=False*)

Generate slices for series-like data

Parameters

- **series_length** (*int*) – Series length.
- **batch_size** (*int*) – Batch size.
- **discard_last_batch** (*bool, optional*) – If the last batch not complete, ignore it. Defaults to False.

Yields slice

4.2.7 realseries.utils.utility module

function like save load model, early stop in model training.

class realseries.utils.utility.**EarlyStopping**(*monitor='val_loss'*, *patience=7*, *delta=0*, *verbose=False*)

Bases: object

Early stops the training if validation loss doesn't improve after a given patience.

Parameters

- **patience** (*int*) – How long to wait after last time validation loss improved. Default to 7
- **verbose** (*bool*) – If True, prints a message for each validation loss improvement. Default to False

- **delta** (*float*) – Minimum change in the monitored quantity to qualify as an improvement. Default to 0.

save_checkpoint(*value*, *model*)
Saves checkpoint when validation loss decrease.

Parameters

- **value** (*float*) – The value of new validation loss.
- **model** (*model*) – The current better model.

class realseries.utils.utility.**aleatoric_loss**
Bases: torch.nn.modules.module.Module

The negative log likelihood (NLL) loss.

Parameters

- **gt** – the ground truth
- **pred_mean** – the predictive mean
- **logvar** – the log variance

loss
the nll loss result for the regression.

forward(*gt*, *pred_mean*, *logvar*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

training: bool

realseries.utils.utility.**load_model**(*model*, *path*)
Load Pytorch model.

Parameters

- **model** (*pytorch model*) – The initialized pytorch model.
- **model_path** (*string or path*) – Path for loading model.

Returns The loaded model.

Return type model

class realseries.utils.utility.**mmd_loss**
Bases: torch.nn.modules.module.Module

The mmd loss.

Parameters

- **source_features** – the ground truth
- **target_features** – the prediction value

loss_value
the nll loss result for the regression.

forward(*source_features*, *target_features*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

gaussian_kernel_matrix(*x*, *y*, *sigmas*)

maximum_mean_discrepancy(*x*, *y*, *kernel=<function mmd_loss.gaussian_kernel_matrix>*)

pairwise_distance(*x*, *y*)

training: bool

realseries.utils.utility.**save_model**(*model*, *model_path*)

Save pytorch model.

Parameters

- **model** (*pytorch model*) – The trained pytorch model.
- **model_path** (*string or path*) – Path for saving model.

4.2.8 realseries.utils.visualize module

Plot the data.

realseries.utils.visualize.**mat_plot**(*X*, *y*, *fig_size=(15, 10)*, *title=None*, *if_save=False*, *name=None*)

Plot array X and y.

Parameters

- **X** (*1darray*) – Array 1.
- **y** (*1darray*) – Array 2.
- **fig_size** (*tuple, optional*) – Size of the figure. Defaults to (15, 10).
- **title** (*str, optional*) – Figure title.. Defaults to None.
- **if_save** (*bool, optional*) – Whether or not save.. Defaults to False.
- **name** (*Str, optional*) – Save figure name.. Defaults to None.

realseries.utils.visualize.**pd_plot**(*tab*, *fig_size=(15, 10)*, *cols=None*, *title=None*, *if_save=False*, *name=None*)

Plot time series for pandas data.

Parameters

- **tab** – Pandas file.
- **fig_size** – Figure size.
- **cols** – Specify which cols to plot.
- **title** – Figure title.
- **if_save** – Whether or not save.
- **name** – Save figure name.

`realseries.utils.visualize.plot_anom(pd_data_label, pred_anom, pred_score, fig_size=(9, 5), if_save=False, name=None)`

Visualize origin time series and predicted result.

Parameters

- **pd_data_label** (*dataframe*) – Pandas dataframe and the last column is label.
- **pred_anom** (*1darray*) – The predicted label.
- **pred_score** (*1darray*) – The predicted anomaly score.
- **fig_size** (*tuple, optional*) – Figure size. Defaults to (9, 5).
- **if_save** (*bool, optional*) – Whether to save or not. Defaults to False.
- **name** (*str, optional*) – Save file name. Defaults to None.

`realseries.utils.visualize.plot_mne(X, columns=None, sfreq=1, duration=1000, start=0, n_channels=20, scalings='auto', ch_types=None, color=None, highpass=None, lowpass=None, filtorder=4)`

plot mne raw data

Parameters

- **X** (*numpy array*) – data with shape (n_samples, n_features)
- **columns** (*list, optional*) – the string name or ID of each column features
- **sfreq** (*int, optional*) – sample frequency. Defaults to 1.
- **duration** (*int, optional*) – Time window (s) to plot in the frame for showing. The lesser of this value and the duration of the raw file will be used. Defaults to 1000.
- **start** (*int, optional*) – The start time to show. Defaults to 0.
- **n_channels** (*int, optional*) – num of channels to show in one frame. Defaults to 20.
- **scalings** (*dict, optional*) – Scaling factors for the traces. If any fields in scalings are 'auto', the scaling factor is set to match the 99.5th percentile of a subset of the corresponding data. If scalings == 'auto', all scalings fields are set to 'auto'. If any fields are 'auto' and data is not preloaded, a subset of times up to 100mb will be loaded. If None, defaults to:

```
dict(mag=1e-12, grad=4e-11, eeg=20e-6, eog=150e-6, ecg=5e-4,
emg=1e-3, ref_meg=1e-12, misc=1e-3, stim=1,
resp=1, chpi=1e-4, whitened=1e2).
```

  The larger the scale is, the amplitudes of this channel will zoom smaller.

