AnomalyDetection_7_MovingAverage_Refinement_2

September 29, 2020

1 AnomalyDetection_7_MovingAverage_Refinement_2

Updates from previous notebook: - this notebook will refine the moving_avg() class

1.1 Libraries and Configuration

[1]:
```
""" Libraries """

#file / system libraries
import os
import datetime as dt

# mathematical

from numpy.fft import ifft
from numpy.fft import fft
import numpy as np

# data exploration

import pandas as pd

# data visualization

import matplotlib.pyplot as plt

""" Configuration """

# pandas

pd.set_option('display.max_columns', None)
```

1.2 Data

[2]:
```
base = '/Users/yousefnami/KinKeepers/ProjectAI/Kin-Keepers/Data/{}'
names = ['rohan','ignacio']
end_labels = ['_filtered.csv']
dfs = []

for index,name in enumerate(names):
    dfs.append(pd.read_csv(base.format(names[index]+end_labels[0]),index_col =
 ↪0))
```

[3]:
```
dfs[0] = dfs[0].sort_values(by="date")
print(dfs[0].head())
dfs[0].tail()
```

```
                    date  accX  accY  accZ   gyrX   gyrY   gyrZ  files  \
24048  2020-09-09 16:27:07  0.08  0.01  0.11 -13.49   5.99 -21.58      4
24049  2020-09-09 16:27:08  0.05 -0.03 -0.12  22.86  23.19  -1.88      4
24050  2020-09-09 16:27:09  0.00 -0.01  0.00   4.29  -5.49   7.91      4
24051  2020-09-09 16:27:09 -0.03  0.06 -0.14 -16.51  -0.30  -1.01      4
24052  2020-09-09 16:27:10 -0.01  0.00  0.01   2.25 -21.86  11.49      4

       accTotal   gyrTotal
24048  0.136382  26.144915
24049  0.133417  32.617328
24050  0.010000  10.540982
24051  0.155242  16.543585
24052  0.014142  24.798028
```

[3]:
```
                    date  accX  accY  accZ    gyrX   gyrY   gyrZ  files  \
7958  2020-09-15 03:05:24  0.03  0.12  0.04   67.21  16.81  35.61      1
7959  2020-09-15 03:05:25  0.07  0.21  0.14  105.74  43.41  46.95      1
7960  2020-09-15 03:05:26  0.07  0.21  0.14   27.43  26.10  14.13      1
7962  2020-09-15 03:05:27  0.01  0.02  0.00    3.29   1.90   1.36      1
7961  2020-09-15 03:05:27  0.07  0.09  0.05   27.43  26.10  14.13      1

      accTotal    gyrTotal
7958  0.130000   77.896292
7959  0.261916  123.570539
7960  0.261916   40.413758
7962  0.022361    4.035307
7961  0.124499   40.413758
```

[4]:
```
dfs[1].head()
```

[4]:
```
                   date  accX  accY  accZ  gyrX  gyrY  gyrZ  files  accTotal  \
0  2020-09-13 17:09:25  0.02  0.12  0.03  1.47  3.32  2.22      1  0.125300
```

```
1   2020-09-13 17:09:26  0.02  0.12  0.03  1.47  3.32   2.22      1  0.125300
2   2020-09-13 17:09:27  0.01  0.01  0.00  7.43  6.82  10.10      1  0.014142
12  2020-09-13 17:09:34  0.01  0.01  0.00  6.64  7.07  12.45      1  0.014142
13  2020-09-13 17:09:34  0.01  0.01  0.00  4.12  3.61   5.81      1  0.014142

     gyrTotal
0    4.255784
1    4.255784
2   14.273307
12  15.782173
13   7.985149
```

