Anomaly Detection Library

Kapil Thakkar and Reshma Kumari

27th May, 2016

Abstract

This document explains the methods and corresponding function written for Anomaly detection Library to detect anomaly in multiple time series. All corresonding files can be found in "library" folder. We have performed analysis on Onion data considering its Retail Price, Wholesale Price and Arrival data. File used to perform analysis and studying various results is "library/fullAnalysis.py". To study local and national anomalies file used is "library/fullAnalysisUpdated.py". "fullAnalysis.py" file contains required comments to understand how analysis is performed.

Contents

Chapter 1

Window Based Correlation

Refer file "library/window_correlation.py".

1.1 Introduction

This technique is basically applied on two time-series. Let's say we have two time series as series1 and series2. So, in this method, we first find correlation at various lags between these two time series. User can specify minimum and maximum lag to consider. So, for each value between minimum and maximum lag, we find correlation values.

After finding correlation values at all lags, we consider that lag at which correlation value is higher, among all previously calulated correlation values, at all lags. Let's say that lag be "x". So, depending upon that "x", we shift series1 or series2. If "x" is positive, we move series2 by "x" units and if it is negative than we shift series1 by $|x|$ units.

Now, we are ready to apply window correlation. Take window size,"w" as input. First window will be from 1st element to w'th element of both the time series after aligning by lag "x". Find correlation for this window between two time-series and save it in an array. Now, slide window by "w" elements and calculate correlation value again and so on. Now, we have correlation values at multiple windows.

Now, let's say both the series should have been positively correlated. So, what we do is, we choose threshold by MAD test if not provided to us, and find all correlation values which are below that threshold and report all those windows as anomaly. Similarly, for negative correlation values above the threshold value is reported as anomaly.

1.2 Related Functions

1.2.1 correlation(arr1, arr2, maxlag, pos, neg)

This function calculates correlation between arr1 and arr2 at all possible lags between -maxlag to +maxlag, as specified by pos and neg parameters.

- Input Parameters

1. arr1 *(list)* : Input series 1 as a list of float values
2. arr2 *(list)* : Input series 2 as a list of float values
3. maxlag *(int)* : maximum (maxlag) and minimum (-maxlag) lag to consider while calculating correlation betweem arr1 and arr2
4. pos *(int, 1 or 0)* : To consider positive lag or not, i.e. 1 to maxlag, if value is 1, than positive lag will be considered, else not.
5. neg *(int, 1 or 0)* : To consider negative lag or not, i.e. -maxlag to -1, if value is 1, than negative lag will be considered, else not.

- Output *(list)* :
  Returns list of tuples of the form

  (lag, correlation value at this lag)

1.2.2 getMaxCorr(arar1,positive_correlation)

If both the series are positively correlated than we will be interested in maximum positive correlation or if both series are negatively correlated than we will be interested in minimum negative correlation, which is specified by positive_correlation parameter.
This function takes list of tuples of the form (lag, correlation value at this lag) as input. Returns lag value at which correlation value is maximum, if positive_correlation is True, and returns lag at which correlation value is minimum if positive_correlation is False.

- Input Parameters

  1. arr1 *(list)* : list of tuples of the form
     (lag, correlation value at this lag)
     i.e. correlation values at various lags
  2. positive_correlation *(boolean, "True" or "False")* :
     - True: If value of this parameter is True than it will return lag at which correlation value if maximum (positive)
     - False: If value of this parameter is False than it will return lag at which correlation value if minimum (negative)

- Output *(Tuple)* :
  returns single tuple of the form (lag,correlation value at this lag), i.e. lag at which optimum correlation value is found along with correlation value.

1.2.3 correlationAtLag(series1, series2, lag, window_size)

This function first aligns two series by given lag. If lag is positive than it shifts start of series2 else start of series1. After aligning both the series according to lag, this function calculates correlation between both series at all windows.

window_size states size of the window. So, we will start with first window taking first "window_size" elements from each series and will calculate correlation. We will save this correlation value in list and will slide to next window. Next window will start after "window_size" elements. In such a way, we calculate, correlation at all windows and return the list of correlation values.

- Input Parameters

  1. series1 *(list)* : Input series 1 as a list of float values
  2. series2 *(list)* : Input series 2 as a list of float values
  3. lag *(int)* : lag at which series needs to be adjusted as explained above
  4. window_size *(int)* : window size to be considered

- Output *(list)* :
  Returns list of correlation values (of float type) for all windows calculated at given lag

1.2.4 WindowCorrelationWithConstantLag(arr1, arr2, window_size,maxlag, positive_correlation, pos, neg)

This is sort of driver function, which will call above 3 functions. This function will first get lag at which series needs to be adjusted. Than using this lag, it will calculate correlation values at all windows and will return it.

