APIs Manual

Created: 28th Jan. 2019
Modified: 28th Jan. 2019
Author: wwt
Version: 1.0

Project name

Unsupervised Anomaly Detection

Introduction

Implementation of anomaly detection algorithms/models for discovering anomalies (occurrence of potential attacks) in Multivariate Time Series of Log Event Counts

Dependencies:

- python 3 (python 2 may be compatible but no guarantee)
- numpy-1.15.4
- scipy-1.2.0
- scikit-learn-0.20.2
- matplotlib-3.0.2
- pyyaml-3.13

Code structure:

```
|Unsupervised-Anomaly-Detection
    |common
        |clean_data.py    # APIs for data preprocessing
        |common_funcs.py  # common APIs for reading/saving/extracting/formating data, etc.
        |show_decisions.py   # A simple API for showing detection results
    |detect_algos
        |EDK-means
            |data_norm.txt       # identical to that in preprocessed_data
            |data_std.txt        # identical to that in preprocessed_data
            |detect_EDK-means.py   # fitting and training APIs for K-means
            |parameters.yaml       # parameters of data input and model
            |saved_model.yaml    # persistence of the model on disk
            |train_res_EDKmeans.txt   # detections made in our training
        |isoForest
            |data_norm.txt       # identical to that in preprocessed_data
            |data_std.txt        # identical to that in preprocessed_data
            |detect_isoForest.py   # fitting and training APIs for Isolation Forest detector
            |parameters.yaml        # parameters of data input and model
            |saved_model.mdl     # persistence of the model on disk
            |train_res_isoforest.txt   # detections made in our training
        |OCSVM
            |data_norm.txt       # identical to that in preprocessed_data
            |data_std.txt        # identical to that in preprocessed_data
            |detect_isoForest.py   # fitting and training APIs for One Class SVM
            |parameters.yaml        # parameters of data input and model
            |saved_model.mdl     # persistence of the model on disk
            |train_res_OCSVM.txt   # detections made in our training
        |others
    |examples
        |example1   # a demonstration of how to detect anomalies using OCSVM detector
        |raw_log_files   # file containing raw log data
    |preprocessed_data
        |data_norm.txt    # aggregated, normalized data
        |data_std.txt     # aggregated, standardized data
    |raw_datasets
        |jiuzhouLog
            |event-xxx.txt   # raw data of log
    |training_set_decisions
        |pred_xxx.txt   # decisions made on our training process
```

Main APIs

1. preprocess_data (*raw_files_list*, *out_fname*, *headers=0*, *rescale='std'*, *begin_time=''*, *end_time=''*)

Module: common/clean_data.py

Description: The main API for data pre-processing. Read raw data files (e.g., ~/raw_datasets/raw_log_files/event-crond.txt), combine them into an aggregated, normalized/standardized matrix (with each row as a frame containing timestamp and counts of events) in memory, and finally write to a specified disk file (e.g., ~/preprocessed_data/data_norm.txt)

Parameters:

- *raw_files_list*: raw data to be pre-processed, content should be in format: event-name/yyyy-mm-dd/hh/count

Raw data (API input) format example:

```
event-xxx/2018-06-27/00h/90
  event-xxx/2018-06-27/17h/23
  event-xxx/2018-06-27/18h/175
  ...
```

- *out_fname*: aggregated data matrix in format:

  API output example (normalized):

```
# time, CROND, RSYSLOGD, SESSION, SSHD, SU (header)
  2018-06-29-00, 0.829, 0.0, 0.796, 0.155, 0.884615
  2018-06-29-01, 0.804, 0.0, 0.805, 0.154, 0.903
  ...
```

- *headers*: number of headlines at the front
- *rescale*: apply Normalization ('norm') or (Z-scale) Standardization ('std') to data, default = 'std'
- *begin_time*: starting timestamp of the data after pre-processed,
  time span of raw data may be shrunk or extended,
  default = '', which means taking starting time of shortest channel
- *end_time*: ending timestamp of the data after pre-processed, time span of raw data may be shrunk or extended; default = '', which means taking ending time of shortest channel

return: pre-processed data, identical to that stored in the file 'out_fname'

2. model. fit(*train_file='data_std.txt'*, *config='parameters.yaml'*, *model_file=''*, *slotting=True*, *plotting=False*):

Module:

- detect_algos/EDK-means/detect_EDK-means.py
- detect_algos/OCSVM/detect_OCSVM.py
- detect_algos/isoForest/detect_ isoForest.py

Description: The main API for training models (i.e., fitting given training data). Train the model(s) on the given (pre-processed) data set, within each time slot if enabled. The model will be saved to local disk as specified by the *'model_save_path*' in config file.

Parameters:

- *train_file*: contains data to fit. Default = 'data_std.txt'

  training data format example:

```
# time, CROND, RSYSLOGD, SESSION, SSHD, SU (header)
  2018-06-29-00, 0.829, 0.0, 0.796, 0.155, 0.884615
  2018-06-29-01, 0.804, 0.0, 0.805, 0.154, 0.903
  ...
```

- *config*: configuration file path. Configuration files differ for different models with the main difference in '*Model parameters*' part of the config file in YAML format. Default = 'parameters.yaml'.

