Name : Tang Sophearin

Affiliation : Department of Intelligence Science and Technology, Nishida Laboratory

Student ID : 6930-30-3963

Pattern Recognition – Sequential Data

1. *Discuss the similarity and difference between Kalman Filter and Hidden Markov Model (forward algorithm).*

Umbrella term 'Dynamic Bayesian networks' consists of two models: HMM and State Space Model.

In each model, we specify:

- Hidden space : Discrete or Continuous.
- Hidden state dynamics : Linear or Non-linear.
- Nature of observations : Typically, conditionally multinomial or Normal.
- Measurement model connecting hidden state to observations.

We can perform various statistical estimates on each specific model, such as:

- Filtering : finding estimates for the current state.
- Prediction : finding estimates for future state.
- Smoothing : finding estimates for past state.

Both Kalman Filter and Forward algorithm are used in filtering.

However, Kalman Filter is used in situation where the state is continuous, the state dynamics and measurement linear and all noise is Normal. Forward algorithm does almost the same things, but the state is discrete.

2. *Explain how to estimate the coefficients a_1 and a_2 of the auto-regressive model (AR model)*

$$y_t = a_1 y_{t-1} + a_2 y_{t-2} + \varepsilon_t$$

from observed sequence data $y_1, y_2, \dots, y_T \in R$.

There are many ways to estimate the coefficient of AR model, such as OLS (ordinary least squares) method or Yule-Walker method.

OLS is a direct way to estimate the coefficient of AR model. However, besides AR(1), it is not very robust and easy to derive the estimation. For our example of AR(2), I will use Yule-Walker Equations to estimate the coefficient a_1 and a_2.

Solution:

We have:

$$y_t = a_1 y_{t-1} + a_2 y_{t-2} + \varepsilon_t \quad (1)$$

Where we assume that:

$$\varepsilon_t \sim Normal(0, \sigma^2);\ \ y_t\ is\ stationary \rightarrow E[y_t] = c, c\ is\ constant.$$

If we take expectation from both sides of the model, we get:

$$\mu = a_1 \mu + a_2 \mu \quad (2)$$

Subtract (1) by (2)

$$y_t - \mu = a_1 (y_{t-1} - \mu) + a_2 (y_{t-2} - \mu) + \varepsilon_t$$

Define $\tilde{y}_t = y_t - \mu$, then $E[\tilde{y}_t] = 0$, and

$$\tilde{y}_t = a_1 \tilde{y}_{t-1} + a_2 \tilde{y}_{t-2} + \varepsilon_t$$

Multiply both side with $\tilde{y}_{t-k}$ and take expectation

$$E[\tilde{y}_t \tilde{y}_{t-k}] = a_1 E[\tilde{y}_{t-1} \tilde{y}_{t-k}] + a_2 E[\tilde{y}_{t-2} \tilde{y}_{t-k}] + E[\varepsilon_t \tilde{y}_{t-k}]$$

Since $E[\tilde{y}_t] = 0$ and assume $E[\varepsilon_t \tilde{y}_{t-k}] = 0$ in case (k>0) because there are no correlation between noise and past value, we get autocovariance function as follow

$$\gamma_{-k} = a_1 \gamma_{-k+1} + a_2 \gamma_{-k+2}$$

Since $\gamma_{-k} = \gamma_k$, we have

$$\gamma_k = a_1 \gamma_{k-1} + a_2 \gamma_{k-2}$$

Divide by γ_0 which is the variance (covariance of data with itself), we obtain Yule-Walker Equation

$$r_k = a_1 r_{k-1} + a_2 r_{k-2}$$

For k = 1,2

$$r_1 = a_1 r_0 + a_2 r_{-1}$$
$$r_2 = a_1 r_1 + a_2 r_0$$

But $r_{-k} = r_k$ and $r_0 = \frac{\gamma_0}{\gamma_0} = 1$

$$r_1 = a_1 + a_2 r_1$$

$$r_2 = a_1 r_1 + a_2$$

We can rewrite in matrix form

$$\begin{bmatrix} r_1 \\ r_2 \end{bmatrix} = \begin{bmatrix} 1 & r_1 \\ r_1 & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$$