[304]:
```python
import datetime as dt
import numpy as np
import matplotlib.pyplot as plt

class moving_avg:

    """
    a class used to store a moving average values, parameters and methods

    Dependencies:
    -------------
    import datetime as dt
    import numpy as np
    import matplotlib.pyplot as plt


    Attributes:
    -----------

    data ( class var ): [*[*float]]
        stores all the datapoints for each window

    time_frame_start ( class var ): [datetime]
        the start of the moving average window

    time_stamps ( class var ): [*[*datetime]]
        stores the timestamps for each data point within it's window

    all_data ( class var ): [*[*float]]
        stores all the datapoints, without any partitioning for the windows␣
↪they are in

    averages ( class var ): [*float]
        stores the values of the moving average for each window
```

```
    time_frame ( optional - 5 ): int
        the length of the moving window in units of hours

    weight ( optional - (0.0, 0.75) ): (float, float)
        weight to apply to numbers greater than the specified quartile

    Methods:
    --------

    __init__( self, time_frame = 5, weight = (1.0, 0.75)):
    initialises class based on inputs; converts 'time_frame' to seconds

    plot( self, figsize = (16,8), labels = ('gyrTotal', ' accTotal'),␣
↪plot_original = False, offset = False):
        plots an n by m plots (n = 2, m = DoF) of the moving average. First row␣
↪plots the moving averages against
        discontinuous time, where the x_axis represents the index of the points␣
↪given as input, NOT the actual time
        they represent. The second row plots the moving averages against␣
↪continuous time, where the x_axis represents
        the time difference from the first datapoint.

        Components:
        -----------
        figsize ( optional - (16, 8) ): (int, int):
            sets figure size

        labels ( optional - ('gyrTotal', 'accTotal') ): (str, str)
            labels for the y_value of the time-series plots

        plot_original ( optional - False): bool
            setting to true will plot the datapoints in the background
            of the moving average (without any WEIGHTING applied).

        offset ( optional - False): bool
            will plot, on the first row, an offset version of the moving␣
↪average where it begins at the end of the
            first time window, as opposed to te beginning


    create_times ( self ):
        creates a list of the correct plotting index (i.e. real continuous␣
↪time) by calculating the difference in
        seconds from the first datapoint

    create_moving_data ( self ):
```

```
        updates the datapoints for each window based, saving also the␣
↪time_stamps. Performs the function:
        recent_average( self )


    recent_average( self ):
        calculates the average in each time frame, accounting for the weightage␣
↪specified by the user

    """
    data = [[]]
    time_frame_start = []
    time_stamps = [[]]
    all_data = [[]]
    averages = []
    time_frame = [0]
    weight = [0]

    def __init__( self, time_frame = 5, weight = (1.0, 0.75)):
        if self.time_frame[0] == 0:
            self.time_frame[0] = time_frame*3600
            self.weight[0] = weight
            print('done')

    def plot( self, figsize = (16,8), labels = ('gyrTotal', ' accTotal'),␣
↪plot_original = False, offset = False):
        dofs = len(self.data[0][0])
        self.create_moving_data()

        real_time = self.create_times(self.time_frame_start)
        averages = np.asarray(self.averages).reshape((-dofs,dofs))
        all_data = np.asarray(self.all_data).reshape((-dofs,dofs))
        fig = plt.figure(figsize = figsize)
        plt.suptitle(('Time series data for {0}.'
                      'The moving average line has a multiplicative factor of␣
↪{1} applied to values '
                      'greater than the {2} quantile in the given time window').
