QBUS6840: Tutorial 6 – Exponentially Weighted Smoothing

Objectives

- Implement EWM algorithm manually
- Learn how to use pandas EWM methods
- Analyze and evaluate EWM

1. Exponential weighted smoothing

Let's start with the exponential weighted smoothing.
The component from of EWM is defined as:

$$\widehat{y_{t+1|1:t}} = l_t = \alpha y_t + (1-\alpha) l_{t-1}$$

where $0 \leq \alpha \leq 1$.

Here, we first calculate the level l_t and then use it as the forecast $\widehat{y_{t+1|1:t}}$

Continue the calculation steps:

$$\begin{aligned} l_{t-1} &= \alpha y_{t-1} + (1-\alpha) l_{t-2} \\ l_{t-2} &= \alpha y_{t-2} + (1-\alpha) l_{t-3} \\ l_{t-3} &= \alpha y_{t-3} + (1-\alpha) l_{t-4} \end{aligned}$$

$$\cdots$$

$$\begin{aligned} l_2 &= \alpha y_2 + (1-\alpha) l_1 \\ l_1 &= \alpha y_1 + (1-\alpha) l_0 \end{aligned}$$

Finally, we combine the above equations together, we will finally have:

$$\widehat{y_{t+1|1:t}} = l_t = \alpha y_t + \alpha(1-\alpha) y_{t-1} + \alpha(1-\alpha)^2 y_{t-2} + \cdots + \alpha(1-\alpha)^{t-1} y_1 + (1-\alpha)^t l_0$$

You may notice in order to get the l_t which is the forecast $\widehat{y_{t+1|1:t}}$, we need to define the value of α and l_0.

However, the value of α and l_0.are sensitive to your time-series data. Inappropriate value selection will result in inaccurate forecasting results.

In this task, we will calculate the exponentially weighted smoothing results of "`AustralianVisitors.csv`" dataset. Firstly, we will manually smooth the data by using a self-defined function, and then call `ewm()` function in `pandas`. Finally, we will discuss how to select the best fitting α. Note that selecting the value of l_0.is also important. However, in this tutorial, we only choose $l_0 = y_0$ for simplicity. You can also refer to the steps of selecting α to select l_0.

Step 1: Load the Data

Download the "`AustralianVisitors.csv`" file from the QBUS6840 Canvas site.

Start by importing `pandas`, `matplotlib` and `numpy` and import the dataset:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

visitors = pd.read_csv('AustralianVisitors.csv')
y = visitors['No of Visitors']
```

Step 2: Manually smooth the data

First define the smoothing weight and create a list to store our smoothed values and fill with an initial starting point

```
alpha = 0.1
smoothed_manual = [y[0]]  # this is l0 in the method
```

Updates[0]:

Then iterate over the data and smooth it

```
for i in range(len(visitors)-1):
    smoothed_manual.append( alpha * y[i] + (1-
alpha)*smoothed_manual[i] )
```

Why we have `len(visitors)-1` in here?

Plot your manual smoothed curve by using plt.plot() funciton

```
fig = plt.figure()
plt.plot(smoothed_manual, label = "manual smoothed curve,
alpha = 0.1")
```

Can you set $l_0 = \frac{(y_1+y_2+y_3)}{3}$ and redo the smoothing?
Could you find any difference for difference choice of l_0 values?

Step 3: Smooth the data using Pandas

We can use the Pandas EWM function to smooth the data automatically: http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.ewm.html

```
smoothed1 = y.ewm(alpha=0.05, adjust=False).mean()
smoothed2 = y.ewm(alpha=0.1, adjust=False).mean()
smoothed3 = y.ewm(alpha=0.3, adjust=False).mean()
```

```
smoothed4 = y.ewm(alpha=0.7, adjust=False).mean()
```

Plot and compare the different values of alpha. Start by creating a new figure and saving a reference to it. We will use the figure reference to save the figure to an image file later.