- **color** (*dict | color object, optional*) – Color for the data traces. If None, defaults to:

```
dict(mag='darkblue', grad='b', eeg='k', eog='k', ecg='m',
emg='k', ref_meg='steelblue', misc='k', stim='k',
resp='k', chpi='k'). Defaults to None.
```

- **ch_types** (*list, optional*) – Definition of channel types like ['eeg', 'eeg', 'eeg', 'ecg']. It can be used to change the color of each channel by setting *color*. Defaults to None.
- **highpass** (*float, optional*) – Highpass to apply when displaying data. Defaults to None.
- **lowpass** (*float, optional*) – Lowpass to apply when displaying data. If *highpass* > *lowpass*, a bandstop rather than bandpass filter will be applied. Defaults to None.

- **filtorder** (*int, optional*) – 0 will use FIR filtering with MNE defaults. Other values will construct an IIR filter of the given order. This parameter will work when *lowpass* or *highpass* is not None. Defaults to 4.

Returns Instance of matplotlib.figure.Figure

Return type fig

CHAPTER

FIVE

TIME SERIES DATASETS

RealSeries provides several example datasets that can be used for *Forecast with Uncertainty*, and *Anomaly Detection*, *Granger causality*.

5.1 Forecast Datasets

Dir	Name	External	channel	HNN		MC_dropout		CRMMD	
				RMSE	EPIW	RMSE	EPIW	RMSE	EPIW
Forecast_data	air_quality	https://archive.ics.uci.edu/ml/datasets/Air+Quality	muti	79.60	0.058	81.16	0.339	80.69	0.010
	bike_sharing	https://archive.ics.uci.edu/ml/datasets/bike+sharing+dataset	muti	40.71	0.054	38.86	0.258	37.93	0.006
	metro_traffic	https://archive.ics.uci.edu/ml/datasets/Metro+Interstate+Traffic+Volum	muti	556.3	0.102	523.6	0.304	545.5	0.017
	pm25	https://archive.ics.uci.edu/ml/datasets/Beijing+PM2.5+Data	muti	58.81	0.022	70.95	0.331	57.43	0.010

5.2 Anomaly Detection Datasets

Dir	Name	External	paper	channel	Lumino			SR_CNN			IForest			LSTM_dym			Rrcf			VAE		
					pre	rec	f1	pre	rec	f1	pre	rec	f1	pre	rec	f1	pre	rec	f1	pre	rec	f1
realKnownCause	nyc_taxi	https://github.com/numenta/NAB/tree/master/data/realKnownCause	[RCF] [NAB]	single	0.99	0.19	0.33	0.99	0.59	0.75	0.99	0.39	0.57	0.99	0.39	0.57	0.99	0.19	0.33	0.99	0.79	0.88
realTweets	UPS	https://github.com/numenta/NAB/tree/master/data/realTweets	[NAB] [NAB]	single	0	0	0	0.98	0.99	0.99	0	0	0	0.73	0.99	0.95	0	0	0	0.85	0.99	0.92
	FE	https://github.com/numenta/NAB/tree/master/data/realTweets	[NAB] [NAB]	single	0	0	0	0.37	0	0.01	0.98	0.99	0.99	0.87	0.99	0.93	0.99	0.99	0.99	0.96	0.99	0.98
	KO	https://github.com/numenta/NAB/tree/master/data/realTweets	[NAB] [NAB]	single	0	0	0	0.99	0.49	0.66	0.99	0.49	0.66	0	0	0	0.99	0.01	0.01	0	0	0
	IBM	https://github.com/numenta/NAB/tree/master/data/realTweets	[NAB] [NAB]	single	0	0	0	0.99	0.99	0.99	0	0	0	0.99	0.61	0.76	0	0	0	0	0	0
	GOOG	https://github.com/numenta/NAB/tree/master/data/realTweets	[NAB] [NAB]	single	0	0	0	0	0	0	0.99	0.99	0.99	0	0	0	0	0	0	0.99	0.99	0.99

5.3 Granger causality Datasets

datasets		External	GC		DWGC	
			accuracy	recall	accuracy	recall
NAR simulation	window length=10	https://github.com/ZHzhang01/DWGC/tree/master/data	0.42	0.58	0.44	0.73
	window length=20		0.76	0.65	0.80	0.65
	window length=30		0.93	0.66	0.94	0.67
	window length=100		1	0.86	1	0.88
ENSO data(DWGC)			Notebook: http://git.real-ai.cn/realseries/realseries/blob/causality_branch/notebooks/DWGC.ipynb			
ENSO data(GC)			Notebook: http://git.real-ai.cn/realseries/realseries/blob/causality_branch/notebooks/GC.ipynb			