- Input Parameters

  1. arr1 *(list)* : Input series 1 as a list of float values
  2. arr2 *(list)* : Input series 2 as a list of float values
  3. window_size *(int)* : window size to be considered while calculating window correlation
  4. maxlag *(int)* : maximum (maxlag) and minimum (-maxlag) lag to consider while calculating correlation betweem arr1 and arr2, to align both the series
  5. positive_correlation *(boolean, "True" or "False")* :
     - True: This suggest that both the series are positively correlated
     - False: This suggest that both the series are negatively correlated
  6. pos *(int, 1 or 0)* : If value of this parameter is 1 than we will consider positive values for lag, i.e. 1 to +maxlag to align both the series initially
  7. neg *(int, 1 or 0)* : If value of this parameter is 1 than we will consider negative values for lag, i.e. -maxlag to -1 to align both the series initially

- Output *(list)* :
  Returns tuple of the form (lag,array) Where lag is lag value for which whole series is shifted and then at that lag, we have calculated correlation for all window. Correlation value for all windows is stored in array.

1.2.5 anomaliesFromWindowCorrelationWithConstantlag(arr1, arr2, window_size=15,maxlag=15, positive_correlation=True, pos=1, neg=1, default_threshold = True, threshold = 0):

This is main function of this method. This is driver of whole method. Using previously stated methods, it will first gather correlation values at different windows. Than depending upon which type of threshold is to be used, it will filter out anomalies. If default threshold is to be used, than it will be calculated using MAD test on the correlation values at each window, else threshold provided by user will be used.

Correlation values not satisfying threshold will be reported along with the date range of that window.

- Input Parameters

  1. arr1 *(list)* : Input series 1 as a list of tuples of the form (date,value)
  2. arr2 *(list)* : Input series 2 as a list of tuples of the form (date,value)
  3. window_size *(int)* : window size to be considered while calculating window correlation
  4. maxlag *(int)* : maximum (maxlag) and minimum (-maxlag) lag to consider while calculating correlation betweem arr1 and arr2, to align both the series
  5. positive_correlation *(boolean, "True" or "False")* :
     - True: This suggest that both the series are positively correlated
     - False: This suggest that both the series are negatively correlated
  6. pos *(int, 1 or 0)* : If value of this parameter is 1 than we will consider positive values for lag, i.e. 1 to +maxlag to align both the series initially
  7. neg *(int, 1 or 0)* : If value of this parameter is 1 than we will consider negative values for lag, i.e. -maxlag to -1 to align both the series initially
  8. default_threshold *(boolean, "True" or "False")* : whether to use default threshold or not. If True, default threshold will be used using MAD test on calculated correlation values for all windows.
  9. threshold *(float)* : if default_threshold is False, than this user provided threshold will be used.

- Output *(list)* :
  This function filter out anomalies and returns them. This function returns List of tuples of the form
  (start_date,end_date,correlation_value),
  where (start_date, end_date) specifies range of the window and correlation_value is value of correlation of that window

1.3 Description

Putting all together, here is the summary:

Function "WindowCorrelationWithConstantLag", first makes use of "correlation" function, to calculate correlation values at all lags to find out at which lag it needs to be align. Result of "correlation" function is passed to "getMaxCorr" function. Which will return lag at which optimum value of correlation is present. This output will be used by "correlationAtLag" function, to caluclate correlation at all windows after aligning both series by input lag. So, in this way "WindowCorrelationWithConstantLag" combines these three functions and returns correlation value at each window.

Function "anomaliesFromWindowCorrelationWithConstantlag" is the main driver. This function calls "WindowCorrelationWithConstantLag" and gets the correlation values at all windows and filters out anomalies (either using threshold calculated by MAD test or by user provided threshold) and returns them in the format of (start_date,end_date,correlation_value), where (start_date, end_date) specifies range of the window and correlation_value is value of correlation of that window.

Chapter 2

Slope Based Detection

Refer file "library/slopeBasedDetection.py".

2.1 Introduction

The method works on two time-series. It finds the ratio of steepness at two different points in the time-series. Let's say we have two time series as series1 and series2. So in this method, we first find the rate of change in the time-series values for both time-series followed by taking ratio of these rate of change. i.e suppose we have two points on time-series 1 as y_{11} and y_{12} and on time-series 2 as y_{21} and y_{22}. Rate of change between these points is calculated as following

$$S_1 = \frac{y_{12} - y_{11}}{y_{11}}$$

,

$$S_2 = \frac{y_{22} - y_{21}}{y_{21}}$$

Ratio of steepness (**rs**) is calculated as

$$rs = \frac{S_1}{S_2}$$

The **rs** is calulated between first and last point of every window of size "w" provided as input.

Now, we have rate of change in the steepness(**rs**) for every window. The outliers are detected by the threshold value provided by user or if not, then threshold is computed using MAD test on the all **rs** calculated above.

