Machine Learning in Finance
Section I3
Lecture 9

Michael G Sotiropoulos

NYU Tandon
Deutsche Bank

5-Apr-2019

Contents

We have a vector $\boldsymbol{x}$ of inputs (or features, or regressors), and an associated output y (or target, or regressand). We want to

1. Learn (or fit) the relationship between $\boldsymbol{x}$ and y from the data (in-sample)
2. Predict an output y given a new input $\boldsymbol{x}$ not seen so far (out-of-sample)

A discrete output $y \in \{C_k\}$ defines a **classification** problem (C_k is class label)
A continuous output $y \in \mathbb{R}$ defines a **regression** problem

- Both $\boldsymbol{x}$ and y are random variables.
- The system is fully described by the (unknown) joined density $p(\boldsymbol{x}, y)$.

NOTE: The total empirical information entering the problem is the observations $(\boldsymbol{x}, y)$. All other quantities such as weights w, parameters θ, basis functions ϕ, distributions p, etc. are entities used to define a specific model.

- A classification model provides a mapping: $\boldsymbol{x} \to \hat{C}_k$
- A regression model provides a mapping: $\boldsymbol{x} \to \hat{y}$

Machine learning algorithms are computational methods for constructing the above mappings.

In supervised learning we have the notion of **loss function** L, which measures the deviation between the model output $\hat{C}_k$ or $\hat{y}$ and the actual output C_k or y. Model parameters are tuned so as to minimize the expected loss $\mathbb{E}[L]$.

Classification problem: The space of inputs is cut up into regions R_j so that if $\boldsymbol{x} \in R_j$ then $\boldsymbol{x} \to \hat{C}_j$. If the true class is C_k and the model outputs $\hat{C}_j$ we have a loss L_{kj}. The expected loss is

$$\mathbb{E}[L] := \sum_k \sum_j \int_{R_j} L_{kj} p(\boldsymbol{x}, C_k)\, d\boldsymbol{x} \tag{1}$$

$$= \sum_j \int_{R_j} \sum_k L_{kj} p(C_k|\boldsymbol{x})\, p(x)\, d\boldsymbol{x} \tag{2}$$

The loss minimizing mapping is: $\boldsymbol{x} \to \hat{C}_j = \min_j \sum_k L_{kj} p(C_k|\boldsymbol{x})$

There are in general three ways of solving the problem

1. **Generative**: model the joint distribution $p(\boldsymbol{x}, C_k)$; computationally heavy
2. **Discriminative**: model the posterior density $p(C_k|\boldsymbol{x})$ (classifier algos)
3. **Function approximation**: construct the mapping $\boldsymbol{x} \to \hat{C}_j$ in one step (neural nets)

Regression Problem: The expected loss for output variable $y \in \mathbb{R}$ becomes

$$\mathbb{E}[L] := \int \int L(y, \hat{y}(\boldsymbol{x}))\, p(\boldsymbol{x}, y)\, d\boldsymbol{x} dy \tag{3}$$

Using the square loss function

$$L(y, \hat{y}(\boldsymbol{x})) := (\hat{y}(\boldsymbol{x}) - y)^2 \tag{4}$$

the expected loss can be written as

$$\mathbb{E}[L] = \underbrace{\int (\hat{y}(x) - \mathbb{E}[y|\boldsymbol{x}])^2\, p(\boldsymbol{x})\, d\boldsymbol{x}}_{\text{model}} + \underbrace{\int (\mathbb{E}[y|\boldsymbol{x}] - y)^2\, p(\boldsymbol{x})\, d\boldsymbol{x}}_{\text{noise}} \tag{5}$$

The loss minimizing mapping is: $\boldsymbol{x} \to \hat{y}(\boldsymbol{x}) = \mathbb{E}[y|\boldsymbol{x}]$.

Again, there are in general three ways of solving the problem

1. **Generative**: model the joint distribution $p(\boldsymbol{x}, y)$; computationally heavy, but the full stochastic process can be simulated
2. **Discriminative**: make assumptions about the noise, and model the conditional expectation $\mathbb{E}[y|\boldsymbol{x}]$ (regression models)
3. **Function approximation**: construct the optimal mapping $\boldsymbol{x} \to \hat{y}(\boldsymbol{x})$ in one step (neural nets)

In batch mode all data are available at the start and are processed in one step.