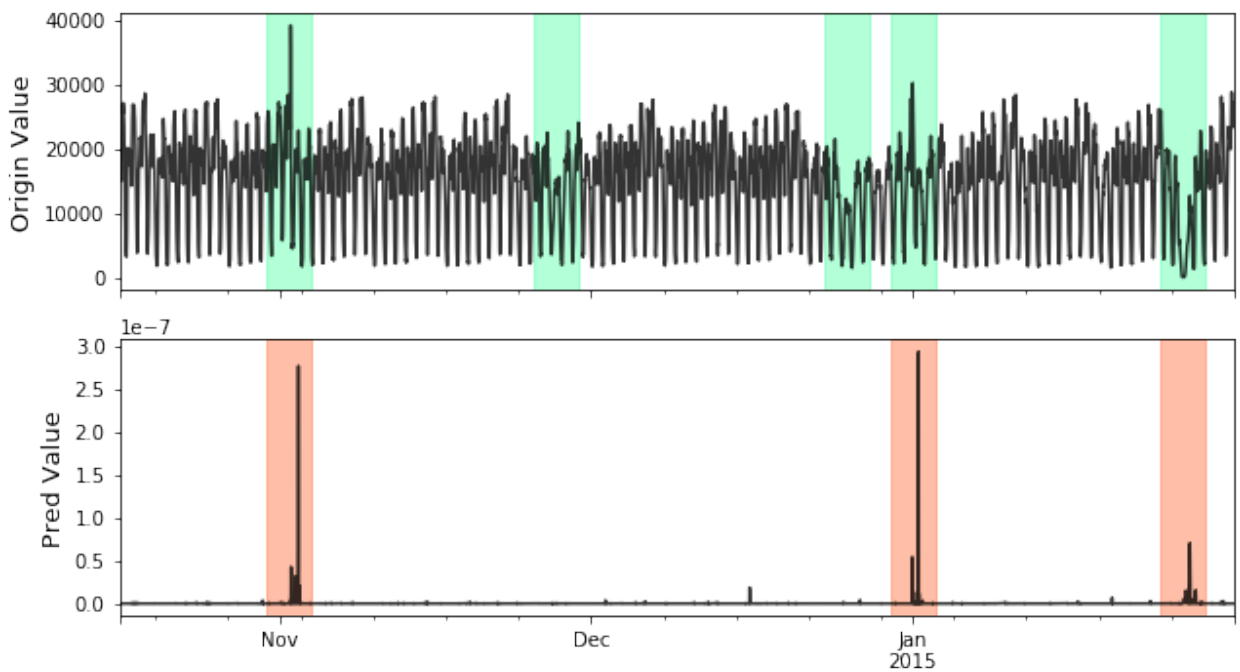
CHAPTER

SIX

ANOMALY DETECTION

6.1 Problem Description

Time Series Anomaly Detection's task is to find out the possible anomalies lies in time series, like this:



To formalize, give a time series with either single channel or multi channels:

$$\mathbf{X} = \{x_{ij}\} = \begin{Bmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ \cdots & & & \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{Bmatrix}, i\text{: channel id}, j\text{: index id.}$$

We aim to find the possible anomalies lies in X:

$$x_{pq}$$

6.2 Models

Abbr	Algorithm	type	supervise	Year	Class	Ref
IForest	Isolation forest.	outlier ensembles	unsupervised	2008	`realseries.models.iforest.IForest`	[IForest]
LSTM_dynamic	LSTM based nonparametric anomaly thresholding.	neural networks	unsupervised	2018	`realseries.models.lstm_dynamic.LSTM_dynamic`	[LSTM_dynamic]
Lumino	A light weight library luminol.	distance based	unsupervised	2017	`realseries.models.lumino.Lumino`	[Lumino]
RCForest	Robust random cut forest.	outlier ensembles	unsupervised	2016	`realseries.models.rcforest.RCForest`	[RCForest]
LSTMED	RNN based time-series prediction.	neural networks	unsupervised	2016	`realseries.models.rnn.LSTMED`	[LSTMED]
SR_CNN	Spectral Residual and ConvNet based anomaly detection.	neural networks	supervised	2019	`realseries.models.srcnn.SR_CNN`	[SR_CNN]
STL	Seasonal and Trend decomposition using Loess.	weighted regression	unsupervised	1990	`realseries.models.stl.STL`	[STL]
VAE	Variational Auto-Encoder anomaly detection.	probabilistic	unsupervised	2018	`realseries.models.vae_ad.VAE_AD`	[VAE]

References

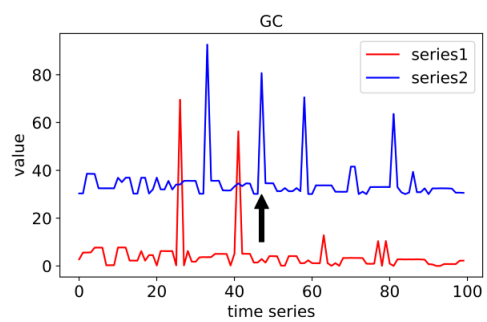
CHAPTER

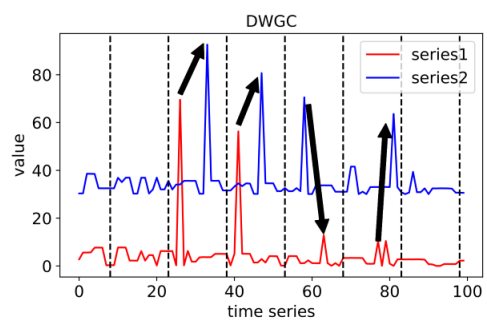
SEVEN

GRANGER CAUSALITY

7.1 Problem Description of Granger causality

Granger causality Detection's task is to find out the Granger causality lies in multi-channel time series on window level or channel level, like this:





To formalize, give a time series with multi channels:

$$\mathbf{X} = \{x_{ij}\} = \begin{Bmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ \cdots & & & \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{Bmatrix}, i\text{: channel id}, j\text{: index id.}$$

We aim to find the Granger causality lies in channel level:

We also try to pinpoint causality from the channel level to the point-to-point level:

$$\{x_i\} \Longrightarrow \{x_j\}$$
$$\{x_{it_1}\} \Longrightarrow \{x_{jt_2}\}$$

where 2 time points are in the same window index.

