QBUS6840: Tutorial 6 exponential smoothing(trend)

- The ideal scenario

trend

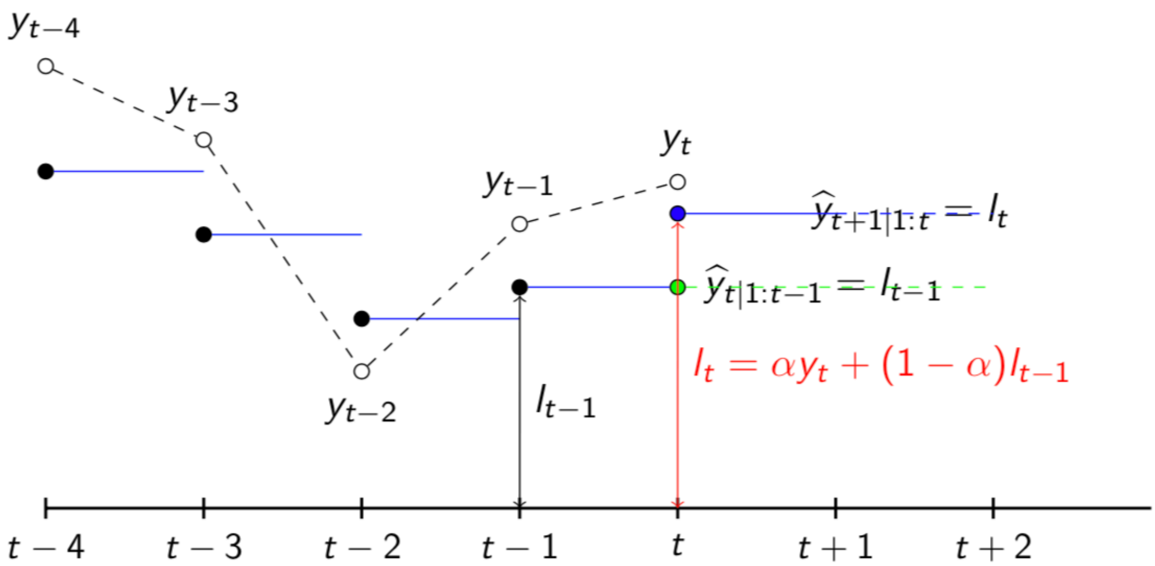
$$y_t = \omega_0 + \omega_1 t + S_t + \varepsilon_t$$

- Additive decomposition model: assuming ω_0, ω_0 and S_t (M different values) are fixed constants.
- Simple exponential method: modelling the case where $S_t = 0$, $\omega_1 = 0$ (or constant) and ω_0 changes with time
- Trend corrected exponential method: modelling the case where $S_t = 0$, both ω_1 and ω_0 are changing
- How to model the data if the level, the level growth rate (the trend), and seasonal patterns are changing?

Simple exponential smoothing

图示

Explanation: Simple exponential smoothing



迭代公式:

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

当然

$$0 \leq \alpha \leq 1$$

展开下:

$$\begin{aligned}
l_1 &= \alpha y_1 + (1-\alpha) l_0 \\
l_2 &= \alpha y_2 + (1-\alpha) l_1 = \alpha y_2 + (1-\alpha)\alpha y_1 + (1-\alpha)^2 l_0 \\
l_3 &= \alpha y_3 + (1-\alpha) l_2 = \alpha y_3 + (1-\alpha)\alpha y_2 + (1-\alpha)^2 \alpha y_1 + (1-\alpha)^3 l_0
\end{aligned}$$

通项公式:

$$l_t = \alpha y_t + (1-\alpha) l_{t-1} = \alpha yt + (1-\alpha)\alpha y_{t-1} + (1-\alpha)^2 \alpha y_{t-2} + \ldots + (1-\alpha)^{t-1} \alpha y_1 + (1-\alpha)^t l_0$$

可以观察到:

1. 由于 y_i 都是能得到的，决定预测好坏的就是 α 和 l_0；
2. $y_{\widehat{t+1|1:t}}$ 或者说 l_t 是迭代生成的。
3. y_t 的 t 其实是 从1开始的，但 python 里 list 的第一个位置的 index 是 0，即 y_t 就是 python 里的 Y[t-1]，比如y_1 在 python 里就是 Y[0]