If the two time-series are expected to move in tandem, then all the points with **rs** greater than threshold are reported whereas if the two time series should not move in tandem then all the points with **rs** less than threshold are reported.

2.2 Related Functions

2.2.1 slopeBasedDetection(series1,smoothed1,series2,smoothed2, next_val_to_consider, default_threshold, threshold, what_to_consider)

This function smoothes the provided time series data using exponential moving average(if needed) and calculates **rs** for first and last point of every window of size next_val_to_consider.

- Input Parameters

  1. series1 *(list)* : Input series 1 as a list of float values
  2. smoothed1 *(boolean)* : Whether series1 is smoothed or not? If not (value of this parameter is *False*) smoothing will be done
  3. series2 *(list)* : Input series 2 as a list of float values
  4. smoothed2 *(boolean)* : Whether series2 is smoothed or not? If not (value of this parameter is *False*) smoothing will be done
  5. next_val_to_consider *(int)* : indicates the size of window or next point in time-series to calculate slope of steepness. Default is 7 days.
  6. default_threshold *(int)* : Whether to consider default threshold or not. If *True*, the threshold is calculated using MAD Test.
  7. threshold *(int)* : Threshold value to consider if default_threshold is set to *False*
  8. what_to_consider *(int)* : Can be either 1,0 or -1. If the series are supposed to move in tandem, 1 is set otherwise -1 is set. In case we don't know the correlation between two, 0 is set.

- Output *(list)* :
  Returns list of tuples of the form

  (first,second,slope_value)

  where first and second are the array index of passed series for which the **rs** is calculated.

2.2.2 anomalyDatesSlopeBaseddetetion(slopeBasedResult,any_series)

This function basically takes result of "slopeBasedDetection" as input along with any series which is list of tuples of the form (date, value), and gives date to each anomaly.

The result returned by "slopeBasedDetection" function just provides index of data point, which is reported as anomaly. But we have time series, so we need to provide date, instead of index of data point. So, this function basically, attaches each anomaly with its date and returns it.

- Input Parameters

1. slopeBasedResult *(list)* : This is list of anomalies reported by "slopeBasedDetection" function.
2. any_series *(list)* : Any list/series of tuples in the format (Date,Value), date will be used from this series to find date against each anomaly.

- Output *(list)* :
  Returns list of tuples of the following form:
  (start_date,end_date,slope_value)

2.2.3 slopeBased(series1,smoothed1,series2,smoothed2,next_val_to_consider, default_threshold, threshold, what_to_consider)

This is main function of this anomaly detection technique. This function first calls "slopeBasedDetection" function, gets list of anomalies. After that, it calls "anomalyDatesSlopeBaseddetetion" function to attach date with each anomaly and than returns result.

- Input Parameters

  1. series1 *(list)* : Input series 1 as a list of tuples of the forms (date, value)
  2. smoothed1 *(boolean)* : Whether series1 is smoothed or not? If not (value of this parameter is *False*) smoothing will be done
  3. series2 *(list)* : Input series 2 as a list of tuples of the forms (date, value)
  4. smoothed2 *(boolean)* : Whether series2 is smoothed or not? If not (value of this parameter is *False*) smoothing will be done
  5. next_val_to_consider *(int)* : indicates the size of window or next point in time-series to calculate slope of steepness. Default is 7 days.
  6. default_threshold *(int)* : Whether to consider default threshold or not. If *True*, the threshold is calculated using MAD Test.
  7. threshold *(int)* : Threshold value to consider if default_threshold is set to *False*
  8. what_to_consider *(int)* : Can be either 1,0 or -1. If the series are supposed to move in tandem, 1 is set otherwise -1 is set. In case we don't know the correlation between two, 0 is set.

- Output *(list)* :
  Returns list of tuples of the following form:
  (start_date,end_date,slope_value)
  where, start_date and end_date are points on which the steep was computed along with the slope_value which was spotted as outlier.

2.3 Description

slopeBased function is called to find anomalies based on the rate of change in value. It calls slopeBasedDetection method to compute the slopes between

points in windows. Anomaly points are reported based on the parameters provided to the function.
In order to map dates against every anomaly points instead of array index, anomalyDatesSlopeBaseddetetion is called which provides (start_date,end_date,slope_value) as final output.

Chapter 3

Linear Regression

Refer file "library/linear_regression.py".

3.1 Introduction

This technique is applied on two time series where one is independent variable and other is dependent variable. Let's say independent variable is "x" represented by series1 and "y" is dependent variable which is represented by series2, where y=f(x).