7.2 Models of Granger causality

Abbr	Algorithm	level	supervise	Year	Class	Ref
GC	Granger causality	channel level	unsupervised	1969	`realseries.models.GC.GC`	[GC]
DWGC	Dynamic window-level Granger causality	window level	unsupervised	2020(original)	`realseries.models.DWGC.DWGC`	[DWGC]

References

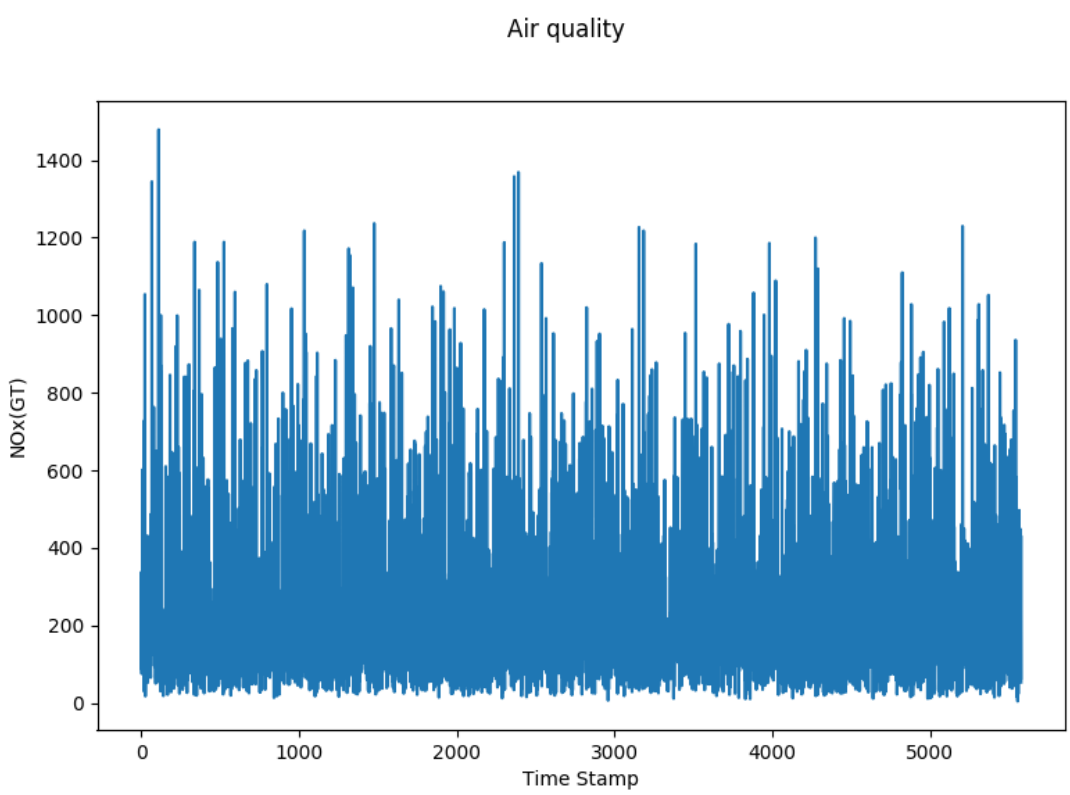
CHAPTER

EIGHT

FORECAST WITH UNCERTAINTY

8.1 Problem Description

Time Series Forecast's task is to model multiple regression sub-problems in sequence, and the tendency along the sliding windows can beused to examine the obtained predictive uncertainty. Forecasting involves taking models fit on historical data and using them to predict future observations.