Replace left side with $\hat{b}$ and right side with $\hat{R}\hat{\Phi}$

$$\hat{b} = \hat{R}\hat{\Phi} \rightarrow \hat{R}^{-1} b = \hat{\Phi}$$

Finally, as we can see, we can first calculate variance, 1st and 2nd autocovariance $\gamma_0, \gamma_1, \gamma_2$; then calculate ρ_1 and ρ_2 from data; then we can calculate the coefficient a_1 *and* a_2.

The generalized version is:

$$\begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ \vdots \\ r_{p-1} \\ r_p \end{bmatrix} = \begin{bmatrix} 1 & r_1 & r_2 & \cdots & r_{p-1} \\ r_1 & 1 & r_1 & \cdots & r_{p-2} \\ r_2 & r_1 & 1 & & r_{p-3} \\ & \vdots & & \ddots & \vdots \\ r_{p-2} & r_{p-3} & r_{p-4} & \cdots & r_1 \\ r_{p-1} & r_{p-2} & r_{p-3} & \cdots & 1 \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \vdots \\ \phi_{p-1} \\ \phi_p \end{bmatrix}$$

$\hat{b}$ $\hat{R}$ $\hat{\phi}$

$$\hat{b} = \hat{R}\hat{\phi}$$

3. *Apply AR model estimation to any real data and evaluate short-term prediction.*

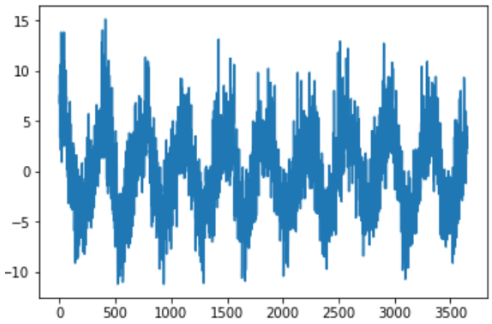
I used daily minimum temperature in Melbourn, Australia as data and applied AR(2) to evaluate short term prediction.

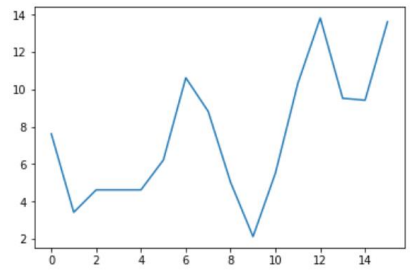
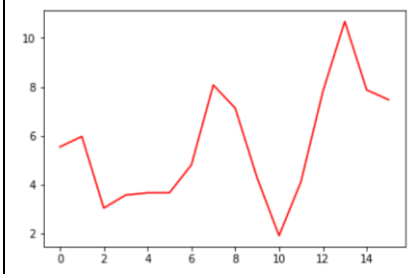
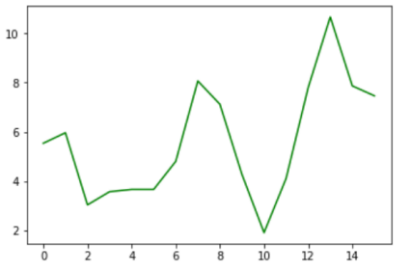
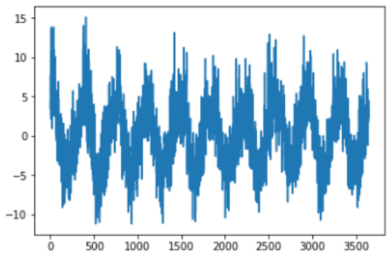
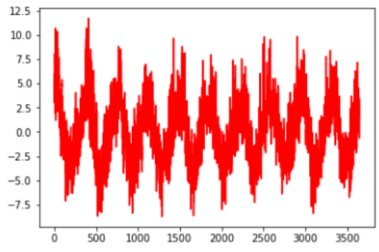
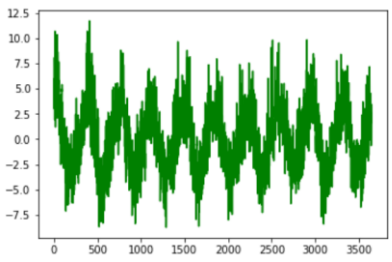
As a result, I got 6.577323 mean squared error between actual data and predictions. I also tried using built-in AR library from statsmodels.tsa.ar_model. The mean square error from the built-in library was 6.577318, which is very close to manual implementation.

