doi: 10.2166/9781789062380_ 0075

Chapter 4

Water demand forecasting | time series data

Jarai Sanneh[1], A. Di Mauro[2], S. Venticinque[2], G. F. Santonastaso[2], A. Di Nardo[2], Yi Wang[3] and Juneseok Lee[1]*

[1]Civil & Environmental Engineering, Manhattan College, Riverdale, NY 10471, USA
[2]Università degli Studi della Campania Luigi Vanvitelli, Aversa 81031, Italy
[3]Electrical and Computer Engineering, Manhattan College, Riverdale, NY 10471, USA
**Corresponding author: juneseok.lee@manhattan.edu*

LEARNING OBJECTIVES

At the end of this chapter, you will be able to:

(1) Apply ARIMA/SARIMA to forecast water demand in time-series data.
(2) Discuss the practical aspects and implications of using Machine Learning to water demand in time-series data.
(3) Build and run time series data using machinear learning techniques (MATLAB and Python).
(4) Interpret modeling results.

4.1 INTRODUCTION

Water demand forecasting is crucial in many aspects of Water Distribution Systems (WDS) because it helps minimize cost, optimize operations, and provide strategies for water conservation (Kofinas *et al.*, 2014). It plays a vital role in the planning, operations, and management of physical assets for water utilities such as pumping stations, treatment plants, tanks, and distribution networks, which rely on future consumption forecasts (Arandia *et al.*, 2015; Billings & Jones, 2008). For instance, water utilities need short-term water demand forecasting in order to provide a more stable urban freshwater supply that will be used in a timely manner 'by adjusting water supply to actual demand and consumption' (Kofinas *et al.*, 2014).

Traditional ***time series forecasting methods*** such as Auto-Regressive Moving Average (ARMA), Auto-Regressive Integrated Moving Average (ARIMA) and Seasonal Auto-Regressive Integrated Moving Average (SARIMA) have been used for decades to forecast water demand using time series historical data. Redondo *et al.* (2018) used ARIMA models to make operational analysis in a drinking water treatment plant by analyzing how the water quality is affected by rainfall. The results showed that the ARIMA models were more accurate for analyzing the water treatment operations using a weekly timescale compared to a daily timescale 'due to significant daily variations in the control parameters of water quality in the plant' (Redondo *et al.*, 2018). Lee and Chae (2016) developed seasonal ARIMA models to make hourly water demand forecasting for micro water grids (Lee & Chae,

2016). Arandia *et al.* (2015) forecasted short-term water demand using SARIMA models to make both offline and online forecasts. The offline forecasts were made using the most recent historical data to 're-estimate the models' while the online forecasts were made by combining the SARIMA models (state-space form) with data assimilation by applying a Kalman Filter (KF) to update the models efficiently (Arandia *et al.*, 2015).

In the past decade, artificial intelligence (AI) had a rapidly growing presence in many applications, including the water sector. Machine learning (ML) techniques are an artificial intelligence approach that has drawn serious attention in water-demand forecasting. Machine learning techniques have the advantage of being able to forecast nonlinear relationships between response variables and their predictors in time series models with the presence of noisy data. The increasing use of smart water metering in the water sector has made available a great amount of data which cannot be processed with traditional methods (Cominola *et al.*, 2015). Therefore, the need has emerged to identify new data analysis techniques able to extract valuable information from available data and support water utilities in their decision systems. Analytics in the Drinking Water Industry support improvements in demand side management and water distribution network efficiencies, lead significant water savings, promote customers' sustainable behaviours, identify peak hours of use, and facilitate water forecast demand modelling (Monks *et al.*, 2019).

In this context, machine learning techniques (MLT) represent the key to many challenges. In the literature, especially in the last five years, various MLT for water demand analysis and forecasting have been proposed showing how they can also be applied in the water sector (Pesantez *et al.*, 2020; Rahim *et al.*, 2020; Villarin & Rodriguez-Galiano, 2019; Xenochristou *et al.*, 2018).

4.2 TIME SERIES DATA ANALYSIS

A time series, consisting of a sequence of numerical observations recorded successively in time, has an intrinsic feature of dependence between adjacent observations, which is analyzed using time series analysis (Box *et al.*, 2016). ARIMA and SARIMA models utilize historical time series data and consist of a three-step iterative process: identification, estimation, and diagnostics checking (Box *et al.*, 2016).

4.2.1 ARIMA model

An ARIMA model is denoted as ARIMA(p,d,q) and is expressed using the mathematical formulations given in Equations (4.1)–(4.4) (Lee & Chae, 2016):

$$Y_t = \mu + \sum_{k=1}^{p} \varnothing_k Y_{t-k} + \in_t \tag{4.1}$$

$$Y_t = C + \in_t + \sum_{k=1}^{q} \varnothing_k \epsilon_{t-k} \tag{4.2}$$

$$Y_t = \mu + \sum_{k=1}^{p} \varnothing_k Y_{t-k} + \in_t + \sum_{k=1}^{q} \theta_k \epsilon_{t-k} \tag{4.3}$$

$$\varnothing_p(B)(1-B)^d Y_t = \theta_q(B)\epsilon_t \tag{4.4}$$

where $\varnothing$ = autoregressive or damping parameter; θ = moving average parameter; μ = mean value of the process; ϵ_t = forecast error at time t, in which ϵ_t is assumed to follow a normal (0, σ) distribution, σ = standard deviation of the process (Lee & Chae, 2016). Equation (4.1) defines an autoregressive process of order p, AR(p), 'which predicts values from previous values'; Equation (4.2) defines a

moving average process of order q, MA(q), 'which accounts for previous random trends'; Equation (4.3) defines an autoregressive moving average process of order (p,q), ARMA(p,q); and Equation (4.4) defines an autoregressive integrated moving average process of order (p,q) differenced by order d, ARIMA(p,d,q) (Lee & Chae, 2016).