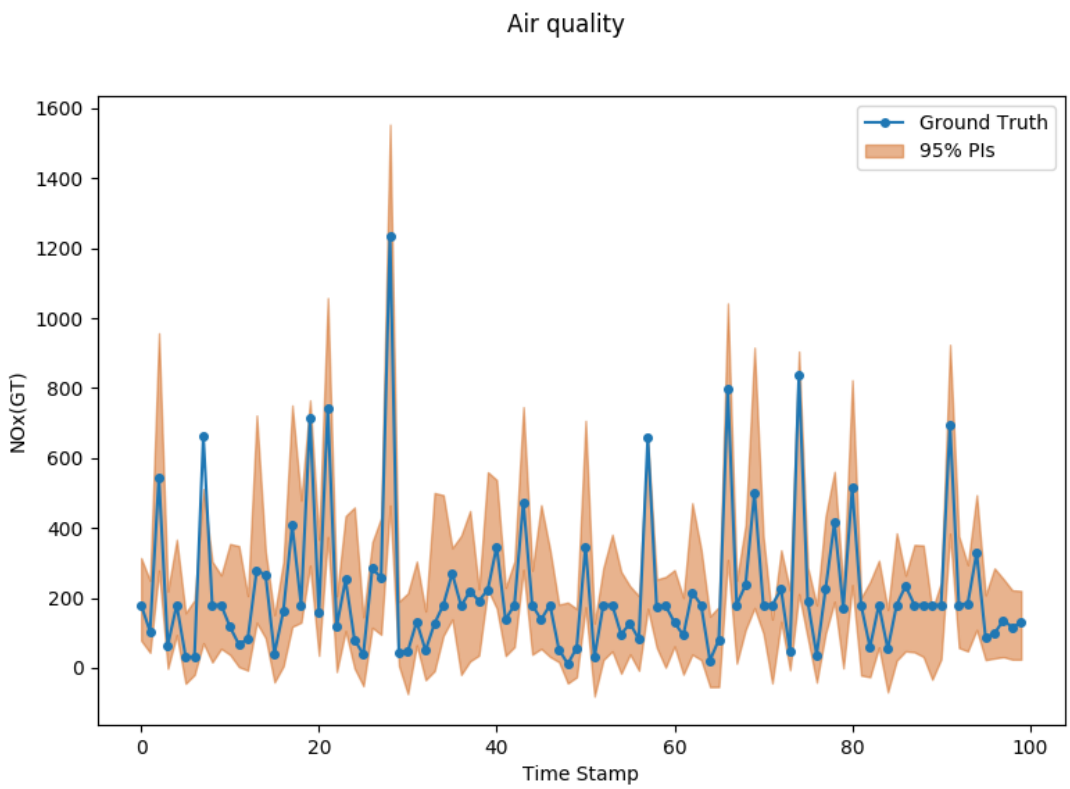
To formalize, give a time series with either single channel or multi channels:

$$Y = \{\boldsymbol{y_1}, \boldsymbol{y_2}, \ldots, \boldsymbol{y_T}\} \text{ where } \boldsymbol{y_t} \in \mathbb{R}^n$$

where n is the variable dimension or channels, we aim at predicting a series of future signals in a rolling forecasting fashion. That being said, to predict y_{T+h}, where h is the desirable horizon ahead of the current time stamp, we assume $\{\boldsymbol{y_1}, \boldsymbol{y_2}, \ldots, \boldsymbol{y_T}\}$ are available. Likewise, to predict the value of the next time stamp y_{T+h+1}, we

assume $\{\boldsymbol{y_1}, \boldsymbol{y_2}, \ldots, \boldsymbol{y_T}, \boldsymbol{y_{T+1}}\}$ are available. We hence formulate the input matrix at time stamp T as $X_T = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_T\} \in \mathbb{R}^{n \times T}$.

Moreover, we also want to know the uncertainty of predictive distribution, a calibrated forecaster outputs the cumulative distribution function (CDF) F_i by the predictive distribution for each input x_i. Generally, we use predictive intervals (PIs) to represent the uncertainty. For example, given the probability 95%, the forecaster should output the 95% prediction interval.



8.2 Models

Abbr	Algorithm	type	supervise	Year	Class	Ref
CRMMD	Using maximum mean discrepancy (MMD) to perform distribution matching.	kernel distance, two-step	supervised	2020	`realseries.models.crmmd.CRMMD`	[CRMMD]
HNN	Heteroscedastic neural networks output the mean and variance to obtain the uncertainty.	neural networks	supervised	2017	`realseries.models.hnn.HNN`	[HNN]
MC-dropout	Approximate Bayesian inference in deep Gaussian process.	Bayesian approximation	supervised	2016	`realseries.models.mc_dropout.MC_dropout`	[MC-dropout]

References

CHAPTER

NINE

CONTRIBUTION

This project is headed by Wenbo Hu.

Other core members include:

- Xianrui Zhang (Ph.D Student @ BUAA)
- Wenkai Li (Bacholar Student @ CS of Tsinghua)
- Zhiheng Zhang (Bacholar Student @ BUPT)
- Peng Cui (Master @ CS of Tsinghua)

CHAPTER

TEN

HOW TO CONTRIBUTE

Any contribution are welcomed. Here is a guide of how to conribute a spicific model to realseries.

Since RealSeries uses a split-joint structure to formulize the lib. To contribute a model is to follow the following steps:

1. create yourmodel.py in realseries/models;
2. inherite base.py to yourmodel.py;
3. develop your model;
4. write a jupyter notebook example and a documentation.

CHAPTER

ELEVEN

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[IForest] Liu F T, Ting K M, Zhou Z H. Isolation forest[C]//2008 Eighth IEEE International Conference on Data Mining. IEEE, 2008: 413-422.

[LSTM_dynamic] Hundman K, Constantinou V, Laporte C, et al. Detecting spacecraft anomalies using lstms and non-parametric dynamic thresholding[C]//Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2018: 387-395.

[Lumino] Luminol. Github, https://github.com/linkedin/luminol

[RCForest] S. Guha, N. Mishra, G. Roy, & O. Schrijvers, Robust random cut forest based anomaly detection on streams, in Proceedings of the 33rd International conference on machine learning, New York, NY, 2016 (pp. 2712-2721).

[LSTMED] Malhotra P, Ramakrishnan A, Anand G, et al. LSTM-based encoder-decoder for multi-sensor anomaly detection[J]. arXiv preprint arXiv:1607.00148, 2016.

[SR_CNN] Ren H, Xu B, Wang Y, et al. Time-Series Anomaly Detection Service at Microsoft[C]//Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2019: 3009-3017.

[STL] Cleveland R B. STL: A seasonal-trend decomposition procedure based on loess. 1990[J]. DOI: citeulike-article-id, 1435502.