手动实现 Simple exponential smoothing

```
# 伪代码
#第一步确定 α 和 l_0, smoothed_manual / level 存的是 l_0 到 l_t, 同时也是 y_1 到 y_t+1 的预测值

alpha = 0.1
#smoothed_manual = [y[0]]
level = [l_0]

# 根据迭代公式生成新的 level
for i in range( data_length - 1 ):
  level.append( alpha * Y[i] + (1 - alpha) * level[i] )

```

- 为什么 range 的范围是 data_length - 1:

  举例子说明，假如有1000 个数据(y_1 到 y_{1000})，data_length 为 1000。我们想根据这组数据做 Simple exponential smoothing, 得到1000个预测值($\hat{y_1}$ 到 $\hat{y}_{1000}$)，但是也就是 l_0 到 l_{999}，迭代生成 l_{999} 的时候要用到 Y[998] 和 l_{998} 。 所以 i 的范围应该是[0, 998], 也就是 range(999), 换句话说 range(data_length -1)

  - 但这里的 $\hat{y}_1$ 是由 自己设置的l_0 得到的， 是不可信的

Pandas EWM 指数加权滑动

如果要求滑动平均 EWMA 的话，先 ewm() 再 mean()

`DataFrame.ewm`(*alpha*, *adjust*)

- alpha 和手动的 alpha 一样
- adjust

  > When adjust is True (default), weighted averages are calculated using weights (1-alpha)**(n-1), (1-alpha)**(n-2), …, 1-alpha, 1.
  >
  > When adjust is False, weighted averages are calculated recursively as:
  >
  > weighted_average[0] = arg[0]; weighted_average[i] = (1-alpha)*weighted_average[i-1] + alpha*arg[i].

```
smoothed = y.ewm(alpha=0.05, adjust=False).mean()
```

寻找最佳 alpha

1. 首先实现 SSE 的公式

$$SSE = \Sigma(y_i - \hat{y_i})$$

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

2. 遍历(0,1) 的所有 alpha 值，计算其对应的 SSE 的值。

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

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

- smoothed $\hat{y_2}$ 到 $\hat{y_{t+1}}$, smoothed[:-1] $\hat{y_2}$ 到 $\hat{y_t}$
- y.values 是 y 对应的一维 array, y.values[1:] 指的是 y_2 到 y_t

3. 用 np.argmin 函数找到 SSE 最小的 alpha 值

   np.argmin() 返回 array 最小值的 index

```
optimal_alpha_one = alphas[ np.argmin(sse_one) ]
```

寻找最佳的 l_0

Tutorial 的意思是寻找 l_0 的过程和 α 一样，就是 比较 SSE 的大小。

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.