↪format(labels,*self.weight[0]))

        for i in range(averages.shape[1]):
            for row in range(2):
                fig.add_subplot(2,averages.shape[1],averages.shape[1]*row+i+1)
                if row%2 == 0:
                    if plot_original == True:
                        plt.plot([j for j in range(len(self.
↪time_stamps[-1]))],all_data[:,i],'.')
```

```
                    x_index = [j for j in range(averages[:,i].shape[0])]

                    if offset == True:
                        x_index = [item + len(self.data[0]) for item in x_index]

                    plt.plot(x_index,averages[:,i],'.',label = 'offset = {}'.
↪format(offset))
                    plt.xlabel('discontinuous time, $\hat{t}$, in seconds')
                    plt.legend()

                else:
                    if plot_original == True:
                        real_time_all = self.create_times(self.time_stamps[-1])
                        plt.plot(real_time_all,all_data[:,i],'.')

                    plt.plot(real_time,averages[:,i],'.', markersize = 2)
                    plt.xlabel('continuous time, ${t}$, in seconds')


                plt.ylabel('average {} in a {} hour window'.
↪format(labels[i],int(self.time_frame[0]/3600)))

        plt.show()

    def create_times ( self, time_array ):
        real_time = [0]
        for time in time_array[1:]:
            real_time.append((time - time_array[0]).total_seconds())
        return real_time


    def create_moving_data ( self ):

        if len(self.time_frame_start) == 1:
            self.recent_average()
        counter = 0
        for i,time in enumerate(self.time_stamps[-1]):
            if time not in self.time_frame_start:
                counter += 1
                self.data.append(
                    [
                        [0 for k in range(len(self.data[0][0]))] for j in self.
↪data[-1][:counter]
                    ] + self.data[-1][counter:]
                )
                self.time_frame_start.append(time)
```

```
                self.recent_average()

    def recent_average( self ):
        dofs = len(self.data[0][0])
        multiplier = self.weight[0][0]
        window = np.asarray(self.data[-1]).reshape(-dofs,dofs)
        for axis in range(window.shape[1]):
            quantile = np.quantile(window[:,axis],self.weight[0][1])
            window[:,axis] = np.where(window[:,axis] > quantile, window[:
↪,axis]*multiplier, window[:,axis])


        self.averages.append([
            window[:,index].mean() for index in range(window.shape[1])
        ])

class average(moving_avg):
    """

    Dependencies:
    -------------
    moving_avg (class)

    Attributes:
    -----------

    datapoint: [*float]
        datapoint to be considered for averaging, length --> degrees of freedom.
↪ Note that the points much be fed
        into the class in chronological order.

    time: str
        time data point is recorded in the format 'YYYY-mm-dd HH:MM:SS'

    Methods:
    --------

    __init__(self, datapoint, time):
        initilises class; converts time to datetime; stores new datapoint and␣
↪time;
        if new time exceeds average window, creates new storage location

    store_new_datapoint ( self ):
        stores the new datapoint

    """
```

```
    def __init__(self,datapoint,time):

        super().__init__()

        self.datapoint = datapoint
        self.time_stamps[-1].append(dt.datetime.strptime(time,'%Y-%m-%d %H:%M:
↪%S'))
        self.all_data[-1].append(datapoint)


        if not self.time_frame_start:
            self.time_frame_start.append(self.time_stamps[-1][-1])

        if (self.time_stamps[-1][-1] - self.time_frame_start[-1]).
↪total_seconds() < self.time_frame[0]:
            pass
        else:
            self.create_moving_data()
            self.data.append([])

        self.store_new_datapoint()

    def store_new_datapoint( self ):
        self.data[-1].append([
            point for point in self.datapoint
        ])
```