[VAE] Xu H, Chen W, Zhao N, et al. Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications[C]//Proceedings of the 2018 World Wide Web Conference. 2018: 187-196.

[Robust] Su Y, Zhao Y, Niu C, et al. Robust Anomaly Detection for Multivariate Time Series through Stochastic Recurrent Neural Network[C]//Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2019: 2828-2837.

[NAB] Ahmad S, Lavin A, Purdy S, et al. Unsupervised real-time anomaly detection for streaming data[J]. Neurocomputing, 2017, 262: 134-147.

[NAB2] Lavin A, Ahmad S. Evaluating Real-Time Anomaly Detection Algorithms–The Numenta Anomaly Benchmark[C]//2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA). IEEE, 2015: 38-44.

[GC] Clive WJ Granger. Investigating causal relations by econometric models and cross-spectral methods. Econometrica: journal of the Econometric Society, pages 424–438, 1969.

[DWGC] Dynamic Window-level Granger Causality of Multi-channel Time Series. Zhiheng Zhang, Wenbo Hu, Tian Tian, Jun Zhu. arXiv:2006.07788. 2020.

[CRMMD] Peng Cui, Wenbo Hu and Jun Zhu. Calibrated Reliable Regression using Maximum Mean Discrepancy. arXiv: 2006.10255.

[HNN] Alex Kendall and Yarin Gal. What uncertainties do we need in Bayesian deep learning forcomputer vision?Advances in neural information processing systems, pages 5574–5584, 2017.

[MC-dropout] Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representingmodel uncertainty in deep learning. Ininternational conference on machine learning, pages1050–1059, 2016.

PYTHON MODULE INDEX

r

realseries.models.AR, 25
realseries.models.base, 28
realseries.models.crmmd, 43
realseries.models.DWGC, 26
realseries.models.GC, 26
realseries.models.hnn, 45
realseries.models.iforest, 29
realseries.models.lstm_dynamic, 30
realseries.models.lumino, 32
realseries.models.mc_dropout, 47
realseries.models.NAR, 27
realseries.models.rcforest, 33
realseries.models.rnn, 34
realseries.models.seqvl, 36
realseries.models.sr, 36
realseries.models.srcnn, 38
realseries.models.stl, 39
realseries.models.vae_ad, 40
realseries.models.vae_dense, 41
realseries.utils.data, 49
realseries.utils.dataset, 51
realseries.utils.errors, 52
realseries.utils.evaluation, 53
realseries.utils.preprocess, 54
realseries.utils.segment, 57
realseries.utils.utility, 57
realseries.utils.visualize, 59

INDEX

A

adjust_metrics() (*in module realseries.utils.evaluation*), 53
adjust_predicts() (*in module realseries.utils.evaluation*), 53
aleatoric_loss (*class in realseries.utils.utility*), 58
anomaly_score (*realseries.models.iforest.IForest attribute*), 29
AR (*class in realseries.models.AR*), 25
augmentation() (*in module realseries.utils.preprocess*), 54

B

bandpass_cnt() (*in module realseries.utils.preprocess*), 54
baseline_oneday() (*in module realseries.utils.evaluation*), 53
baseline_threeday() (*in module realseries.utils.evaluation*), 53
BaseModel (*class in realseries.models.base*), 28
BatchSegment (*class in realseries.utils.segment*), 57

C

calc_seasonal() (*realseries.models.stl.STL static method*), 39
calc_trend() (*realseries.models.stl.STL static method*), 39
CRMMD (*class in realseries.models.crmmd*), 43

D

Data (*class in realseries.utils.dataset*), 51
data2supervised() (*realseries.utils.dataset.Data method*), 51
data_iterator() (*realseries.utils.dataset.Data method*), 51
data_to_seqvl_format() (*realseries.utils.dataset.Data method*), 51
detect() (*realseries.models.AR.AR method*), 25
detect() (*realseries.models.base.BaseModel method*), 28
detect() (*realseries.models.DWGC.DWGC method*), 26
detect() (*realseries.models.GC.GC method*), 26
detect() (*realseries.models.iforest.IForest method*), 30
detect() (*realseries.models.lstm_dynamic.LSTM_dynamic method*), 31
detect() (*realseries.models.lumino.Lumino method*), 32
detect() (*realseries.models.NAR.NAR_Network method*), 27
detect() (*realseries.models.rcforest.RCForest method*), 34
detect() (*realseries.models.rnn.LSTMED method*), 35
detect() (*realseries.models.seqvl.SeqVL method*), 36
detect() (*realseries.models.sr.SpectralResidual method*), 37
detect() (*realseries.models.srcnn.SR_CNN method*), 38
detect() (*realseries.models.stl.STL method*), 39
detect() (*realseries.models.vae_ad.VAE_AD method*), 41
detect() (*realseries.models.vae_dense.VAE_Dense method*), 42
drift() (*realseries.models.stl.STL static method*), 39
DWGC (*class in realseries.models.DWGC*), 26

E

EarlyStopping (*class in realseries.utils.utility*), 57
estimators_ (*realseries.models.iforest.IForest attribute*), 29
estimators_ (*realseries.models.iforest.IForest property*), 30
estimators_samples_ (*realseries.models.iforest.IForest attribute*), 29
estimators_samples_ (*realseries.models.iforest.IForest property*), 30
evaluate() (*in module realseries.utils.evaluation*), 53
evaluation_model() (*realseries.models.crmmd.CRMMD method*), 43
evaluation_model() (*realseries.models.hnn.HNN method*), 45
evaluation_model() (*realseries.models.mc_dropout.MC_dropout method*), 47

exponential_running_demean() (*in module realseries.utils.preprocess*), 55
exponential_running_standardize() (*in module realseries.utils.preprocess*), 55
extend_series() (*realseries.models.sr.SpectralResidual static method*), 37