Example: **Batch linear regression** with M basis functions $\phi_j(\boldsymbol{x})$ and weights w_j The regression function that models $\mathbb{E}[y|\boldsymbol{x}]$ is

$$f(\boldsymbol{x}, \boldsymbol{w}) = \sum_{j=0}^{M-1} w_j \phi_j(\boldsymbol{x}) = \boldsymbol{w}^T \phi(\boldsymbol{x}) \tag{6}$$

A regression model with Gaussian noise takes the form

$$y = f(\boldsymbol{x}, \boldsymbol{w}) + \epsilon. \tag{7}$$

- The errors are zero-mean normal, so they follow the normal density $\mathcal{N}(\epsilon|0, \Sigma)$ with Σ the error covariance matrix. Therefore

$$p(y|\boldsymbol{x}, \boldsymbol{w}, \Sigma) = \mathcal{N}(y|f(\boldsymbol{x}, \boldsymbol{w}), \Sigma) \tag{8}$$

- Since the observations are assumed IID, the likelihood function $\mathcal{L}(\boldsymbol{w}, \Sigma)$ becomes a product over the N observations in the sample

$$\mathcal{L}(\boldsymbol{w}, \Sigma) := p(\{y\}|\{\boldsymbol{x}\}, \boldsymbol{w}, \Sigma) = \prod_{n=1}^{N} \mathcal{N}(y_n|f(\boldsymbol{x}_n, \boldsymbol{w}), \Sigma) \tag{9}$$

Maximizing the log-likelihood, $\log \mathcal{L}$, with respect to the weights $\boldsymbol{w}$ gives the estimates

$$\boldsymbol{w}_{ML} = \left(\Phi^T \Phi\right)^{-1} \Phi^T \boldsymbol{y} \tag{10}$$

with the design matrix built from the inputs $\boldsymbol{x}$ and the basis functions ϕ_j as

$$\Phi = \begin{pmatrix} \phi_0(\boldsymbol{x}_1) & \phi_1(\boldsymbol{x}_1) & \cdots & \phi_{M-1}(\boldsymbol{x}_1) \\ \phi_0(\boldsymbol{x}_2) & \phi_1(\boldsymbol{x}_2) & \cdots & \phi_{M-1}(\boldsymbol{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\boldsymbol{x}_N) & \phi_1(\boldsymbol{x}_N) & \cdots & \phi_{M-1}(\boldsymbol{x}_N) \end{pmatrix} \tag{11}$$

The intercept of the regression is included by the function $\phi_0(\boldsymbol{x}) = 1$.

- Regularization terms can be added to the loss function to avoid overfitting
- By changing the basis functions and error distributions we can construct several types of regression models (polynomial, spline, logistic, Poisson, probit,...).

In online mode, each observation is processed in sequence, using Bayes theorem.

Example: **Online linear regression** with M basis functions $\phi_j(x)$ and weights w_j
In the Bayesian approach the weights $\boldsymbol{w}$ are also random variables with associated distributions. The method is iterative.

1. Start with a **prior** distribution $p(\boldsymbol{w})$.
2. Observe a new data subsample $\mathcal{D}$, and compute the **likelihood** $p(\mathcal{D}|\boldsymbol{w})$
3. Update the **posterior** distribution of the parameters using Bayes theorem

$$p(\boldsymbol{w}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{w})}{p(\mathcal{D})} p(\boldsymbol{w}) \tag{12}$$

If we choose a normal prior $p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}_0, S_0)$, the likelihood in eq. (9) and the above update procedure result in a normal posterior. After N updates

$$p(\boldsymbol{w}|y) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}_N, S_N) \tag{13}$$

$$\boldsymbol{m}_N = S_N \left(S_0^{-1} m_0 + \beta \Phi^T y \right) \tag{14}$$

$$S_N^{-1} = S_0^{-1} + \beta \Phi^T \Phi \tag{15}$$

- A non-informative prior has a very large variance, $S_0 = I_{M\times M}/\alpha$ with $\alpha \to 0$, in which case $m_N \to w_{ML}$

Sequential Data: Characteristics (I)

In finance we are dealing primarily with sequential data, i.e. time series of trade and quote prices and sizes at various sampling or aggregation frequencies. Financial data have special characteristics