4.2.2 SARIMA model

A SARIMA or seasonal ARIMA model is obtained when an ARIMA model has a seasonal component (periodic pattern). It is denoted as ARIMA(p,d,q)x(P,D,Q)$_s$ and is expressed using Equation (4.5) (Arandia *et al.*, 2015):

$$\Phi_P(B^s)\varnothing(B)(1-B^s)^D(1-B)^d Y_t = \delta + \Theta_Q(B^s)\theta(B)\epsilon_t \tag{4.5}$$

$$\Phi_P(B^s) = 1 - \Phi_1 B^s - \Phi_2 B^{2s} - \ldots - \Phi_P B^{Ps} \tag{4.6}$$

$$\Theta_Q(B^s) = 1 + \Theta_1 B^s + \Theta_2 B^{2s} + \ldots + \Theta_Q B^{Qs} \tag{4.7}$$

$$\varnothing(B) = 1 - \varnothing_1 B - \varnothing_2 B^2 - \ldots - \varnothing_p B^p \tag{4.8}$$

$$\theta(B) = 1 + \theta_1 B + \theta_2 B^2 + \ldots + \theta_q B^q \tag{4.9}$$

$$\delta = \mu(1 - \varnothing_1 - \ldots - \varnothing_p)(1 - \Phi_1 - \ldots - \Phi_P) \tag{4.10}$$

$$B^k Y_t = Y_{t-k} \tag{4.11}$$

where Equations (4.6)–(4.11) give the seasonal autoregressive polynomial, seasonal moving average polynomial, ordinary (non-seasonal) autoregressive (AR) polynomial, and the ordinary (non-seasonal) moving average (MA) polynomial respectively; B is the backshift operator as defined in Equation (4.11); P is the seasonal AR polynomial order, Q is the seasonal MA polynomial order, p is the non-seasonal AR polynomial order, q is the non-seasonal MA polynomial order, D is the seasonal differencing order, d is the non-seasonal differencing order, s is the seasonal period, Y_t is the water demand time series, μ = mean value of the process; ϵ_t = forecast error at time t, in which ϵ_t is assumed to follow a normal (0, σ) distribution, and σ = standard deviation of the process.

4.2.3 Creating ARIMA/SARIMA models using econometric toolbox

This example shows how to use MATLAB's Econometric Modeler App to create ARIMA and SARIMA models for time series analysis using the following 36-months hypothetical water demand data, with each time step corresponding to one month:

[266.0, 145.9, 183.1, 119.3, 180.3, 168.5, 231.8, 224.5, 192.8, 122.9, 336.5, 185.9, 194.3, 149.5, 210.1, 273.3, 191.4, 287.0, 226.0, 303.6, 289.9, 421.6, 264.5, 342.3, 339.7, 440.4, 315.9, 439.3, 401.3, 437.4, 575.5, 407.6, 682.0, 475.3, 581.3, 646.9]

You can download the Econometrics toolbox in MATLAB by clicking on Apps → Get More Apps → and then search for 'Econometrics Toolbox' in the Add-On Explorer Search bar. You can run the example by using the following procedures:

Step 1. Save the water demand data as an excel file with each data value in a row so that you have one column of data (you can write the 'water demand' header in column A and row 1 and the data values in column A from rows 2 to 37. Import it to MATLAB's workspace by clicking on Home → Import Data.

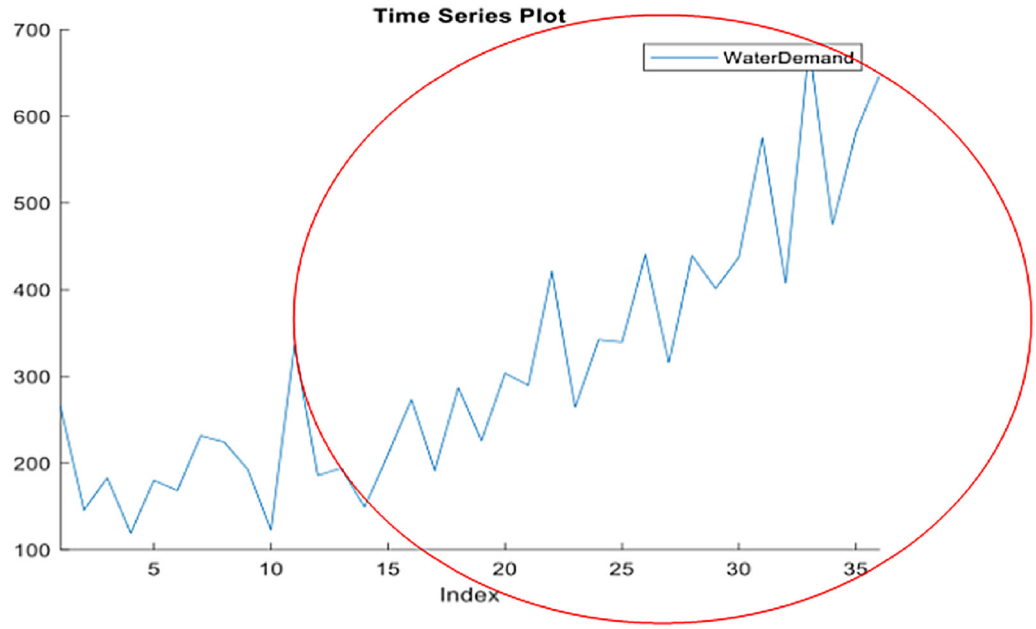

Figure 4.1 Time series plot of water demand.