```
plt.plot(smoothed1, label = "Alpha = 0.05")
plt.plot(smoothed2, label = "Alpha = 0.1")
plt.plot(smoothed3, label = "Alpha = 0.3")
plt.plot(smoothed4, label = "Alpha = 0.7")
plt.title("Various values of Alpha")
```

Enable the legend and move it to the top left corner

```
plt.legend(loc="upper left")
```

Then finish the plot

```
plt.show(block=False)
```

Save the plot to an image file

```
fig.savefig("Alpha.pdf")
```

Compare your manually smoothed result with ewm function result (with alpha = 0.1), what differences could you observe?

In this example, we have used different α values, say 0.05, 0.1, 0.3, and 0.7. Can you also call `ewm()` function with α equals to 0 and 1 respectively? What is the curve looks like?

Generally, for these 2 extreme cases:

- $\alpha = 0$: the forecasts of all future values are equal to the average (or "mean") of the historical data, which is called average method.
- $\alpha = 1$: simply set all forecasts to be the value of the last observation, which is called Naive method in statistics.

If you can't understand this, please go back and review the equations in the beginning.

Step 4: Find the optimal value of alpha for 1-step smoothing

We can do a grid search over a range of possible alpha values to select the alpha that produces the least error.

First define a function to calculate the SSE to measure the error

```
def sse(x, y):
    return np.sum(np.power(x - y,2))
```

Then create an empty list to store SSE scores and a range of alpha values to iterate over

```
sse_one = []
alphas = np.arange(0.01,1,0.01)
```

Use your variable explorer to check variable `alphas`. Then test each alpha value in turn.

```
for i in alphas:
    smoothed = y.ewm(alpha = i, adjust=False).mean()
    sse_one.append( sse(smoothed[:-1], y.values[1:]) )
```

Let's plot the SSE vs Alpha

```
plt.figure()
plt.plot(sse_one)
plt.title("SSE for one step smoothing")
plt.ylabel("SSE")
plt.xlabel("Alpha")
plt.xticks(np.linspace(0, 100, 10), ["{0:1.1f}".format(x)
for x in np.linspace(0,1,10) ])
plt.show(block=False)
```

What is the size of `smoothed` and `sse_one`?

Find the best fitting alpha based on the minimum SSE. Here you can use `np.argmin()` function

```
optimal_alpha_one = alphas[ np.argmin(sse_one) ]
print("Optimal Alpha for 1-step forecast
{0}".format(optimal_alpha_one))
```

Step 5: Find the optimal value of alpha for 2-step smoothing

To calculate the optimal alpha for a 2-step ahead forecasting, we can shift smoothed series by two time units. This can be achieved by making error between the third value in the original and the first in the smoothed, the 4th in original and the second smoothed value and so on.

```
sse_two = []
alphas = np.arange(0.01,1,0.01)
for i in alphas:
    smoothed = y.ewm(alpha = i, adjust=False).mean()
    sse_two.append( sse(smoothed[:-2], y.values[2:]) )
```

What is the meaning of `smoothed[:-2]`?
What is the size of `smoothed` and `sse_two`?

Then plot the SSE vs Alpha and find the best fitting alpha.

```
plt.figure()
plt.plot(sse_two)
plt.title("SSE for two step smoothing")
plt.ylabel("SSE")
plt.xlabel("Alpha")
plt.xticks(np.linspace(0, 100, 10), ["{0:1.1f}".format(x)
for x in np.linspace(0,1,10) ])
plt.show(block=False)

optimal_alpha_two = alphas[np.argmin(sse_two)]

print("Optimal Alpha for 2-step forecast
{0}".format(optimal_alpha_two))
```

2. Trend method

In this task, we will implement the trend method for smoothing, more specifically, Holt's linear method.

In the simple exponential smoothing task, we set level l_t as the forecast result, which is:

$$\widehat{y_{t+h|1:t}} = l_t$$

Compared with the previous method, Holt's linear method forecast the results by combining the level and trend.

In general, the Forecast equation in Holt's linear method is defined:

$$\widehat{y_{t+h|1:t}} = l_t + hb_t$$

where the level is:

$$l_t = \alpha y_t + (1-\alpha)(l_{t-1} + b_{t-1})$$

and the trend is:

$$b_t = \beta(l_t - l_{t-1}) + (1-\beta)b_{t-1}$$

For one-step forecast, we have $h = 1$, which is:

$$\widehat{y_{t+1|1:t}} = l_t + b_t$$

$$\widehat{y_{t+1|1:t}} = \alpha y_t + (1-\alpha)(l_{t-1} + b_{t-1}) + \beta(l_t - l_{t-1}) + (1-\beta)b_{t-1}$$

where $0 \leq \alpha \leq 1$ and $0 \leq \beta \leq 1$.

Note that similar with simple exponential smoothing, you still need to define the value of hyper parameters, i.e. α and β.

For this task, we will still use the "`AustralianVisitors.csv`" dataset. Firstly, define the smoothing weight α and β. Then we initialize the $l_0 = y_1$ and $b_0 = y_2 - y_1$ and create a list `holtsmoothed_manual` to store our smoothed values