- Sequential order matters, because of the presence of serial correlation
- Each observation is not in general an IID sample from a single underlying distribution. Consequently, the likelihood function is not a simple product over observations
- Stationarity of the underlying **joint** distribution is always a concern

A time series x_t is a collection of random variables, fully specified by the (unknown) joint distribution $p(x_1, x_2, \ldots)$. For a sample of N observations, the joint density can be written in terms of the conditionals as

$$p(x_1, x_2, \ldots x_N) = \prod_{t=1}^{N} p(x_t | x_{t-1}, x_{t-2}, \ldots, x_2, x_1) \tag{16}$$

- To control model complexity we use the **Markovian property**, i.e. we *assume* that the conditional densities above depend only on a small number of the most recent observations.

An M^{th} order Markov chain is defined by the property

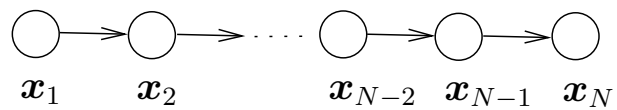
$$p(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_N) =$$

$$p(x_1)\, p(x_2|x_1)\, p(x_3|x_2, x_1) \ldots \prod_{t=M+1}^{N} p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}, \boldsymbol{x}_{t-2}, \ldots, \boldsymbol{x}_{t-M}) \tag{17}$$

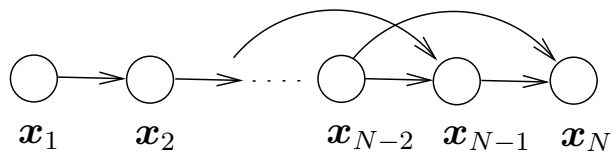
The simplest choice beyond independence is $M = 1$, giving the 1^{st} order chain

$$p(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_N) = p(x_N|x_{N-1})\, p(x_{N-1}|x_{N-2}) \ldots p(x_2|x_1)\, p(x_1) \tag{18}$$

If the conditional distributions $p(x_t|x_{t-1})$ are the same for every time index t, the Markov chain is **homogeneous**. All transitions from one node to the next are controlled by the same transition density, which may depend on common parameters to be learned from the data.



A second order Markov chain is defined by $p(x_t|x_{t-1}, x_{t-2})$, as shown below.



Autoregressive time series models are developed similarly to regression models. A 1st order linear Markov chain with the assumption that errors are zero mean and normally distributed, $\epsilon_t \sim \mathcal{N}(\epsilon_t|0, \sigma_\epsilon^2)$, defines the AR(1) model

$$x_t = \phi_0 + \phi_1 x_{t-1} + \epsilon_t \tag{19}$$

Similarly, a 2nd order linear Markov chain defines the AR(2) model

$$x_t = \phi_0 + \phi_1 x_{t-1} + \phi_2 x_{t-2} + \epsilon_t \tag{20}$$

The lag (or backshift) operator L acts on a variable as $L^i x_t := x_{t-i}$.
We define the p-th order polynomial of the lag operator as

$$\Phi(L) := 1 - \phi_1 L - \phi_2 L^2 - \ldots - \phi_p L^p \tag{21}$$

Then the general AR(p) model can be defined as

$$\Phi(L)\, x_t = \phi_0 + \epsilon_t \tag{22}$$

- Left-multiplying the above by the inverse polynomial Φ^{-1} and expanding it in a power series of L, we get that x_t is a weighted sum of past noise terms

$$x_t = \Phi^{-1}(L)\,\phi_0 + \Phi^{-1}(L)\,\epsilon_t = \mu + \sum_{i=0}^{\infty} \psi_i \epsilon_{t-i} \tag{23}$$

MA representation

Homogeneity is a strong assumption. A weaker one is (weak) stationarity.
A process x_t is **weakly stationary** if it satisfies the following three properties

1. $\mathbb{E}[x_t] = \mu$ (finite constant) for all t
2. $\mathrm{Var}(x_t) = \sigma^2$ (finite positive constant) for all t
3. $\mathrm{Cov}(x_t, x_s) = \gamma(|t-s|)$ (only a function of the lag) for all t and s

The autocorrelation function (**ACF**) ρ depends only on the lag h

$$\rho(h) = \frac{\gamma(h)}{\sigma^2} \tag{24}$$

Assuming that the AR(p) process is stationary, from eq. (23) we can see how the mean μ is related with the intercept ϕ_0 and the other coefficients ϕ_i

$$\phi_0 = (1 - \phi_1 - \phi_2 - \ldots - \phi_p)\,\mu \tag{25}$$

For AR(1) the covariance and ACF are given by (shew)

$$\gamma(h) = \sigma^2 \phi_1^h \frac{1}{1-\phi_1^2}, \quad \rho(h) = \frac{\gamma(h)}{\gamma(0)} = \phi_1^h \tag{26}$$

Stationarity requires $|\phi_1| < 1$. The ACF decays exponentially fast with the lag.