Step 2. Open the Econometric Modeler app and click on Import → Import from Workspace to import and load the water demand time series data.

Step 3. The time series is plotted automatically and is shown in Figure 4.1. From the time series plot, the presence of a linear trend and seasonality (cyclic pattern) is evident, which means that the time series is non-stationary. Box–Jenkins models can only be applied to stationary time series, therefore, the nonstationary time series needs to be differenced to make it stationary.

Step 4. Click on the time series tab in the data browser (see Figure 4.2) and click on the time series variable that was just loaded. You can right-click to rename the variable 'Water Demand.'

Step 5. Click on 'ACF' and 'PACF' in the plots tab (see Figure 4.3) to plot the autocorrelation function (ACF) and partial autocorrelation function (PACF) of the time series as shown in Figures 4.4 and 4.5 respectively. ACF, which 'gives the correlation of time-series data with its previous time-series data,' and PACF, which 'correlates the time-series with its own lagged values separated by certain time units,' are analytical tools that are used to assess the 'reliability of time-series analysis' (ArunKumar *et al.*, 2021).

The presence of a trend can also be noticed by looking at the ACF plot, which is indicated by continuing large autocorrelations even after several lags (NCSS). The first five lags in the ACF plot shown in Figure 4.4 are significant, which indicates the presence of a trend.

Step 6. Click on 'difference' in the econometric modeler tab to perform a first order non-seasonal difference operation ($d=1$) to remove the trend. A new differenced time series shown in Figure 4.6 was created with 'Diff' automatically added next to the variable name, for example WaterDemandDiff. It is clear that there is no trend present anymore, however, if trend was still present, a second order difference operation ($d=2$) would have been applied by clicking on 'WaterDemandDiff' and clicking on 'difference' to get a new time series with the variable name 'WaterDemandDiffDiff' – the two 'Diff' words after the name of the variable means that the time series was differenced twice ($d=2$).

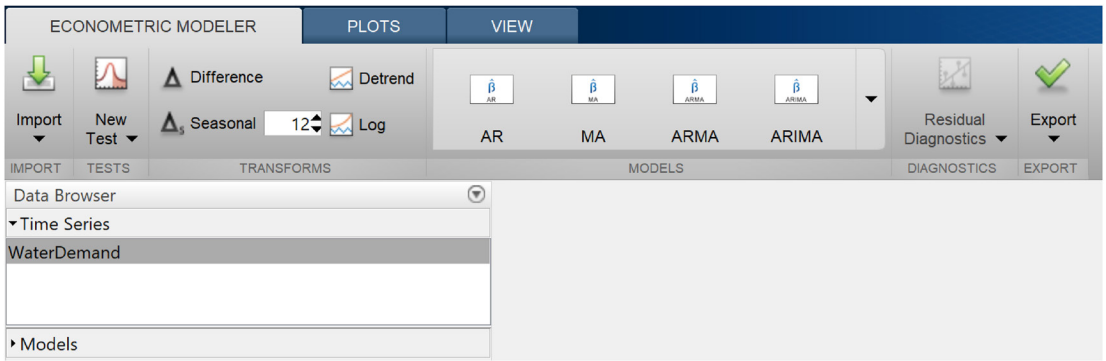

Figure 4.2 Data browser.

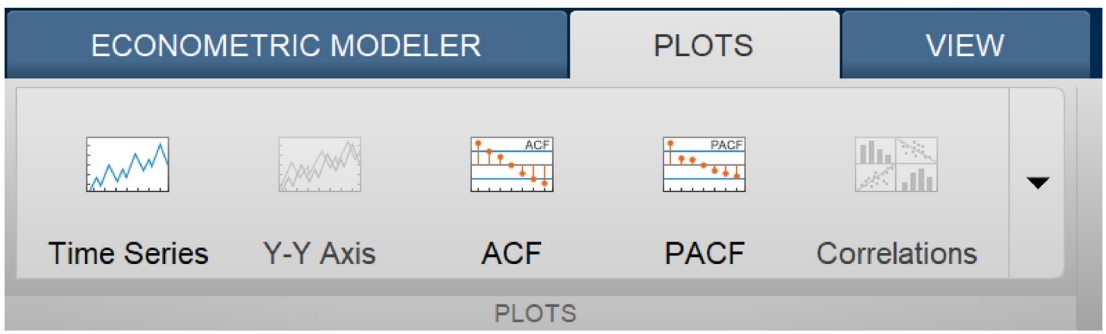

Figure 4.3 Plots tab.

Step 7. Click on 'WaterDemandDiff' in the time series tab, and then click on 'ACF' and 'PACF' to plot the autocorrelation function and partial autocorrelation function respectively of the first order differenced time series, which are shown in Figures 4.7 and 4.8. From the ACF plot, the autocorrelations attenuate quickly, which means that there is no more trend, and a suitable value of d has been attained ($d=1$) (Kofinas *et al.*, 2014). We will refer back to the ACF and PACF plots of 'WaterDemandDiff' in Step 9.

Step 8. The value of p and q are found from the PACF and ACF respectively of the appropriately differenced time series (Kofinas *et al.*, 2014). We have an AR model if the partial autocorrelations of the appropriately differenced time series cut off after a small number of lags, where the value of p is the last lag with a large value, and we have an MA model if the autocorrelations of the appropriately differenced time series cut off after a small number of lags, where the value of q is the last lag with a large value (NCSS). However, if the partial autocorrelation or autocorrelation plots of the appropriately differenced time series do not cut off, that means that we either have a mixed ARIMA model with p and q values greater than zero, or that we have an AR model with $p=0$ when only the partial autocorrelation plot does not cut off, or that we have a MA model with $q=0$ when only the autocorrelation plot does not cut off. If both partial autocorrelation and autocorrelation plots of the appropriately differenced time series do not cut off, we have a mixed ARIMA model with positive p and q values that can be estimated by using trial and error until the autocorrelations are minimal (NCSS).

