QBUS6840: Tutorial 8 – AR Models

Objectives

- Enforce stationarity through differencing
- Interpret ACF/PACF plots
- Understand AR Models
- Develop Python skills

In the previous tutorials, we have learnt the basic smoothing methods for time series data, i.e. Simple Exponential Smoothing, Holt's Linear Trend Smoothing and Holt-Winters' Smoothing model. These models have significant advantages such as easy to learn and train and somehow could produce reliable forecasting results by giving more significance for the recent observations.

However, there are 2 major limitations we need to address. Firstly, it produces forecasts **that lag behind the actual trend**. Secondly, Holt-Winters' method is **best used for forecasts that are short-term and in the absence of cyclical or residual variations**. As a result, forecasts aren't accurate when data with cyclical or residual variations are present. As such, this kind of averaging won't work very well.

In order to address the above issues, in Week 08 and Week 09 tutorial sessions, we will start a new topic: the **A**uto**R**egressive **I**ntegrated **M**oving **A**verage (**ARIMA**) model, which consist 3 major components:

- The **AR part** of ARIMA indicates that the evolving variable of interest y_t is regressed on its own lagged (i.e., prior $y_{t-1:t-p}$) values.
- The **MA part** indicates that the regression residual ε_t is actually a linear combination of error terms $\varepsilon_{t-1:t-q}$ whose values occurred contemporaneously and at various times in the past.
- The **I (for "integrated")** indicates that the data values have been replaced with the difference between their values and the previous values (and this differencing process may have been performed more than once) so that the **ARIMA could handle the non-stationary data**. (Note that the AR, MA and ARMA model are still require a weakly stationary data as input!!!)

The purpose of each of these features is to make the model fit the data as well as possible.

In this week, we mainly focus on the implementation of AR model.

Mathematically, an AR model with p-th order could be defined as:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t$$

$$y_t = c + \sum_{i=1}^{p} \phi_i y_{t-i} + \varepsilon_t$$

where c is a constant value, ϕ_1 ... ϕ_p are the parameters for the model, and ε_t is white noise.

This can be equivalent to the backshift representation with the operator B as:

$$y_t = c + \sum_{i=1}^{p} \phi_i B^i y_t + \varepsilon_t$$

so that moving the summation term to the left side and using polynomial notation, we have:

$$\varepsilon_t = y_t - \sum_{i=1}^{p} \phi_i B^i y_t - c$$

$$\varepsilon_t = (1 - \sum_{i=1}^{p} \phi_i B^i)(y_t - \mu)$$

where $\mu = 1/(1 - \phi_1 - \cdots - \phi_p)$.

An AR model can thus be viewed as the output of a filter which input is white noise. (Don't panic if you can't understand this part so far. We will talk about this in Week 08 Lecture session.)

However, for an AR(p) model, some parameter constraints are necessary for the model to secure a weakly stationary. More specifically, processes in the AR(1) model with only $|\boldsymbol{\phi_1}| < \mathbf{1}$ could be viewed as **weakly stationary** (wide-sense stationary).

Question:
What are the constraints for AR(2) model?
For AR(2) model, we should have the following constraints: $|\phi_2| < 1$, $\phi_2 + \phi_1 < 1$ and $\phi_2 - \phi_1 < 1$

Selecting a suitable order value p can be tricky. We need to observe the PACF curve to find out the cutting off value as the order for our AR model. The partial autocorrelation of an AR(p) process equates zero at lag which is bigger than order of p and provide a good model for the correlation in $y_{1:p+1}$, so the appropriate maximum lag is the one beyond which the partial autocorrelations are all zero.

For more details of AR(p) model, please refer to the guide book and wiki page:
https://otexts.com/fpp2/AR.html
https://en.wikipedia.org/wiki/Autoregressive_model

1. Load and Inspect the Data

Import `pyplot, numpy` and `pandas` as usual. Then import the following new libraries:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import statsmodels as sm
import statsmodels.api as smt
```

We will use both `statsmodels` time series plots and pandas plotting tools.