l_0 一般有两种方式获得，

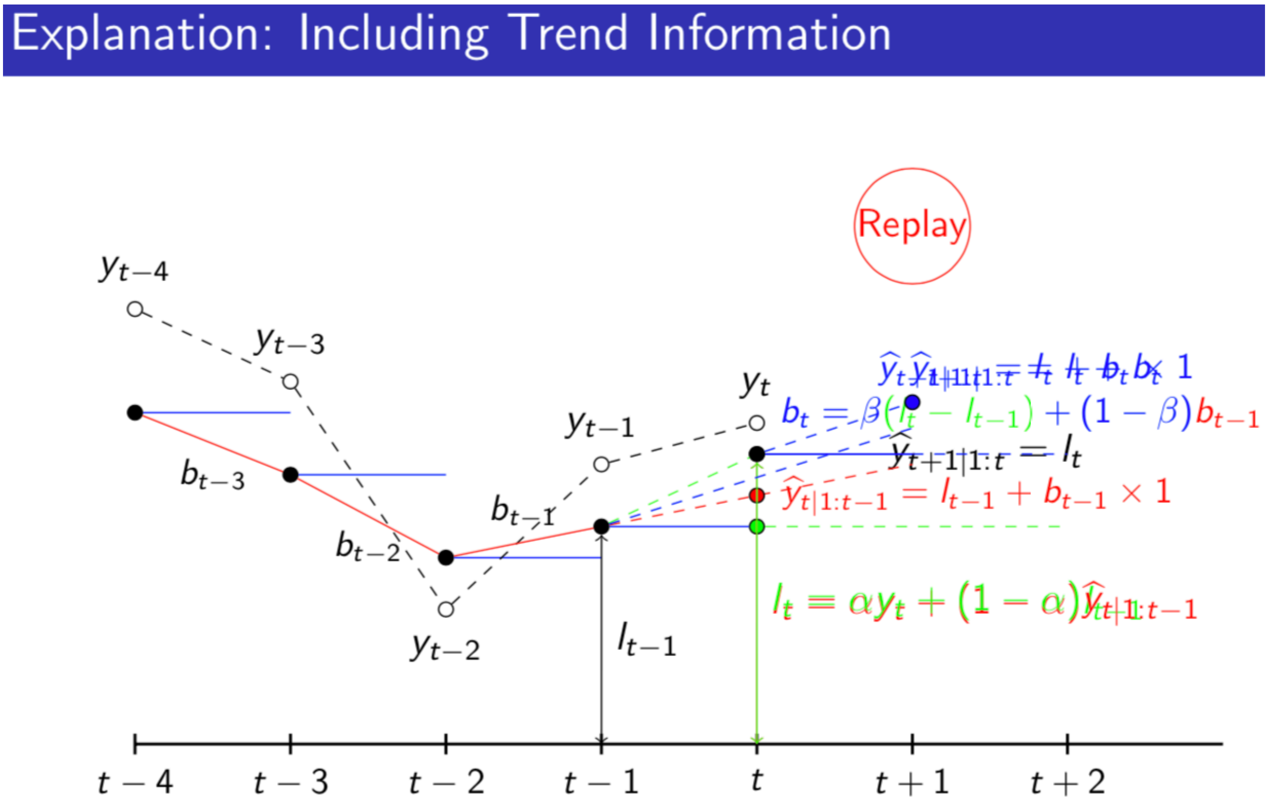
Lec5 P12

- We left l_0 unspecified above.
- How should we set it?
  - Use the average of very initial observations, i.e., y_1, y_2, y_3 etc., or even simply y_1
  - Take l_0 as a parameters, and use an algorithm to estimate it.

- 后一种在 lec 6 里有说，用的是线性回归

Holt’s linear method

图示



递推公式：

$$
\begin{aligned}
\text{Forecast equation} \quad & \hat{y}_{t+h|t} = \ell_t + hb_t \\
\text{Level equation} \quad & \ell_t = \alpha y_t + (1-\alpha)(\ell_{t-1} + b_{t-1}) \\
\text{Trend equation} \quad & b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1},
\end{aligned}
$$

这里的 h 指的是 h-step-ahead forecast 的 h

可以观察到：

1. 需要调的参数是 α 和 β
2. l_0 和 b_0 对结果好坏至关重要

初始化参数

```
alpha = 0.1
beta = 0.1
l = [y[0]]
b = [y[1] - y[0]]

Y = y.tolist()
```

- 设定 α 和 β 值，以及 l_0 和 b_0
- Y 存的是 y_t ， 也就是 y_1 y_2 ...

Smoothing without forecasting

```
holtsmoothed_manual = []
for i in range(len(y)):
    l.append(alpha * Y[i] + (1 - alpha) * (l[i] + b[i]))
    b.append(beta * (l[i+1] - l[i]) + (1 - beta) * b[i])
    holtsmoothed_manual.append(l[i+1])
```

- holtsforecast_manual 存的是 $\hat{y}_{t+0|t}$ = l_t , t [1,len(y)]
- Y 存的是 y_t ， 也就是 y_1 y_2 ...
- len(y) = 312 ,range(len(y)) = [0,311]
  - i = 0 $l_1 = \alpha y_0 + (1-\alpha)l_0$
  - i = 1 $l_2 = \alpha y_1 + (1-\alpha)l_1$
  - i = 2 $l_3 = \alpha y_2 + (1-\alpha)l_2$
  - ......
  - i = 311 $l_{312} = \alpha y_{311} + (1-\alpha)l_{311}$
- holtsmoothed_manual 存的是 [l_1 到 l_{312}]