So, in this method, given values of both variables at different points, i.e. given many pairs of (x,y), which are represented here by series1 and series2, this technique tries to find relation between x and y, i.e. it tries to find best suitable function y=f(x), which can best fit given data. Note that this function can only find linear relation between two variables, i.e. it can find relation such as y = mx + c, where "m" and "c" are some variables, which are found by this method, which can best represent these two series.

After finding that function, for a given value of "x" one can predict, what should be ideal value of "y". So, this technique basically works on this principle. After finding that function, we again apply same function of the given series of "x" and try to predict corresponding series of "y" and see the relative difference between actual "y" series and predicted "y" series. If this relative difference is too high or too low or both (depending upon what user needs), we return those values as anomalies. To decide, whether value is too high or too low, we set up threshold. This threshold can be given by user or can be set automatically by using MAD test on the series generated by taking relative difference. Values beyond this threshold are reported as anomalies.

3.2 Related Functions

3.2.1 linear_regression(x_series, y_series, param = 0, default_threshold = True, threshold = 0)

This function takes two time series, x_series and y_series as input, where x_series is series corresponding to "x" variable (independent variable) and y_series is series corresponding to "y" variable (dependent variable, dependent on "x").

Given these two series, it first finds out best linear relationship between these two variables and as described in the above section, it finds relative difference between predicted and actual "y" series and the ones which are beyond threshold value are reported as anomaly.

As described above, threshold value may be calculated by MAD test on relative difference values by keeping "default_threshold" as "True", and if it is false, user will provide threshold value, by setting up "threshold" parameter above.

Note that i'th value in y_series should be corresponding to i'th value in the x_series.

- Input Parameters

  1. x_series *(list)* : List of float values representing "x" variable (independent variable)
  2. y_series *(list)* : List of float values representing "y" variable (dependent variable)
  3. param *(int, 1 or 0 or -1)* :
     Defines what to be treated as anomaly depending on its value as follows:
     0: Values going out of range, both with positive and negative error
     1: Values with positive errors
     -1: Values with negative errors
     (Here error is relative difference crossing threshold value, positive error is relative difference which is positive and crossing positive threshold value and vice-versa).
  4. default_threshold *(boolean, True or False)* : If this is set as "True", than threshold will be calculated using MAD test, if False, than user given threshold value will be used.
  5. threshold *(float)* : Here, user can provide threshold value if, default_threshold is False.

- Output *(Tuple)* :
  returns Following tuple: (result,regression_object)

  Where, "result" is list of tuples which are anomaly according to linear regression test of following format:

  (Index_of_Data_Point,x_value,y_value,predicted_y_value,
  difference_between_predicted_and_actual_y_value)

  "regression_object" is an object of linear regression test, which represents y=f(x) = mx + c, which can be used to regenerate predicted values for plotting graphs afterwards or for some other task.

  Format of using: regression_object.predict(x_value), where x_value is just one value, for which we need corresponding ideal "y" value.

3.2.2 anomalies_from_linear_regression(result_of_lr, any_series)

This function basically takes result of "linear_regression" as input along with any series which is list of tuples of the form (date, value), and gives date to each anomaly.

The result returned by "linear_regression" function just provides index of data point, which is reported as anomaly. But we have time series, so we need to provide date, instead of index of data point. So, this function basically, attaches each anomaly with its date and returns it.

- Input Parameters

  1. result_of_lr *(list)* : This is list of anomalies reported by "linear_regression" function. Note that here we are just passing list of anomalies only and not the regression object, i.e. we are passing just first element of tuple returned by "linear_regression" function.
  2. any_series *(list)* : Any list/series (x_series or y_series) of tuples in the format (Date,Value), date will be used from this series to attach each anomaly with its corresponding date.

- Output *(list)* :
  Returns list of tuples of the following form:

  (date,x_value,y_value,predicted_y_value, difference_between_predicted_and_actual_y_value)

3.2.3 linear_regressionMain(x_series, y_series, param = 0, default_threshold = True, threshold = 0)

This is main function of this anomaly detection technique. This function first calls "linear_regression" function, gets list of anomalies. After that, it calls "anomalies_from_linear_regression" function to attach date with each anomaly and than returns result.

- Input Parameters

  1. x_series *(list)* : List of tuples of the format (date,value) representing "x" variable (independent variable)
  2. y_series *(list)* : List of tuples of the format (date,value) representing "y" variable (dependent variable)
  3. param *(int, 1 or 0 or -1)* :
     Defines what to be treated as anomaly depending on its value as follows:
     0: Values going out of range, both with positive and negative error
     1: Values with positive errors
     -1: Values with negative errors
     (Here error is relative difference crossing threshold value, positive error is relative difference which is positive and crossing positive threshold value and vice-versa).

4. default_threshold *(boolean, True or False)* : If this is set as "True", than threshold will be calculated using MAD test, if False, than user given threshold value will be used.
5. threshold *(float)* : Here, user can provide threshold value if, default_threshold is False.