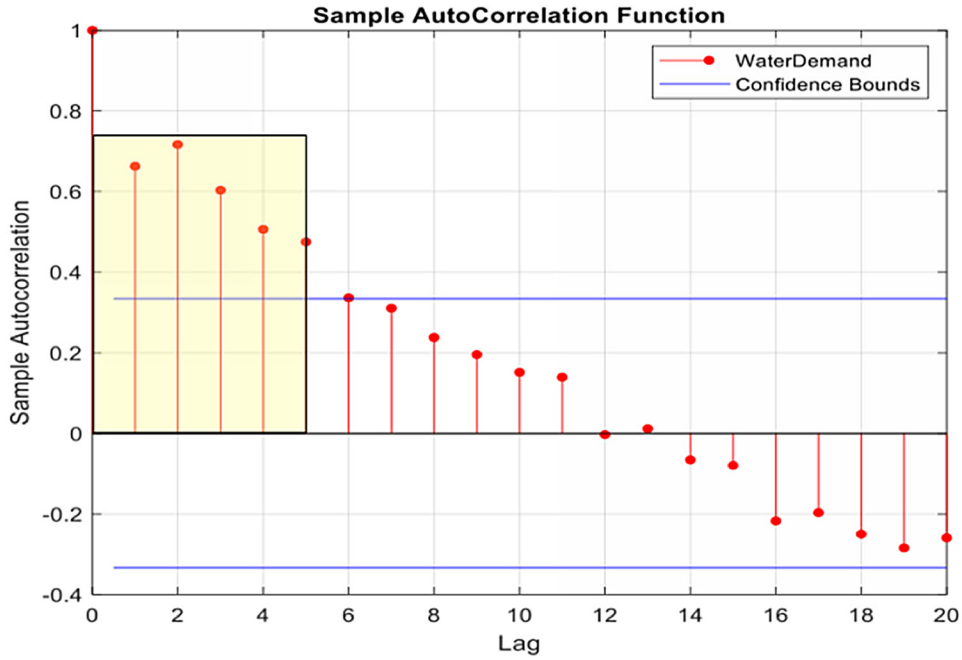

Figure 4.4 Sample autocorrelation function of WaterDemand.

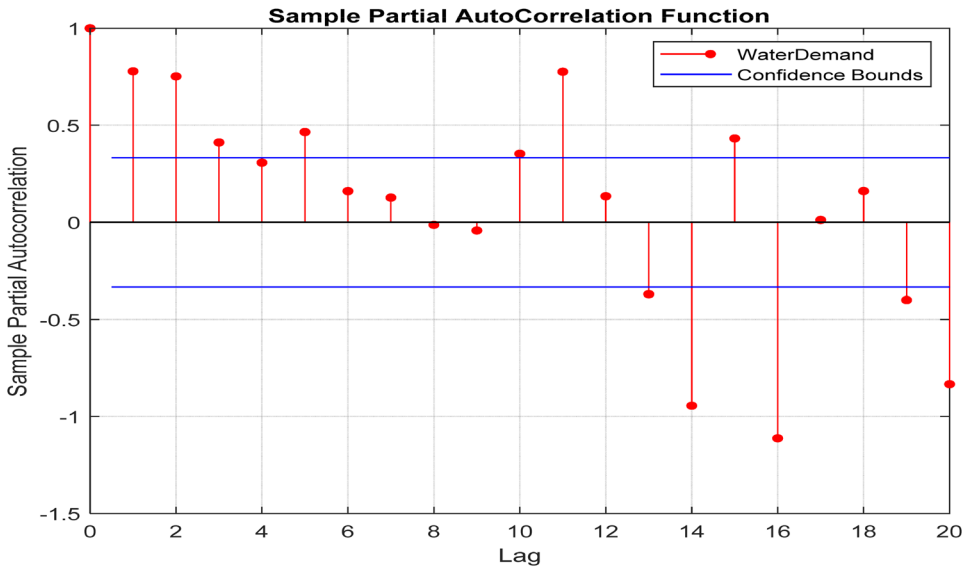

Figure 4.5 Sample partial autocorrelation function of WaterDemand.