```
alpha = 0.1
```

```
beta = 0.1

l = [y[0]]
b = [y[1] - y[0]]
```

Create a new variable `Y` to store the intermediate information:

```
Y = y.tolist()
```

Updates[1]:

For 1-step forecasting, the outcome is defined as:

$$\widehat{y_{t+1|1:t}} = l_t + b_t$$

Let's iterate the data and smooth/forecast the data. Here, we are going to forecast 12 more months after the last observation (t = 312).

```
# Updated by Boyan at 10/April
holtsforecast_manual = []

# Here, we are going to forecast 12 more months after the
last observation (t = 312).
for i in range(len(y)+12):
    # when we reach the end of the original data
    # then we need to forecast the t = T:T+12
    if i == len(Y):
        Y.append(l[-1] + b[-1])
    print(i)
    l.append(alpha * Y[i] + (1 - alpha) * (l[i] + b[i]))
    b.append(beta * (l[i+1] - l[i]) + (1 - beta) * b[i])
    # for forecasting,  Y^2 = l1 + b1
    #                   Y^3 = l2 + b2
    #                   Y^4 = l3 + b3
    #                       ...
    #                   Y^t+1 = lt+bt
    holtsforecast_manual.append(l[i] + b[i])
```

Read the above code carefully, and answer the below question:

(1) What is the size of `Y` and `holtsforecast_manual` **before** you run the code?

(2) What is the size of `Y` and `holtsforecast_manual` **after** you run the code?

(3) In what cases the statements in the if condition will be executed?

(4) Report the value of the last 12 elements in the list `l` and `b`, what you could observe?

Finally, plot your smoothed result:

```
plt.figure()
plt.plot(holtsforecast_manual[:], label = "holt forecast
curve, alpha,beta = 0.1")
```

```
plt.plot(y, label="original data")
plt.title("Holt's linear forecasting")
```

Updates[2]:

For smoothing case, we will set smoothing result with the current level, that is:

$$\widehat{Y_T} = l_t$$

the corresponding code is:

```
#%% if you are focusing on smoothing, then use the below
code:
# Updated by Boyan at 10/April
holtsmoothed_manual = []
Y = y.tolist()

# Updated 12/April
# forecast the data
for i in range(len(y)):
     l.append(alpha * Y[i] + (1 - alpha) * (l[i] + b[i]))
# Calculating l[1], l[2], etc.
     b.append(beta * (l[i+1] - l[i]) + (1 - beta) * b[i])
# Calculating b[1], b[2], etc.
    # for smoothing,    Y^1 = l1
    #                   Y^2 = l2
    #                       ...
    #                   Y^t = lt
     holtsmoothed_manual.append(l[i+1])
```

Ploting the smoothing results:

```
plt.figure()
plt.plot(holtsmoothed_manual[:], label = "holt smoothed
curve, alpha,beta = 0.1")
plt.plot(y, label="original data")
plt.title("Holt's linear smoothing")
```

You could change another value of alpha and beta (i.e. 0.9), and then compare your new results with this one.

For the indexing declaration for both forecasting and smoothing, please refer to the Section 5.5 of the Study Guide.

Note:

1. Pandas does not have a functionality of Holt's linear trend exponential smoothing
2. There is a full set of functionalities of all the Holtwinters smoothing in `statsmodels` package, see https://www.statsmodels.org/dev/generated/statsmodels.tsa.holtwinters.ExponentialSmoothing.html

3. A good online tutorial is https://machinelearningmastery.com/exponential-smoothing-for-time-series-forecasting-in-python/