- Output *(list)* :
  Returns list of tuples of the form

  (start_date,end_date,difference_between_predicted_and_actual_y_value)

  Note that here, start_date is equal to end_date, as we are working day-wise in this technique, instead of any window.

3.3 Description

Putting all together, here is the summary:

"linear_regressionMain" is the main function of this technique, which calls 2 other functions and returns result. First it calls, "linear_regression" function, gets list of anomalies. After that, it calls "anomalies_from_linear_regression" function to attach date with each anomaly and than returns result.

Chapter 4

Graph Based Anomaly Detection Technique

Refer file "library/graphBasedAnomaly.R" for R script. Refer file "library/callingGraphBasedAnomaly.py" to call R script in python.

4.1 Introduction

This technique was introduced by [1]. We have used R implementation given by authors of this book [2]. So, here by using python script, we will be just calling R script with appropriate arguments and will be using result provided by that script.

Graph based anomaly detection technique considers each day as a node of a graph. Similar nodes are connected to each other by some weight. Similarity of nodes are calculated by making use of the values of that node i.e. value(s) of timeseries on that date. Based on this similarity, edge weights are also assigned. Then random walk algorithm is applied on this graph structure and connectivity value of each node is calculated. Graph nodes having the least connectivity values are reported as anomaly.

Note that previous techniques, like Window Correlation, Slope Based and Linear Regression techniques, can take only 2 time series as input. They also don't consider historical values, trend or seasonality. It just makes prediction on the given present data. Whereas, this Graph based anomaly detection technique, can take multiple time series as input and also considers trends, seasonality as well, as explained in research paper [1].

So, here, we take multiple time series as input. Out of them, one will be dependent on rest of the others. We will call R script, it will print result in one csv file. We read that CSV file and return result. Note that here we do not have threshold value. We just give number of points with the least connectivity value and function returns them. If in future, one wants to add threshold value on connectivity than function can be modified according to that as well.

4.2 Related Functions

4.2.1 graphBasedAnomalyCall(dependentVar, numberOfVals, timeSeriesFileNames)

This function calls the R Script "graphBasedAnomaly.R". This function takes multiple time series as input, which are stored in files, whose names are stored in "timeSeriesFileNames" list. This time-series files are generated by us only. Out of these time series, one will be for dependent variable and others will be corresponding to independent variable. So variable, "dependentVar" represents which time series/variable is dependent.

This function executes R script and writes output to the file named "GraphBasedAnomalyOp.csv".

- Input Parameters
  1. dependentVar *(int)* : Index of the dependent variable, where dependentVar = function of independantVars
  2. numberOfVals *(int)* : Each CSV contains how many values? That is each time series has how many values?
  3. timeSeriesFileNames *(list)* : Names of the files in which series is stored. File should contain only series values.
- Output: This function does not generate any output. R Script will write output to CSV file as stated before.

4.2.2 generateCSVsForGraphBasedAnomaly(lists, dateIndex, seriesIndex)

In python code, we have time-series as a list. This list is list of tuples, in which first value of tuple is date and than we have more than one values in the same tuple, representing different time-series. For example, if we have testcase as onion, than for one city we have 3 time series along with date, which is represented as list of tuples of the the form (date, arrival, wholesale price, retail price). But, for R script, we just need time series values. So this function will take series of time series in variable "lists", where lists[i] will represent one timeseries or multiple time series for one object (like explained previously we can have multiple time series for one city).

dateIndex will say which tuple number for the list lists[i] represents date and seriesIndex represents, if lists[i] represents multiple series than which one to take out of them. This can be explained by example as follows:

Let's say, we have lists as folows:

```
[
[(1-1-2010, x1, y1, z1), (2-1-2010, x2, y2, z2), ... ],
[(1-1-2010, x1, y1, z1), (2-1-2010, x2, y2, z2), ... ],
[...], ...
];
```

So, here we have time-series corresponding to two entities, which can be accessed via lists[0] and lists[1]. Now, lists[0] gives us 3 time-series for one entity. But let's say, here we need only one corresponding to "y" time series. So, give dateIndex as 0 here and seriesIndex as 2. So, this function will create 2 CSVs, one for each entity. Each CSV will have values [y1, y2, y3, ...]. One line will contain one value in file.

Note that it is not necessary to have multiple time-series for on entity. We can have just simple structure as follows:

```
[
[(1-1-2010, x1), (2-1-2010, x2), ... ],
[(1-1-2010, y1), (2-1-2010, y2), ... ],
[...], ...
];
```

So here, we have two time-series as x and y, and we can than give dateIndex as 0 here and seriesIndex as 1. This will create two CSVs, one for "x" and other for "y".

After creating these CSVs, this function returns names of the file created.