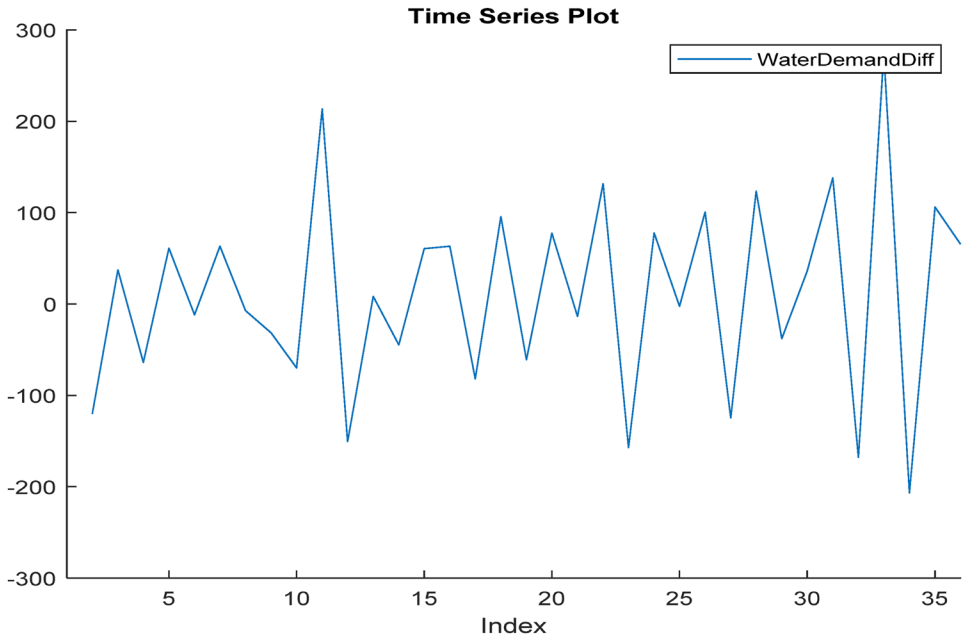

Figure 4.6 Time series plot of WaterDemandDiff.

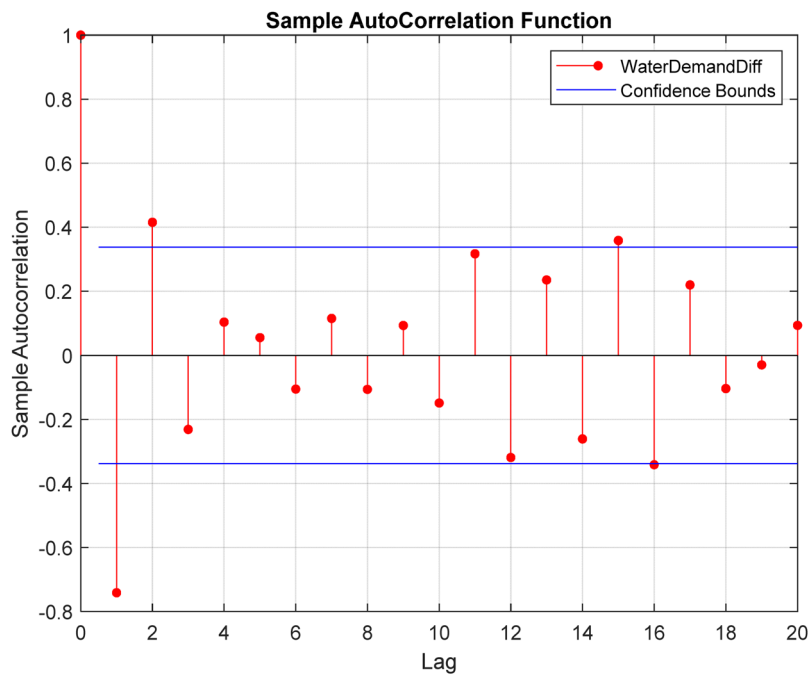

Figure 4.7 Sample autocorrelation function of WaterDemandDiff.

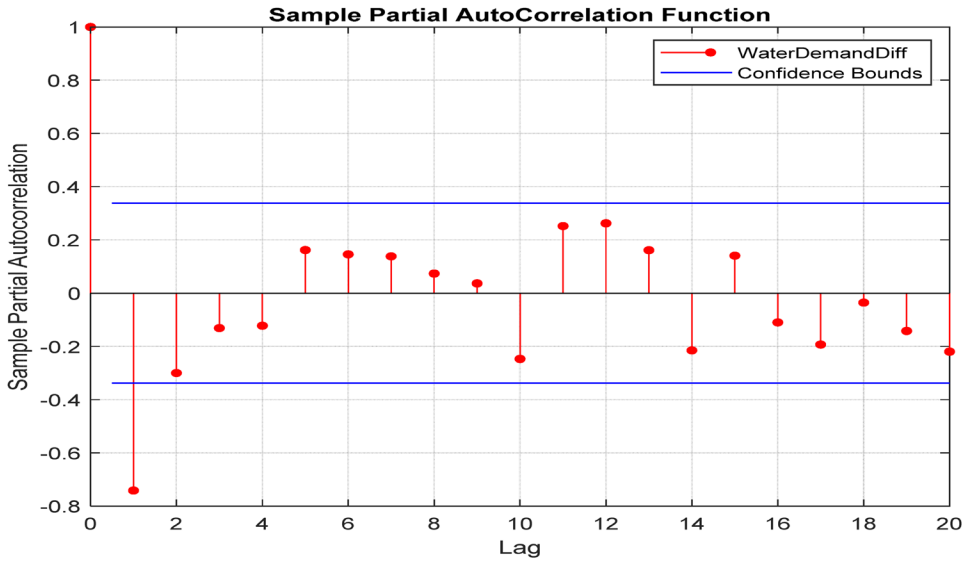

Figure 4.8 Sample partial autocorrelation function of WaterDemandDiff.

Step 9. By looking at the ACF plot of 'WaterDemandDiff' in Figure 4.7, the autocorrelation cuts off shortly after lag 2, therefore q can be chosen as 2. Similarly, by looking at the PACF plot of 'WaterDemandDiff' in Figure 4.8, the partial autocorrelation cuts off shortly after lag 1, therefore p can be chosen as 1. Therefore, we could fit the water demand time series data to an ARIMA (11,2) model where $p=1$, $d=1$, and $q=2$ and then check if the model is a good fit.