$$r_1 = 1, r_2 = 0.77448, r_3 = 0.630632$$

$$\mathrm{a} = [0.71484839, 0.07699588]$$

Original Data



	Original Data	Manual Prediction	Built-in Library Prediction
First 16 data points with first 2 data points removed (lags)			
All data with first 2 data points removed (lags)			

Code:

```
1.  import pandas as pd
2.  import matplotlib.pyplot as plt
3.  import numpy as np
4.  from pandas import DataFrame
5.  from pandas import concat
6.  from sklearn.metrics import mean_squared_error
7.  from statsmodels.tsa.ar_model import AR
8.
9.  #import data
10. data = pd.read_csv("daily-minimum-temperatures-in-me.csv", header=0)
11. data.columns = ['Date', 'Temperature']
12. data = data[['Temperature']]
13.
14. #subtract mean
15. mean = np.mean(data)
16. data = (data - mean)
17.
18. plt.plot(data)
19. plt.show()
20.
21. #calculate variance, first and second autocovariance
22. values = DataFrame(data.values)
23. data_lag = concat([values, values.shift(1), values.shift(2)], axis=1)
24. data_lag.columns = ['t', 't-1', 't-2']
25. autocov = data_lag.cov()
```

```
print(autocov)


#calculate b and R -> a
r = autocov.iloc[0]/autocov.iloc[0]['t']
b = r[1:3]
R = np.empty((2,2))
R[0,0] = 1
R[0,1] = r[1]
R[1,0] = r[1]
R[1,1] = 1

a = np.dot(np.linalg.inv(R), b)

prediction = list()
data_arr = data.values
for i in range(2, len(data)):
    prediction.append(a[0]*data_arr[i-1] + a[1]*data_arr[i-2])

error = mean_squared_error(data_arr[2:len(data)],prediction)
print(error)


#test with built-in AR library
model = AR(data_arr)
model_fit = model.fit(maxlag=2)
lib_prediction = model_fit.predict()
lib_error = mean_squared_error(data_arr[2:len(data)],lib_predictions)
print(lib_error)


#plot actual and predicted data

#close-in
num_data = 16
plt.plot(range(num_data), data_arr[2:num_data+2])
plt.show()
plt.plot(range(num_data), prediction[:num_data], 'red')
plt.show()
plt.plot(range(num_data), lib_prediction[:num_data], 'green')
plt.show()

#all data
num_data = len(data_arr)
plt.plot(range(num_data-2), data_arr[2:num_data])
plt.show()
plt.plot(range(num_data-2), prediction[:num_data], 'red')
plt.show()
plt.plot(range(num_data-2), lib_prediction[:num_data], 'green')
plt.show()
```

Out of curiosity, I also tried applying PCA and AR on a waterfall video. I used PCA to reduce the video dimension from 640 x 360 to 200 (CCR ~ 90%) and using autorgression order 10^{th} to generate 1200 frames of the video.

Link to video: https://www.youtube.com/watch?v=cgmlImX4M_A&feature=youtu.be

Link to source code: https://drive.google.com/drive/folders/1Z-kTiqsMuCYgwMzA9E7NhPeW_krtY5Dv