- Input Parameters

  1. lists *(list)* : List of time-series, where lists[i] = list of tuple of the form (date, val1 [, val2, val3, ...])
     where date is in form of string and values in square brackets are optional.
  2. dateIndex *(int)* : column number of date in list of tuple (starting with 0)
  3. seriesIndex *(int)* : column number of series in list of tuple (starting with 0)

- Output *(Tuple)*:
  returns tuple of the form, (dates,fileNames),

  Where,
  fileNames: Generated multiple CSVs, corresponding to each series for the input of R script. Returns name of these files.
  dates: Separated date from the series, so that later we can combine result of the R script (anomalies) with dates.

4.2.3 getAnomalies(dates,resultFile, numOfPtsReqd)

This function does the work of combining result of R script with the date. Result generated by R will be in some file, which is passed here as resultFile parameter. This will have indices for each day. So using this we append dates to it. So now, we have connectivity value for each date. This function sorts them according to connectivity value and returns the number of points required stated by parameter numOfPtsReqd, which has low connectivity value.

- Input Parameters

1. dates *(list)* : List of dates, returned by "getAnomalies" function.
2. resultFile *(string)* : Path of file to which output of R script is written
3. numOfPtsReqd *(int)* : Number of anomalous points required

- Output *(list)*:
  reurns list of tuples of the form:
  (start_date, end_date, connectivity_value)
  Note that, here start_date will be same as end_date, as this function returns results day-wise.

4.2.4 graphBasedAnomalyMain(lists, dependentVar, numOfPtsReqd, dateIndex=0, seriesIndex=1)

This is the main function of this method. This function makes call to other functions, uses the output of one, as a input to other, combines all functions and returns generated output.

- Input Parameters

1. lists *(list)* : List of time-series, where lists[i] = list of tuple of the form (date, val1 [, val2, val3, ...])
   where date is in form of string and values in square brackets are optional.
2. dependentVar *(int)* : Index of the dependent variable, where dependentVar = function of independantVars
3. numOfPtsReqd *(int)* : Number of anomalous points required
4. dateIndex *(int)* : column number of date in list of tuple (starting with 0)
5. seriesIndex *(int)* : column number of series in list of tuple (starting with 0)

4.3 Description

Putting all together, here is the summary:

"graphBasedAnomalyMain" is the main function. First, it calls "generateCSVsForGraphBasedAnomaly", which will generate files for each series, which will be used as input for R script. It also generates list of dates. Now, this list of file is passed to function, "graphBasedAnomalyCall", which will execute R script and will generate output in predefined file. This file name, along with dates and number of anomaly points required is passed to function "getAnomalies", which will return output in required format.

Chapter 5

Multivariate Time Series Anomaly Detection Technique

Refer file "library/MultivariateSeriesModified.R" for R script. Refer file "library/multiVariateTimeseries.py" to call R script in python.

5.1 Introduction

The method uses vector autoregressive framework for multivaraiate time-series analysis in order to forecast values. The framework treats all the variables as symmetrical and all the variables are modelled as if they influence each others equally.

VAR generates forecast values for all the variables in recursive manner. Since VAR works on only stationary series, lag needs to be found so that the series could be differenced in order to make them stationary.

The code is implemented in R and python is used to call the R script with Appropriate arguments and process the intermeddiate results generated from the script. Forecast and vars library are used in R to implement VAR model for multiple time-series.

All the interrelated time-series are passed to R script using a csv file. 60% (This can be configured as per user need) of the passed data for every time-series is consumed for modelling the time-series. Rest of the time-series data is used to find the anomalies in the system by finding predicted values and range of higher and lower predicted values.

Multiple csv files (one for each varaiable) are generated as an output to the R script call which has actual values of variables along with the predicted ,lower and higher values of prediction. All the points which does not fall in the forecasted range and the percentage differnce between the actual and forecasted value breached threshold are reported as anomalies.

Note that the threshold is computed with MAD test on the percentage of differnce between actual and forecasted value. If in future, one wants to add threshold value than function can be modified according to that as well.

Refer [3] for more detailed information.

5.2 Related Functions

5.2.1 MultivariateAnomaly(fileName,hd, paramCount,fileStart)

This function is inside R script which takes a fileName containing all the related time-series and generate output files (one file for every component) containing predicted, actual, lower and higher forecasted values.

The name of output files starts with fileStart appended with the sequence number of variable. Like if "RetailWsArrival" is passed as fileStart and there are two variables or series in input file. The names of generated file will be : RetailWsArrival1.csv and RetailWsArrival2.csv

- Input Parameters
  1. fileName *(string)* : Name of the file which contains all the interrelated time series for model
  2. hd *(boolean)* : Whether the CSV file contains header for columns or not
  3. paramCount *(int)* : Number of variables in file
  4. fileStart *(string)* : Prefix for the name of output files to be generated
- Output: This function will write output to CSV file as stated before.