Step 10. Click on 'WaterDemandDiff' in the time series tab and then click on the econometric modeler tab. Click on ARIMA and enter the degree of integration or d as 1, autoregressive order or p as 1, moving average order or q as 2, and then click on 'Estimate' to create the ARIMA model as shown in Figure 4.9. The created model is put under the models tab and has the variable name 'ARIMA_WaterDemandDiff.' A model summary as shown in Figure 4.10 is automatically created and it features the model fit plot to compare the differenced time series and the ARIMA model, the estimated ARIMA model parameters and their associated standard errors and p-values, the residual plot, and the goodness of fit using Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC) to assess the model reliability. The p-values for the constant, AR and MA parameters are used to determine whether the terms in the model are statistically significant by comparing them to the level of significance, α, which is usually taken as 0.05 – a parameter is considered statistically significant if its p-value is less than or equal to $\alpha=0.05$. AIC and BIC are analytical tools that are used to assess the quality or reliability of time-series models by determining 'how well a model explains the relationships between the variables' – the lower AIC and BIC values are, the more a model is 'likely to be considered as a true model' (ArunKumar *et al.*, 2021).

Step 11. As mentioned earlier, the water demand time series had both trend and seasonality, and the trend was removed after it was differenced with $d=1$ to get 'WaterDemandDiff.' Now, the seasonality will be removed, and the time series will be fitted to a SARIMA model. Click on 'WaterDemandDiff' in the time series tab and enter '12' next to 'Seasonal' since the water demand data is monthly, and then click on 'Seasonal' to perform a seasonal difference ($D=1$) to remove the seasonality (see Figure 4.11).

A new seasonal differenced time series with the name 'WaterDemandDiffSeasonalDiff' shown in Figure 4.12 was created with 'SeasonalDiff' automatically added to the name 'WaterDemandDiff.'

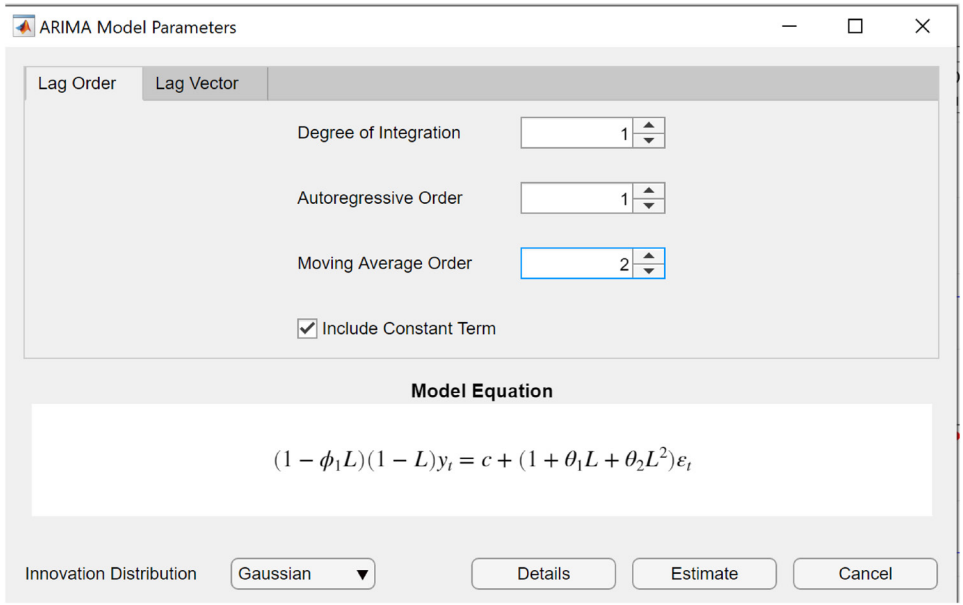

Figure 4.9 ARIMA model parameters.

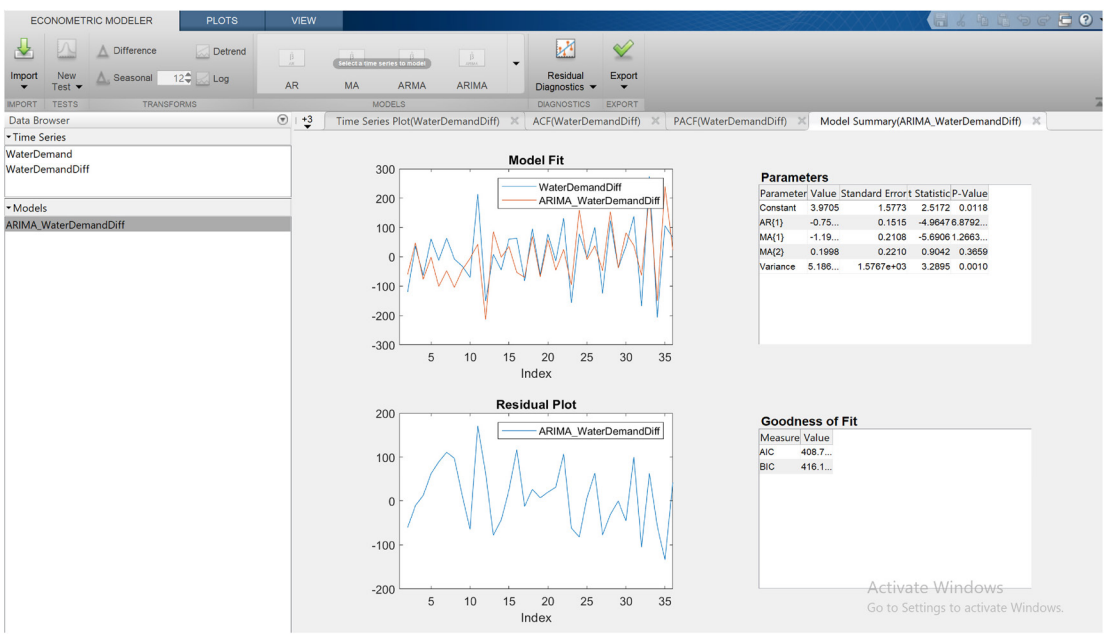

Figure 4.10 Summary results for ARIMA_WaterDemandDiff.