F

filter_is_stable() (*in module realseries.utils.preprocess*), 55
fit() (*realseries.models.AR.AR method*), 25
fit() (*realseries.models.base.BaseModel method*), 28
fit() (*realseries.models.crmmd.CRMMD method*), 44
fit() (*realseries.models.DWGC.DWGC method*), 26
fit() (*realseries.models.GC.GC method*), 27
fit() (*realseries.models.hnn.HNN method*), 46
fit() (*realseries.models.iforest.IForest method*), 30
fit() (*realseries.models.lstm_dynamic.LSTM_dynamic method*), 31
fit() (*realseries.models.lumino.Lumino method*), 33
fit() (*realseries.models.mc_dropout.MC_dropout method*), 48
fit() (*realseries.models.NAR.NAR_Network method*), 27
fit() (*realseries.models.rcforest.RCForest method*), 34
fit() (*realseries.models.rnn.LSTMED method*), 35
fit() (*realseries.models.seqvl.SeqVL method*), 36
fit() (*realseries.models.sr.SpectralResidual method*), 37
fit() (*realseries.models.srcnn.SR_CNN method*), 38
fit() (*realseries.models.stl.STL method*), 39
fit() (*realseries.models.vae_ad.VAE_AD method*), 41
fit() (*realseries.models.vae_dense.VAE_Dense method*), 42
flatten() (*realseries.models.vae_dense.VAE_Dense method*), 42
forecast() (*realseries.models.base.BaseModel method*), 28
forecast() (*realseries.models.crmmd.CRMMD method*), 44
forecast() (*realseries.models.hnn.HNN method*), 46
forecast() (*realseries.models.mc_dropout.MC_dropout method*), 48
forecast() (*realseries.models.stl.STL method*), 40
forecast() (*realseries.models.vae_ad.VAE_AD method*), 41
forecast() (*realseries.models.vae_dense.VAE_Dense method*), 42
forward() (*realseries.utils.utility.aleatoric_loss method*), 58
forward() (*realseries.utils.utility.mmd_loss method*), 58

G

gaussian_kernel_matrix() (*realseries.utils.utility.mmd_loss method*), 59
GC (*class in realseries.models.GC*), 26
generate_arma_data() (*in module realseries.utils.data*), 49
generate_spectral_score() (*realseries.models.sr.SpectralResidual method*), 37
get_errors() (*in module realseries.utils.errors*), 52
get_iterator() (*realseries.utils.segment.BatchSegment method*), 57

H

highpass_cnt() (*in module realseries.utils.preprocess*), 56
HNN (*class in realseries.models.hnn*), 45

I

IF (*realseries.models.iforest.IForest attribute*), 29
IForest (*class in realseries.models.iforest*), 29
impute() (*realseries.models.base.BaseModel method*), 28

L

load() (*realseries.models.base.BaseModel method*), 28
load() (*realseries.models.vae_ad.VAE_AD method*), 41
load() (*realseries.models.vae_dense.VAE_Dense method*), 42
load_data() (*realseries.utils.dataset.Data method*), 51
load_exp_data() (*in module realseries.utils.data*), 50
load_model() (*in module realseries.utils.utility*), 58
load_model() (*realseries.models.crmmd.CRMMD method*), 44
load_model() (*realseries.models.hnn.HNN method*), 46
load_model() (*realseries.models.mc_dropout.MC_dropout method*), 49
load_NAB() (*in module realseries.utils.data*), 49
load_SMD() (*in module realseries.utils.data*), 50
load_split_NASA() (*in module realseries.utils.data*), 50
load_splitted_RNN() (*in module realseries.utils.data*), 50
load_Yahoo() (*in module realseries.utils.data*), 50
load_yahoo() (*realseries.utils.dataset.Data method*), 51
loss (*realseries.utils.utility.aleatoric_loss attribute*), 58
loss_value (*realseries.utils.utility.mmd_loss attribute*), 58
lowpass_cnt() (*in module realseries.utils.preprocess*), 56
LSTM_dynamic (*class in realseries.models.lstm_dynamic*), 30
LSTMED (*class in realseries.models.rnn*), 34