1-step forecasting(12 months)

```
1  holtsforecast_manual = []
2  
3  for i in range(len(y)+12):
4      if i == len(Y):
5          Y.append(l[-1] + b[-1])
6  
7      l.append(alpha * Y[i] + (1 - alpha) * (l[i] + b[i]))
8      b.append(beta * (l[i+1] - l[i]) + (1 - beta) * b[i])
9  
10     holtsforecast_manual.append(l[i] + b[i])
```

- holtsforecast_manual 存的是 $\hat{y}_{t+1|t}$ = $l_t + b_t$, t [0,len(y) + 11]
- 第 7， 8， 9 行和之前的 smoothed 过程一样，因为 forecast 是在 smoothed 结果基础上进行的
- 第四行和第五行在Y 原来的312 个数据 smooth 完之后，将 $l_t + b_t$ 作为新的 y_{t+1} 用来预测。
- range(len(y) + 12) 范围是[0，323]
- Y 存的是 y_t ， 也就是 y_1 y_2 ...
- i = 0 $l_1 = \alpha y_1 + (1-\alpha)(l_0 + b_0)$ $b_1 = \beta(l_1 - l_0) + (1-\beta)b_0$ $\hat{y}_1 = l_0 + b_0$
- i = 1 $l_2 = \alpha y_2 + (1-\alpha)(l_1 + b_1)$ $b_2 = \beta(l_2 - l_1) + (1-\beta)b_1$ $\hat{y}_2 = l_1 + b_1$
- i = 2 $l_3 = \alpha y_3 + (1-\alpha)(l_2 + b_2)$ $b_3 = \beta(l_3 - l_2) + (1-\beta)b_2$ $\hat{y}_3 = l_2 + b_2$
- ......
- i = 311 $l_{312} = \alpha y_{312} + (1-\alpha)(l_{311} + b_{311})$ $b_{312} = \beta(l_{312} - l_{311}) + (1-\beta)b_{311}$ $\hat{y}_{312} = l_{311} + b_{311}$

之后

i == len(Y) = 312 条件成立，Y[312] = $y_{313} = l_{312} + b_{312}$

- i = 312 $l_{313} = \alpha y_{313} + (1-\alpha)(l_{312} + b_{312})$ $b_{313} = \beta(l_{313} - l_{312}) + (1-\beta)b_{312}$ $\hat{y}_{313} = l_{312} + b_{312}$
- ....
- i = 323 $l_{324} = \alpha y_{324} + (1-\alpha)(l_{323} + b_{323})$ $b_{324} = \beta(l_{323} - l_{323}) + (1-\beta)b_{322}$ $\hat{y}_{324} = l_{323} + b_{323}$
- holtsforecast_manual 存的就是$\hat{y}_1$ .... $\hat{y}_{324}$

2-step forecasting(12 months)

```
holtsforecast_manual2 = []

for i in range(len(y)+12):
    if i == len(Y):
        Y.append(l[-1] + 2*b[-1])

    l.append(alpha * Y[i] + (1 - alpha) * (l[i] + b[i]))
    b.append(beta * (l[i+1] - l[i]) + (1 - beta) * b[i])

    holtsforecast_manual.append(l[i] + 2*b[i])
```

- 和 1-step 的区别在于 $\hat{y}_{t+2|t} = l_t + 2b_t$

n-step forecasting (m months)

```
def holt(n,m):
  holtsforecast_manual2 = []

  for i in range(len(y)+n):
      if i == len(Y):
          Y.append(l[-1] + m*b[-1])

      l.append(alpha * Y[i] + (1 - alpha) * (l[i] + b[i]))
      b.append(beta * (l[i+1] - l[i]) + (1 - beta) * b[i])

      holtsforecast_manual.append(l[i] + m*b[i])
```

- 和 1-step 的区别在于 $\hat{y}_{t+n|t} = l_t + nb_t$