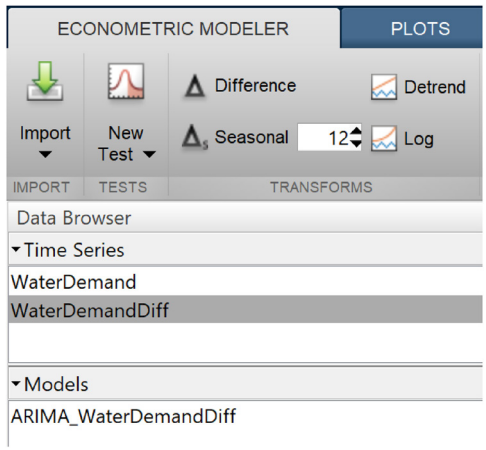

Figure 4.11 Performing seasonal difference.

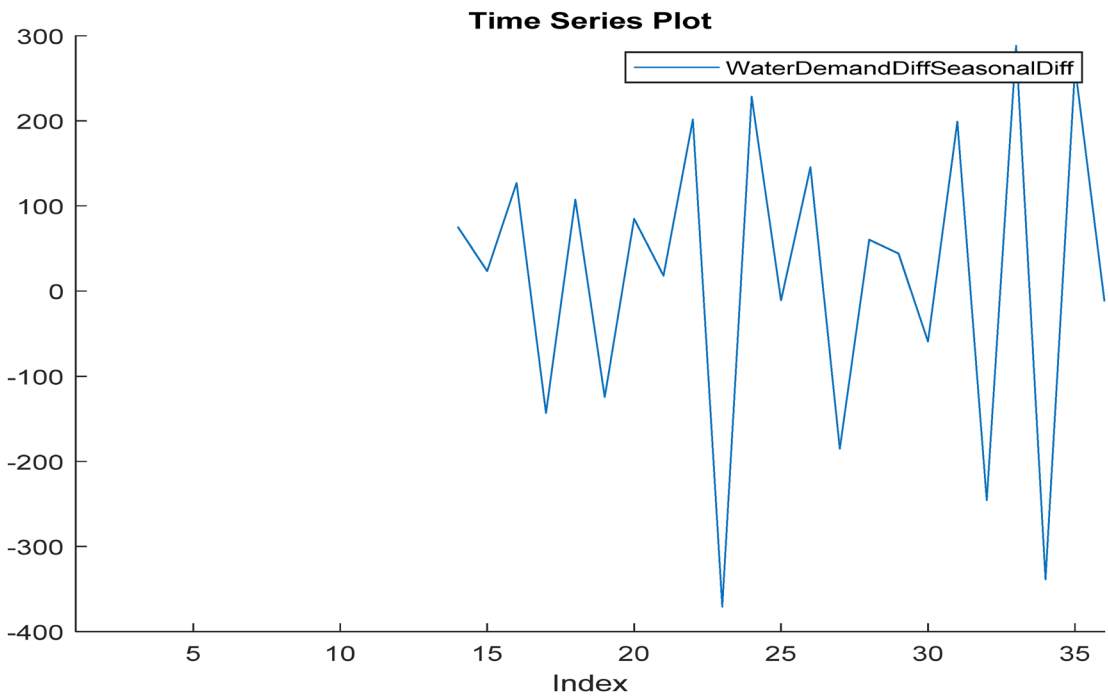

Figure 4.12 Time series plot of WaterDemandDiffSeasonalDiff.

Step 12. We now have most of the terms for the seasonal ARIMA model or ARIMA$(p,d,q) \times$ (P,D,Q) s. The non-seasonal (p,d,q) terms of the model were found previously ($p=1$, $d=1$, and $q=2$), $s=12$, $D=1$, and we can try $P=0$ and $Q=1$. Therefore, we could fit the water demand time series data to an ARIMA $(11,2) \times (01,1)12$ model and then check if the model is a good fit.