Lumino (*class in realseries.models.lumino*), 32

M

mat_plot() (*in module realseries.utils.visualize*), 59
max_samples_ (*realseries.models.iforest.IForest attribute*), 30
max_samples_ (*realseries.models.iforest.IForest property*), 30
maximum_mean_discrepancy() (*realseries.utils.utility.mmd_loss method*), 59
MC_dropout (*class in realseries.models.mc_dropout*), 47
mean() (*realseries.models.stl.STL static method*), 40
mmd_loss (*class in realseries.utils.utility*), 58
model (*realseries.models.crmmd.CRMMD attribute*), 43, 44
model (*realseries.models.hnn.HNN attribute*), 45, 46
model (*realseries.models.lstm_dynamic.LSTM_dynamic attribute*), 31
model (*realseries.models.mc_dropout.MC_dropout attribute*), 47, 48
model (*realseries.models.rnn.LSTMED attribute*), 35
model (*realseries.models.srcnn.SR_CNN attribute*), 38
model (*realseries.models.vae_ad.VAE_AD attribute*), 40
model (*realseries.models.vae_dense.VAE_Dense attribute*), 42
module
    realseries.models.AR, 25
    realseries.models.base, 28
    realseries.models.crmmd, 43
    realseries.models.DWGC, 26
    realseries.models.GC, 26
    realseries.models.hnn, 45
    realseries.models.iforest, 29
    realseries.models.lstm_dynamic, 30
    realseries.models.lumino, 32
    realseries.models.mc_dropout, 47
    realseries.models.NAR, 27
    realseries.models.rcforest, 33
    realseries.models.rnn, 34
    realseries.models.seqvl, 36
    realseries.models.sr, 36
    realseries.models.srcnn, 38
    realseries.models.stl, 39
    realseries.models.vae_ad, 40
    realseries.models.vae_dense, 41
    realseries.utils.data, 49
    realseries.utils.dataset, 51
    realseries.utils.errors, 52
    realseries.utils.evaluation, 53
    realseries.utils.preprocess, 54
    realseries.utils.segment, 57
    realseries.utils.utility, 57
    realseries.utils.visualize, 59

N

naive() (*realseries.models.stl.STL static method*), 40
NAR_Network (*class in realseries.models.NAR*), 27
normalization() (*in module realseries.utils.preprocess*), 56
normalize() (*realseries.utils.dataset.Data method*), 51

O

obtain_anomaly() (*realseries.models.lstm_dynamic.LSTM_dynamic static method*), 31

P

pairwise_distance() (*realseries.utils.utility.mmd_loss method*), 59
pd_plot() (*in module realseries.utils.visualize*), 59
plot_anom() (*in module realseries.utils.visualize*), 59
plot_mne() (*in module realseries.utils.visualize*), 60
point_metrics() (*in module realseries.utils.evaluation*), 53
predict() (*realseries.models.lstm_dynamic.LSTM_dynamic method*), 32
predict() (*realseries.models.vae_ad.VAE_AD method*), 41
predict() (*realseries.models.vae_dense.VAE_Dense method*), 42
predict_next() (*realseries.models.sr.SpectralResidual static method*), 37
process_errors() (*in module realseries.utils.errors*), 52

R

RCForest (*class in realseries.models.rcforest*), 33
realseries.models.AR
    module, 25
realseries.models.base
    module, 28
realseries.models.crmmd
    module, 43
realseries.models.DWGC
    module, 26
realseries.models.GC
    module, 26
realseries.models.hnn
    module, 45
realseries.models.iforest
    module, 29
realseries.models.lstm_dynamic
    module, 30
realseries.models.lumino
    module, 32
realseries.models.mc_dropout
    module, 47

realseries.models.NAR
module, 27
realseries.models.rcforest
module, 33
realseries.models.rnn
module, 34
realseries.models.seqvl
module, 36
realseries.models.sr
module, 36
realseries.models.srcnn
module, 38
realseries.models.stl
module, 39
realseries.models.vae_ad
module, 40
realseries.models.vae_dense
module, 41
realseries.utils.data
module, 49
realseries.utils.dataset
module, 51
realseries.utils.errors
module, 52
realseries.utils.evaluation
module, 53
realseries.utils.preprocess
module, 54
realseries.utils.segment
module, 57
realseries.utils.utility
module, 57
realseries.utils.visualize
module, 59
reform() (*realseries.models.vae_dense.VAE_Dense method*), 42
reshape_for_test() (*realseries.models.seqvl.SeqVL method*), 36
reshape_for_training() (*realseries.models.seqvl.SeqVL method*), 36

S

save() (*realseries.models.base.BaseModel method*), 28
save() (*realseries.models.vae_ad.VAE_AD method*), 41
save() (*realseries.models.vae_dense.VAE_Dense method*), 42
save_checkpoint() (*realseries.utils.utility.EarlyStopping method*), 58
save_model() (*in module realseries.utils.utility*), 59
save_model() (*realseries.models.crmmd.CRMMD method*), 45
save_model() (*realseries.models.hnn.HNN method*), 47
save_model() (*realseries.models.mc_dropout.MC_dropout method*), 49
SeqVL (*class in realseries.models.seqvl*), 36
slice_generator() (*in module realseries.utils.segment*), 57
spectral_residual_transform() (*realseries.models.sr.SpectralResidual method*), 37
SpectralResidual (*class in realseries.models.sr*), 36
SR_CNN (*class in realseries.models.srcnn*), 38
standardization() (*in module realseries.utils.preprocess*), 56
STL (*class in realseries.models.stl*), 39

T

thres_search() (*in module realseries.utils.evaluation*), 54
training (*realseries.utils.utility.aleatoric_loss attribute*), 58
training (*realseries.utils.utility.mmd_loss attribute*), 59

V

VAE_AD (*class in realseries.models.vae_ad*), 40
VAE_Dense (*class in realseries.models.vae_dense*), 41

X

X_train (*realseries.models.AR.AR attribute*), 25

Y

y_hat (*realseries.models.lstm_dynamic.LSTM_dynamic attribute*), 31
y_test (*realseries.models.lstm_dynamic.LSTM_dynamic attribute*), 31
Y_train (*realseries.models.AR.AR attribute*), 25