```
data = pd.read_csv('data.csv')
data = data['Data']
```

Then plot the data:

```
plt.figure()
plt.plot(data)
```

2. Check if the time series is stationary

Is the time series stationary?
What three rules can we use to visually check for stationarity?

Plot the mean, variance of the data, and the covariance of the i-th term and the (i + k)-th term to check its stationarity.

Therefore, we must apply a transformation to make the data stationary. The simplest transformation is differencing. Usually first or second order differencing is enough.

```
# Calculate difference series, data[1]-data[0],
diff_data = pd.Series.diff(data)

# Checking the first entry in diff_data
diff_data.iloc[0]
```

Why is the first value a NaN?

Let's remove the NaN and plot the data

```
diff_data = diff_data.dropna()

# Plot the differenced data
plt.figure()
```

```
plt.plot(diff_data)
```

Is the data now stationary?

3. Inspect ACF and PACF plots

ACF could be used for analyzing the stationarity. In general, it can be shown that for non-seasonal time series:

- If the Sample ACF of a non-seasonal time series “dies down” or “cuts off" reasonably quickly, then the time series should be considered stationary.
- If the Sample ACF of a non-seasonal time series “dies down" extremely slowly or not at all, then the time series should be considered non-stationary.

`Statsmodels` has a built-in ACF plotting function. Let’s examine the ACF and PACF plots for the original data first.

```
smt.graphics.tsa.plot_acf(data, lags=30, alpha = 0.05)
smt.graphics.tsa.plot_pacf(data, lags=30, alpha=0.05)
plt.show()
```

Lags defines the number of time-lagging and `alpha=0.05` defines the confidence interval where the standard deviation is computer according to Bartlett’s formula. You can change 0.05 to other values (i.e. 0.1) to see what happens.

Compare the original data ACF/PACF with the differenced data ACF/PACF:

```
smt.graphics.tsa.plot_acf(diff_data, lags=30, alpha =
0.05)
smt.graphics.tsa.plot_pacf(diff_data, lags=30, alpha =
0.05)
plt.show()
```

4. Generate Samples from an AR process

`Statsmodels` provides functions to generate samples from AR processes. This is useful for comparison with existing data and analyzing the behavior of model parameters.

Start by setting up the parameters:

```
np.random.seed(12345)

arparams = np.array([0.9])
zero_lag = np.array([1])
```

```
ar = np.r_[1, -arparams]  # add zero-lag (coefficient 1)
and negate
c = 0
```

Think about why we add '–' in front of arparams?

Then generate samples from a process of the form $y_t = 0 + 0.9\, y_{t-1} + \varepsilon_t$

```
y = c + sm.tsa.arima_process.arma_generate_sample(ar =
ar, ma = zero_lag, nsample = 250)
```

Plot the times series sample:

```
plt.figure()
plt.plot(y1)
plt.title("Autoregressive (AR) Model")
```

5. Inspect the ACF and PACF of the AR process

```
smt.graphics.tsa.plot_acf(y, lags=30, alpha = 0.05)
smt.graphics.tsa.plot_pacf(y, lags=30, alpha = 0.05)
```

Check out the PACF plot, when does the plot cuts out?
Does the ACF plot dies down eventually?

6. Calculate mean and variance of the AR process

We can use the equation below to calculate the unconditional mean of the process:

$$E(y_t) = \frac{c}{1 - \phi_1}$$

```
y1_uncond_mean = c / (1 - arparams[0])
print(y1_uncond_mean)

diff_y = pd.Series.diff(pd.Series(y)).dropna()

sample_mean = np.mean(diff_y)
print(sample_mean)
```

What is the final result of this unconditional mean?
Since c = 0, this result will be 0 anyway.

To calculate the variance, use the following equation:

$$Var(y_t) = \frac{\sigma^2}{1 - {\phi_1}^2}$$

`arma_generate_sample` uses `np.random.randn` to generate epsilon ε. Therefore we know that by default $Var(\varepsilon_t) = \sigma^2$ and the $E(\varepsilon_t) = 0$.

```
sigma2 = 1

y1_uncond_var = sigma2 / (1 - np.power(arparams[0],2))

sample_var = np.var(diff_y)
print(sample_var)
```