5.2.2 multivaraiateAnalysis(args)

This function calls the R script through python passing args as the input to the R script.

- Input Parameters
  1. args *(list)* : list of strings which serve as input to R script
- Output: No output.

5.2.3 csvTransform(filePath,startDate)

This function segregate the anomalies from all the other points for which the forcast was generated by R script function named "MultivariateAnomaly".

- Input Parameters
  1. filePath *(string)* : Path of output file generated by R script "MultivariateAnomaly" which contains actual, forecasted, lower and higher values
  2. startDate *(string)* : The date from which the forecast was generated by R script

- Output *(list)*:
  returns list of tuples of the form:
  (start_date, end_date, percDiff)
  Where percDiff is the percentage differnce between the actual and forecasted value. Note that, here start_date will be same as end_date, as this function returns results day-wise.

5.3 Description

Putting all together, here is the summary:

"MultivariateAnomaly" is the R Script function which is called by python function "multivaraiateAnalysis". It generates forecasted values based on the model generated by 60% of the input data. Lastly, csvTransform processes the generated file in order to report anomalies which are falling outside the range of lower and higher forecast value and breach the threshold calculated using MAD test on percentage of differnce between actual amd forecasted value.

Chapter 6

Utility

Refer file "library/Utility.py".

6.1 Introduction

Here, we explain some related funtions which are used by stated anomaly detection techniques. Note that some of these functions can be used to process the results of the anomaly techniques.

6.1.1 Functions used by Anomaly detection techniques

- **convertListToFloat(li)**
  Given list of elements, this function type casts all the elements to float type.
  - Input Parameters
    1. li *(int)* : list of elements
  - Output *(list)*:
    Returns list of elements after converting each element into float

- **getColumnFromListOfTuples(lstTuples,i)**

  This function returns i'th element of all tuples as a list.
  - Input Parameters
    1. lstTuples *(int)* : list of tuples. Tuples has no fixed format
    2. i *(int)* : index of which tuple element to return, index starting from zero
  - Output *(list)*:
    Returns list of elements after fetching i'th element from each tuple.

- **findAverageTimeSeries(timeSeriesCollection)**

It takes 2D list of element, where each element of timeSeriesCollection is one time series. It returns average of all time series. (first element of resltant time series will be average of first element of all time series)

For example let,
timeSeriesCollection: [
[1,2,3], # Timeseries 1
[4,5,6], # Timeseries 2
[7,8,9] # Timeseries 3
]

This function will return,
[4,5,6]

- Input Parameters
  1. timeSeriesCollection *(list)* : 2D array of float elements.
- Output *(list)*:
  Returns list after taking average of all time series.

- **writeToCSV(lstData,fileName)**

  This writes the list of tuples into the file provided as input.

  - Input Parameters
    1. lstData *(list)* : list of tuples that needs to be written in csv file.
    2. fileName *(string)* : Name of file in which the list of tuples needs to be written.
  - Output *(file)*:
    Generates a csv file with data written in that.

- **concateLists(lstData)**

  This function converts the list of lists into a single list of tuples.

  For example let,
  timeSeriesCollection: [
  [1,2,3], # Timeseries 1
  [4,5,6], # Timeseries 2
  [7,8,9] # Timeseries 3
  ]

  This function will return,
  [
  (1,4,7), # Timeseries 1
  (2,5,8), # Timeseries 2
  (3,6,9) # Timeseries 3
  ]

- Input Parameters
  1. lstData *(list)* : list of different lists
- Output *(file)*:
  Return a single list of tuples.

- **cleanArray(array)**

  This function removes "nan" (Not a number values) from the list.

  - Input Parameters
    1. array *(list)* : list of float type elements elements.
  - Output *(list)*:
    Returns list of float elements after removing "nan" elements.

- **MADThreshold(array)**

  This function is used to calculate threshold value using Median Absolute Deviation outlier detection method.

  - Input Parameters
    1. array *(list)* : Array of integers, real numbers, etc
  - Output *(float)*:
    threshold value computed using MAD Test.

- **smoothArray(array, alpha = 2.0/15.0)**
  This function smooths input array by exponential moving average technique.

  - Input Parameters
    1. array *(list)* : Array of integers, real numbers, etc
    2. alpha *(float)* : smoothening factor of exponential average smoothing
  - Output *(list)*:
    Array which is exponentially smoothed

- **csv2array(filePath)**

  This function reads csv file into list

  - Input Parameters
    1. filePath *(string)* : Path of file to be read
  - Output *(list)*:
    list of rows in the csv file

- **getColumn(array, column_number)**

  This function fetches a particular column from a list of tuples.