For a general AR(p) model, the ACF at various lags satisfies the **Yule-Walker** equations

$$\rho(h) = \sum_{i=1}^{p} \phi_i \rho(h-i), \quad h = 1, 2, ... \tag{27}$$

These form a linear $p \times p$ system of equations that allow us to solve for the model coefficients $\phi_1, \ldots \phi_p$ in terms of the autocorrelations $\rho(1), \ldots \rho(p)$.

- This is the standard way of fitting an AR(p) for given p.
- Identification of the order p can be done using the partial autocorrelation function (**PACF**) Find the bound for the model

The PACF $\phi_{h,h}$ is the correlation between x_t and x_{t+h} *conditional* on the in-between values $x_{t+1}, x_{t+2}, \ldots x_{t+h-1}$.
It is computed iteratively. We start with $\phi_{1,1} = \rho(1)$.
Then, for each $h > 1$ we fit the regression model

$$x_t = \phi_{0,h} + \phi_{1,h} x_{t-1} + \ldots + \phi_{h,h} x_{t-h} + \epsilon_{h,t} \tag{28}$$

- If x_t follows AR(p) process, then $\phi_{h,h} = 0$ for $h > p$

Motivated by eq. (23), we can build models that only use lagged noise, rather than lagged observations. These are called **Moving Average** (MA) models. An MA(1) model takes the form

$$x_t = \mu + \epsilon_t - \theta_1 \epsilon_{t-1} \tag{29}$$

Clearly $\mathbb{E}(x_t) = \mu$. Moreover, the MA(1) model has the following properties

$$\mathrm{Var}(x_t) = \sigma_\epsilon^2 \left(1 + \theta_1^2\right), \quad \rho(1) = \frac{-\theta_1}{1 + \theta_1^2}, \quad \rho(h) = 0,\ h > 1 \tag{30}$$

An MA(q) model takes the form

$$x_t = \mu + \epsilon_t - \theta_1 \epsilon_{t-1} - \theta_2 \epsilon_{t-2} \ldots - \theta_q \epsilon_{t-q} \tag{31}$$

Using the lag (backshift) operator L and its q-th order polynomial

$$\Theta(L) = 1 - \theta_1 L - \theta_2 L^2 \ldots - \theta_q L^q \tag{32}$$

the MA(q) model can be written as

$$x_t = \mu + \Theta(L)\, \epsilon_t \tag{33}$$

The order of an MA(q) model can be identified by looking at its ACF.

- If x_t follows an MA(q) process, then $\rho(h) = 0$ for $h > q$

Finally, we can **combine** both types of model in an ARMA(p,q) model

$$x_t = \mu + \phi_1 x_{t-1} + \phi_2 x_{t-2} \ldots + \phi_p x_{t-p} + \epsilon_t - \theta_1 \epsilon_{t-1} \ldots - \theta_q \epsilon_{t-q} \tag{34}$$

In terms of the lag polynomials Φ (order p) and Θ (order q) it takes the form

$$\Phi(L)(x_t - \mu) = \Theta(L)\epsilon_t \tag{35}$$

This can also be written as

$$x_t - \mu = \frac{\Theta(L)}{\Phi(L)}\epsilon_t \tag{36}$$

- An ARMA(p,q) process is stationary if the roots of the polynomial $\Phi(z)$ have modulii outside the unit root circle.
- The Dickey-Fuller test tests the null hypothesis that a unit root is present, i.e. the data are non-stationary.
- The augmented Dickey–Fuller (ADF) statistic is a negative number. The more negative the statistic, the stronger the rejection of the null hypothesis for some confidence level

Forecasting with ARMA(p, q) is straightforward. After learning (fitting) the parameters $(\hat{\mu}, \hat{\phi}, \hat{\theta})$, we use the model formula to compute the expected value h steps ahead, $\mathbb{E}[x_{t+h}]$, having observed the series up to x_t.