1.3 On read data

[303]:
```
import random

m_avg_instance = moving_avg()
for item in dfs[1][['gyrTotal','accTotal','date']].values.tolist():
    avg_instance = average(item[0:2],item[2])


#m_avg_instance.average()
m_avg_instance.plot(plot_original=True)
```

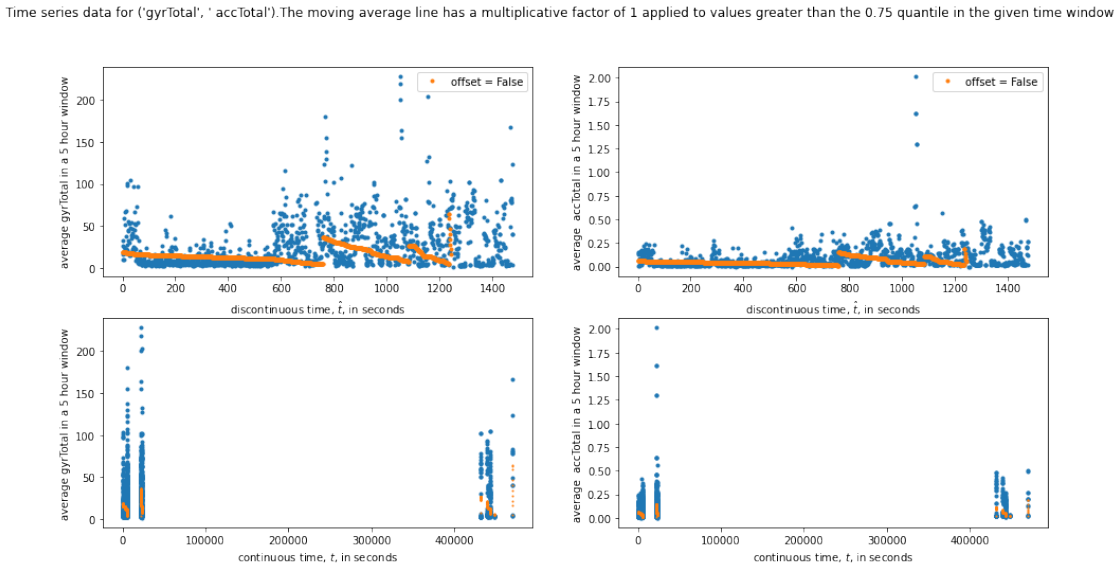
```
      ␣
↪---------------------------------------------------------------------------

       KeyboardInterrupt                         Traceback (most recent call␣
↪last)
```

```
    <ipython-input-303-6f6d0cf8fc5d> in <module>
      3 m_avg_instance = moving_avg()
      4 for item in dfs[1][['gyrTotal','accTotal','date']].values.tolist():
----> 5     avg_instance = average(item[0:2],item[2])
      6
      7


    <ipython-input-301-4a85d3814e03> in __init__(self, datapoint, time)
    216             pass
    217         else:
--> 218             self.create_moving_data()
    219             self.data.append([])
    220


    <ipython-input-301-4a85d3814e03> in create_moving_data(self)
    157                 )
    158                 self.time_frame_start.append(time)
--> 159                 self.recent_average()
    160
    161     def recent_average( self ):


    <ipython-input-301-4a85d3814e03> in recent_average(self)
    162         dofs = len(self.data[0][0])
    163         multiplier = self.weight[0][0]
--> 164         window = np.asarray(self.data[-1]).reshape(-dofs,dofs)
    165         for axis in range(window.shape[1]):
    166             quantile = np.quantile(window[:,axis],self.weight[0][1])


    ~/python_environments/KinKeepers_AI/lib/python3.7/site-packages/numpy/
↪core/_asarray.py in asarray(a, dtype, order)
     81
     82     """
---> 83     return array(a, dtype, copy=False, order=order)
     84
     85


    KeyboardInterrupt:
```

[305]:
```
m_avg_instance = moving_avg(weight = (1,0.75))
for item in dfs[0][['gyrTotal','accTotal','date']].values.tolist():
    avg_instance = average(item[0:2],item[2])
```

```
#m_avg_instance.average()
m_avg_instance.plot(plot_original=True)
```

```
done
```



2 Conclusion

The average class works, at least in determining the correct average.

There are some changes you need to make in terms of the actual class though, these are summarised below: 1. Currently, you cannot choose to plot the average, with the data points, or average on it's own - UPDATE (25.09.2020): there is now an option allowing you to do so 2. Currently, the average is calculated at the end, as opposed to at every stage (this was done to save memory, but when the model is deployed, you will need to calculate it every time) - UPDATE (28.09.2020): this feature has been added, but the feature of taking an average at the end is currenlty not working 3. You need to think about where everything will be stored, and how this will work in conjunction with Rohan's API (best wait for him to come back from holiday before starting this) - UPDATE (25.09.2020): As per Mr Jimenez, this not required right now 4. You need to account for the weightage when calculating the averages - UPDATE (28.09.2020): Weightage functionality added 5. You need to add meaningful xticks, in terms of date and time 6. You need to add functionality to be able to determine when there is a 'break' in the sequence (i.e. value decreases by a a lot) 7. You need to fix the way your class handles stuff in memory: it currently saves the values from previous instantiations as well 8. Need to clean up the function, I get the impression that there is, a LOT of repetition! 9. change the inner code so that DoF = len(labels)

```
[ ]:
```