- Input Parameters
  1. array *(list)* : List of tuples
  2. column_number *(int)* : The index of the column number to be fetched from list of tuples
- Output *(list)*:
  list of elements corresponding to the column

- **formatCSV2Array(z)**

  This function returns the same list with changed datatypes of elements within it.

  - Input Parameters
    1. z *(list)* : List of tuples
  - Output *(list)*:
    return same array changing data type of second column to float

6.1.2 Functions used to process results

- **intersection(numOfResults, list1, resultOf1, list2, resultOf2, list3 = [], resultOf3="linear_regression", list4=[], resultOf4="graph_based", list5=[], resultOf5="spike_detection" , list6=[], resultOf6="multiple_arima")**

  This function is used to take intersection of results of multiple methods (2 or more), which are passed as list here. This function requires minimum of 5 arguments.

  - Input Parameters
    1. numOfResults *(int)* : number of lists you are passing, minimum 2
    2. $list_i$ *(list)* : list represnting result of the i'th algorithm. Note that this is list of tuples of the format,
       (startDate, endDate, value)
    3. $resultOf_i$ *(string)*: this variable states, $list_i$ is of which algorithm, it can be from following:
       (slope_based, linear_regression, graph_based, spike_detection, multiple_arima)
  - Output *(list)*:
    This function returns intersection of all lists. Returned value is list of tuples of the form:
    (date, correlation value, slope_based value, linear_regression value, graph_based value, spike_detection value, multiple_arima value)

- **intersectionOfFinalResults(list1, list2)**

  This function takes intersection of 2 lists (where each list is list of tuple) and returns result. Note that these input lists are generated as a output of "intersection" method.

- Input Parameters
  1. list1 *(list)* : List of tuples of the format:
     (date, correlation value, slope_based value, linear_regression value, graph_based value, spike_detection value, multiple_arima value)
  2. list2 *(list)* : List of tuples of the format:
     (date, correlation value, slope_based value, linear_regression value, graph_based value, spike_detection value, multiple_arima value)
- Output *(list)*:
  This function returns intersection of list1 and list2. Returned value is list of tuples of the form:
  (date, correlation value, slope_based value, linear_regression value, graph_based value, spike_detection value, multiple_arima value)

- **unionOfFinalResults(list1, list2)**
  This function takes union of 2 lists (where each list is list of tuple) and returns result. Note that these input lists are generated as a output of "intersection" method.
  - Input Parameters
    1. list1 *(list)* : List of tuples of the format:
       (date, correlation value, slope_based value, linear_regression value, graph_based value, spike_detection value, multiple_arima value)
    2. list2 *(list)* : List of tuples of the format:
       (date, correlation value, slope_based value, linear_regression value, graph_based value, spike_detection value, multiple_arima value)
  - Output *(list)*:
    This function returns union of list1 and list2. Returned value is list of tuples of the form:
    (date, correlation value, slope_based value, linear_regression value, graph_based value, spike_detection value, multiple_arima value)

- **mergeDates(li)**

  This function merges overlapping time period. The list, which it takes as input, "li", is list of tuples, of the format,
  (startDate, endDate, Value).

  So if, we have overlapping period or two time periods are adjacent, for example if one tuple is (1-1-2015, 15-1-2015, 5) and other tuple is (15-1-2015, 30-1-2015, 6), than this function will produce output as, (1-1-2015, 30-1-2015, 5).

  - Input Parameters
    1. li *(list)* : List of tuples of the format:
       (startDate, endDate, Value)
       Note that here startDate and endDate, are of type *datetime* and Value is of type *float*.

- Output *(list)*:
  This function returns list of tuples of the form:
  (startDate, endDate, Value)

- **resultOfOneMethod(array)**

  This function just converts format of the result list. Usually, anomaly detection methods returns list of tuples of the format,
  (startDate, endDate, Value)
  this function will convert it to list of tuples of the format,
  (date, value)

  Basically, all the dates between startDate and endDate will be added to the result list.

  - Input Parameters

    1. array *(list)* : List of tuples of the format:
       (startDate, endDate, Value)
       Note that here startDate and endDate, are of type *datetime* and Value is of type *float*.

  - Output *(list)*:
    This function returns list of tuples of the form:
    (date, Value) Note that here date is of type *datetime* and Value is of type *float*.

Bibliography

[1] Haibin Cheng, Pang-Ning Tan, Christopher Potter, and Steven A Klooster. Detection and characterization of anomalies in multivariate time series. In *SDM*, pages 413–424. SIAM, 2009.

[2] Nagiza F Samatova, William Hendrix, John Jenkins, Kanchana Padmanabhan, and Arpan Chakraborty. *Practical Graph Mining with R*. CRC Press, 2013.

[3] 9.2 vector autoregressions — otexts. `https://www.otexts.org/fpp/9/2`.