  Config file example:

```
# parameters.yaml for isoForest
# input data config
  data_file: data_std.txt
  rescale: std
  num_channels: 5
  data_format_py3:
    - [timestamp, U13]
    - [CROND, f8]
    - [RSYSLOGD, f8]
    - [SESSION, f8]
    - [SSHD, f8]
    - [SU, f8]
  data_format_py2:
    - [timestamp, S13]
    - [CROND, f8]
    - [RSYSLOGD, f8]
    - [SESSION, f8]
    - [SSHD, f8]
    - [SU, f8]
  headlines: 1
  training_data_range_limit: [-1, -1]# no limit by assigning it to [-1,-1]
  test_data_range_limit: [-1, -1]  # no limit by assigning it to [-1,-1]
  # slotting config
  slot_size: 3  # disable slotting by assigning slot_size to -1
  # model parameters
  model_name: One Class SVM
  SVMKernel: rbf
  gamma: scale
  contamination: 0.1
  decision_save_path: train_result_OCSVM.txt
  model_save_path: saved_model.mdl
```

- *model_file*: specified model file to store pre-trained model(s), save to the path specified in config file if not given.
- *slotting*: if True then load slotting configs from cfg (default = parameters.yaml), no slotting if False
- *plotting*: if True then plot the decisions on training data, no plotting if False.
- ***return***: decision functions on training data.

3. model.detect(*test_file=''*, *config='parameters.yaml'*, *model_file = 'saved_model.mdl'*, *plotting=True*)

Module:

- detect_algos/EDK-means/detect_EDK-means.py
- detect_algos/OCSVM/detect_OCSVM.py

- detect_algos/isoForest/detect_ isoForest.py

Description: The main API for applying pre-trained models to detect anomalies in the specified test data set.

Parameters:

- *test_file*: data file containing test data in the format.
  test data format example:

```
# time, event-crond, event-rsyslogd, event-session, event-sshd, event-su
2018-06-29-00, 0.147, -0.223, 0.571, -0.594, 1.298
2018-06-29-01, -0.215, -0.223, 0.696, -0.597, 1.443
   ...
```

- *config*: configuration file path, identical to the one used in training (fitting) process. Default = 'parameters.yaml'
- *model_file*: model file containing pre-trained model(s).
- *plotting*: if True then plot the decisions on training data, no plotting if False.
- ***return***: decisions on the test data.

Demos

Module: examples/example1/example1.py*

*absolute path in our test = "F:/wwt/projects/codes/Unsupervised-Anomaly-Detection".

Need to re-configure the path before running this example.

Description: A simple example demonstrating the usage of our data pre-processing API and time series anomaly detection APIs (as introduced above). *One Class Support Vector Machine (OCSVM)* detector is trained and tested in this demo. Raw data is a five-channel time series of length 3457, which will be pre-processed by the *preprocess_data()* API into a standardized data set partitioned into two parts – training set and test set. Training set is then fed into the slot-enabled OCSVM models created using *model.fit()*and we use the trained model (stored on local disks as a .mdl file) to detect anomalous frames in the test set with *model.detect()*. Results returned from *model.detect()* are decisions on the test series data.

```
import sys
# module path, i.e., the absolute directory path of Unsupervised-Anomaly-
Detection/
# Needs to be re-configured before running on your machine
project_path = "F:/wwt/projects/codes/Unsupervised-Anomaly-Detection"
sys.path.append(project_path)

import common.common_funcs as cf  # common APIs for
reading/saving/extracting/formating data, etc.
```

```
import common.clean_data as clean
import detect_algos.OCSVM.detect_OCSVM as ocsvm  # our implementation of
One Class SVM anomaly detector



# execute only if run as a script
if __name__ == "__main__":
    # locate raw files and this demo’s path prefix. Need to re-configure
them on your local machine.
    demo_path = 'F:/wwt/projects/codes/Unsupervised-Anomaly-
Detection/examples/example1/'
    raw_files_list = ['F:/wwt/projects/codes/Unsupervised-Anomaly-
Detection/examples/raw_log_files/event-crond.txt',
                      'F:/wwt/projects/codes/Unsupervised-Anomaly-
Detection/examples/raw_log_files/event-rsyslogd.txt',
                      'F:/wwt/projects/codes/Unsupervised-Anomaly-
Detection/examples/raw_log_files/event-session.txt',
                      'F:/wwt/projects/codes/Unsupervised-Anomaly-
Detection/examples/raw_log_files/event-sshd.txt',
                      'F:/wwt/projects/codes/Unsupervised-Anomaly-
Detection/examples/raw_log_files/event-su.txt']
    # content in even-xxx.txt:
    #   event-xxx/2018-06-27/00h/90
    #   event-xxx/2018-06-27/17h/23
    #   event-xxx/2018-06-27/18h/175
    #   event-xxx/2018-06-27/19h/206
    #   event-xxx/2018-06-27/20h/206
    #   event-xxx/2018-06-27/21h/206
    #   event-xxx/2018-06-27/22h/206
    #   event-xxx/2018-06-27/23h/218
    #   ...

    cfg_path = demo_path + 'parameters.yaml'
    model_path = demo_path + 'saved_model.mdl'
    training_file_path = demo_path + 'data_std.txt'
    test_file_path = demo_path + 'data_std.txt'

    # pre-process data and save to '.data_std.txt'
    print(clean.preprocess_data(raw_files_list,
                                demo_path + 'data_std.txt',
                                headers=0, rescale='std', begin_time='2018-
06-29-00', end_time='2018-11-20-00'))


    # train OCSVM on the first 3000 data records, as specified in the
config file (parameters.yaml)
```

```
    # i.e., 2018-06-29 to 2018-10-31
    # after training, the model(s) will be saved to the path specified by
model_file argument

    ocsvm.fit(train_file=training_file_path, config=cfg_path,
model_file=model_path,
              slotting=True, plotting=False)

    # test OCSVM on the last 457 data records, as per the config file
(parameters.yaml)
    # i.e., 2018-11-01 to 2018-11-20
    res = ocsvm.detect(test_file=test_file_path, config=cfg_path,
model_file=model_path,
                       plotting=True)

    print(res)
```