- Recall, random walk is an ARMA(0,0) process, for which $\mathbb{E}[x_{t+h}] = x_t$

Finally, a process is called ARIMA(p, d, q) if when differentiated d times, the result is an ARMA(p, q) process.
Taking the difference between two successive values can be done by applying the difference operator $1 - L$

$$(1 - L)\, x_t = x_t - x_{t-1} \tag{37}$$

Therefore

$$x_t \sim \mathrm{ARIMA}(p, d, q) \;\Leftrightarrow\; (1 - L)^d\, x_t \sim \mathrm{ARMA}(p.q) \tag{38}$$

Differencing a time series helps removing deterministic time trends.

- If x_t has a linear trend, i.e $\mathbb{E}[x_t] = \beta_0 + \beta_1 t$, then the process $(1 - L)\, x_t$ has constant mean β_1 and no trend
- If x_t has a polynomial trend of order m, then the process $(1 - L)^m\, x_t$ has constant mean and no trend

In the process of fitting time series models we need a goodness of fit criterion.

- When the order of an ARIMA model is fixed to (p, d, q), the parameters are estimated by maximizing the empirical log-likelihood ($\log \hat{L}$).
- As the order increases, the model parameters increase and $\log \hat{L}$ of the fitted model is bound to increase (overfitting).

Information criteria strike a balance between accuracy of fit (large $\log \hat{L}$) and parsimony (small number of parameters k). The most common are:

1. AIC (Akaike Information Criterion)

$$AIC = -2 \log \hat{L} + 2k \tag{39}$$

2. BIC (Bayes Information Criterion)

$$BIC = -2 \log \hat{L} + \log(N)k \tag{40}$$

The model with the smallest criterion value is selected as the "best model".

- BIC typically penalizes more the number of parameters than AIC.
- AIC is asymptotically optimal for selecting the minimum MSE model.

Summarizing time series by moving averages is very common in finance.
A simple moving average (SMA) of window size T is defined as

$$\bar{x}_t = \frac{1}{T}\sum_{i=0}^{T-1} x_{t-i} \tag{41}$$

An exponential moving average (EMA) with discount factor λ ($0 \leq \lambda \leq 1$) is defined as

$$\tilde{x}_t = \lambda \sum_{i=0}^{t-1} (1-\lambda)^i x_{t-i} + (1-\lambda)^t x_0 \tag{42}$$

- The EMA is a linear filter of the time series x_t
- The weights are normalized $\sum_{i=0}^{t-1} \lambda (1-\lambda)^i + (1-\lambda)^t = 1$.
- For large enough t the influence of the initial value x_0 can be ignored

Using the notation $\bar{\lambda} := 1 - \lambda$, the EMA can be written in recursive form as

$$\tilde{x}_t = \lambda x_t + \bar{\lambda}\tilde{x}_{t-1} \tag{43}$$

The **half-life** h of an EMA is the number of steps into the past when the weight of that past value is 1/2.

$$\bar{\lambda}^h = \frac{1}{2} \Rightarrow h = -\frac{\ln 2}{\ln \bar{\lambda}} \tag{44}$$

In the technical trading literature it is customary to express the discount factor λ in terms of a number of periods N. There are two common specifications

1. regular: $\lambda = 2/(N+1)$ for which the half life is $h \approx N \ln 2/2 \approx 0.35N$
2. Wilder: $\lambda = 1/N$ for which the half life is $h \approx N \ln 2 \approx 0.69N$

The one-step ahead EMA forecast is $\hat{x}_{t+1|t} = \tilde{x}_t$
The forecasting error is $e_{t+1|t} = x_{t+1} - \tilde{x}_t$
Using the recursive definition in eq (43) we can write the forecast as

$$\hat{x}_{t+2|t+1} = \tilde{x}_{t+1} = \lambda x_{t+1} + \bar{\lambda}\tilde{x}_t = \lambda x_{t+1} + \bar{\lambda}\hat{x}_{t+1|t} = \hat{x}_{t+1|t} + \lambda\left(x_{t+1} - \hat{x}_{t+1|t}\right)$$

Therefore, we get the error-correction form

$$\hat{x}_{t+2|t+1} = \hat{x}_{t+1|t} + \lambda e_{t+1|t} \tag{45}$$

- The one-step ahead forecast for the current period is the previous period forecast plus the discounted previous period forecast error.

Relation to ARIMA:
Using the definition of the forecast error, we have

$$\begin{aligned} e_{t|t-1} - \bar{\lambda} e_{t-1|t-2} &= (x_t - \tilde{x}_{t-1}) - \bar{\lambda}\,(x_{t-1} - \tilde{x}_{t-2}) \\ &= x_t - x_{t-1} - \tilde{x}_{t-1} + \lambda x_{t-1} + \bar{\lambda}\tilde{x}_{t-2} \\ &= x_t - x_{t-1} \end{aligned} \tag{46}$$

Setting $\theta = \bar{\lambda}$, using the lag operator $Lx_t = x_{t-1}$,and setting $e_{t|t-1} = \epsilon_t$ the above expression becomes

$$(1 - L)\, x_t = (1 - \theta L)\, \epsilon_t \tag{47}$$

- An EMA model provides forecasts equivalent to an ARIMA(0,1,1) model.

Taking the EMA of an EMA yields the DEMA (double exponential smoothing)

$$\tilde{x}_t^{(2)} = \lambda \tilde{x}_t^{(1)} + \bar{\lambda}\tilde{x}_{t-1}^{(2)} \tag{48}$$

This can be shown to be equivalent to an ARIMA(0,2,2) model.

The **pandas** library has significant functionality for processing time series data.

Time-related data types:

- A point in time can be a Python **datetime** or a pandas **Timestamp**
- Differences between times can be Python **timedelta** or pandas **Timedelta**
- Pandas supports time spans with the **Period** class (year, day, hour,...)
- Dataframes can be indexed by time. The index type is **DatetimeIndex**
- Dataframes can be indexed by Period. The index type is **PeriodIndex**
- There is full support of datetime arithmetic and time zone conversion
- Period-aware resampling, grouping, merging

Aggregation or "window" functions: `count, mean, var, std, corr, apply,...`
They operate on three types of windows:

- standard moving or **rolling**: ex. `pd.rolling_corr(x, y, ...)`
- standard **expanding**: ex. `pd.expanding_max(x, ...)`
- **exponentially-weighted** moving: ex. `pd.ewma(x, halflife=...)`

Plotting: pandas plots are time aware and can format results accordingly.

Scikit-Learn does not have time series specific functionality.
We use pandas to do the data processing, and convert to numpy arrays for running Scikit-Learn algorithms.

StatsModels has times series modeling functionality in **statsmodels.tsa**
For ARIMA model fitting we can use the following classes

- `statsmodels.tsa.tools.adfuller`; runs ADF test for stationarity
- `statsmodels.tsa.ar_model.AR`; fits AR(p) models
- `statsmodels.tsa.arima_model.ARMA`; fits ARMA(p,q) models
- `statsmodels.tsa.arima_model.ARIMA`; fits ARIMA(p,d,q) models
- `statsmodels.tsa.holtwinters.SimpleExpSmoothing`; simple ewma
- `statsmodels.tsa.holtwinters.Holt`; Holt-Winters ewma

All classes above have fit and predict methods.

The **statsmodels.tsa.statespace** package contains state-space models.

For details see the notebook **L09-TimeSeries-Examples.ipynb**

Summary and Further Reading

1. We reviewed the principles of supervised learning.
2. We reviewed ARIMA and EMA models.
3. We identified Python tools that implement the above ideas.

Section 13.1 in [CB] has a nice exposition of Markov models.
ARIMA and EMA forecasting are well presented in [MJK] and [HKOS].
Look at the Time Series Analysis examples available in [SM].

References

[CB] Bishop C. "Pattern Recognition and Machine Learning". Springer, 2006

[MJK] Montgomery D.C., Jennings C.L., Kulahci M. "Introduction to Times Series Analysis and Forecasting" 2nd edition. Wiley 2015

[HKOS] Hyndman R.J., Koehler A.B., Ord J.K., Snyder R.D. "Forecasting with Exponential Smoothing". Springer, 2008

[SM] StatsModels. Statistics in Python. Online docs available here