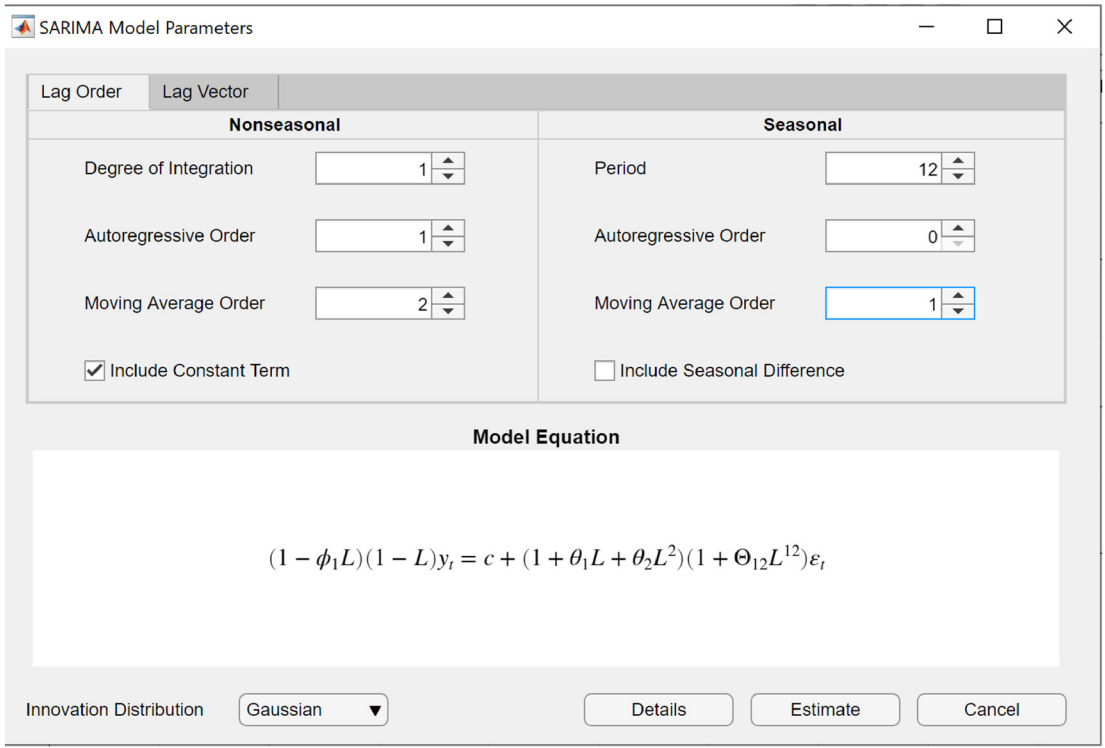

Figure 4.13 SARIMA model parameters.

Step 13. Click on 'WaterDemandDiffSeasonalDiff' in the time series tab and then click on the econometric modeler tab. Click on the arrow next to ARIMA to show all of the available models and then click on SARIMA and enter the non-seasonal degree of integration or d as 1, non-seasonal degree autoregressive order or p as 1, non-seasonal degree moving average order or q as 2, seasonal period or s as 12, seasonal degree autoregressive order or P as 0, seasonal degree moving average order or Q as 1, and then click on 'Estimate' to create the SARIMA model as shown in Figure 4.13. Normally, you should click on the checkbox next to 'Include Seasonal Difference' to include the seasonal difference term, however, checking that box for this example causes an error since the water demand data size is small – we will include the seasonal difference term manually when we do the forecast in the next section.

Step 14. The created model is put under the models tab and has the variable name 'SARIMA_WaterDemandDiffSeasonalDiff.' The automatically created model summary is shown in Figure 4.14. The AIC and BIC of the ARIMA $(11,2)\times(01,1)12$ model are 286.9 and 293.1 respectively, which are about half of the values for the ARIMA (11,2) model, which has an AIC of 408.7 and BIC of 416.2. Therefore, the SARIMA model has a better fit than the ARIMA model for this monthly water demand data, which makes it more reliable.

Step 15. Click on the econometrics modeler tab and then click on 'ARIMA_WaterDemandDiff' in the model tab followed by 'Export' → 'General Function' to generate a MATLAB code for creating the selected ARIMA model. A new MATLAB file with the model code will be automatically opened. Go back to the Econometric Modeler app and do same for the SARIMA model: click on 'SARIMA_WaterDemandDiffSeasonalDiff' in the model tab followed by 'Export' → 'General

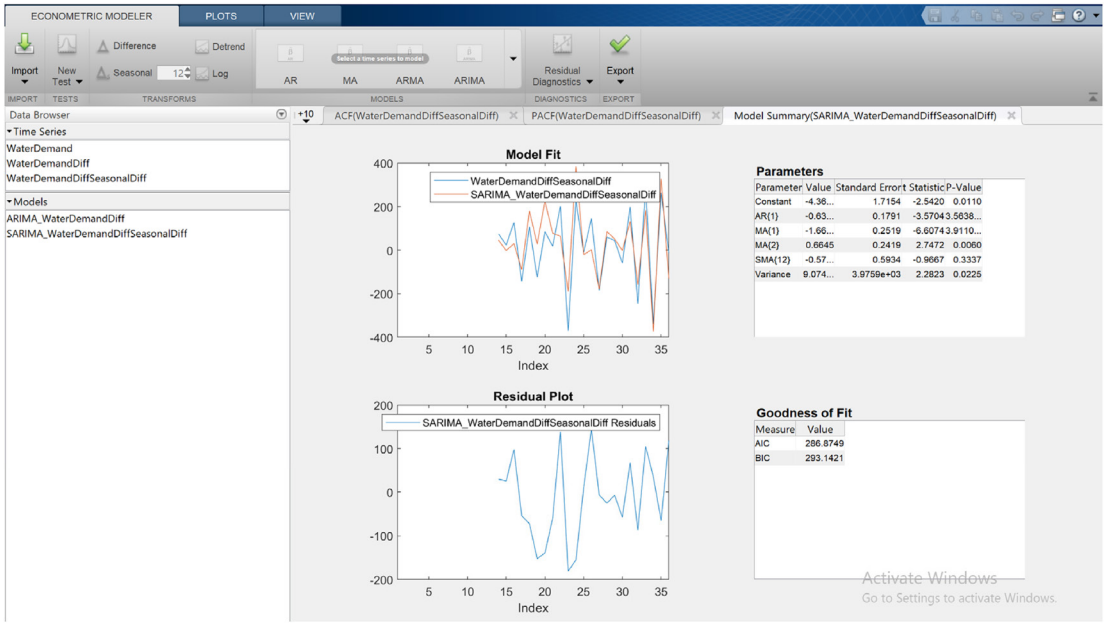

Figure 4.14 Summary results for SARIMA_WaterDemandDiffSeasonalDiff.

Function' to generate a MATLAB code for creating the selected SARIMA model. Save the two MATLAB files since we will use them in the forecasting section.

Step 16. Click on 'Export' → 'Generate Report' to generate a report summarizing the results of what we did using the econometrics modeler app. The report can be either in pdf, docx, or html format, and you can click on the check box next to the name of the time series or models that that you would like to include in the report (see Figure 4.15).

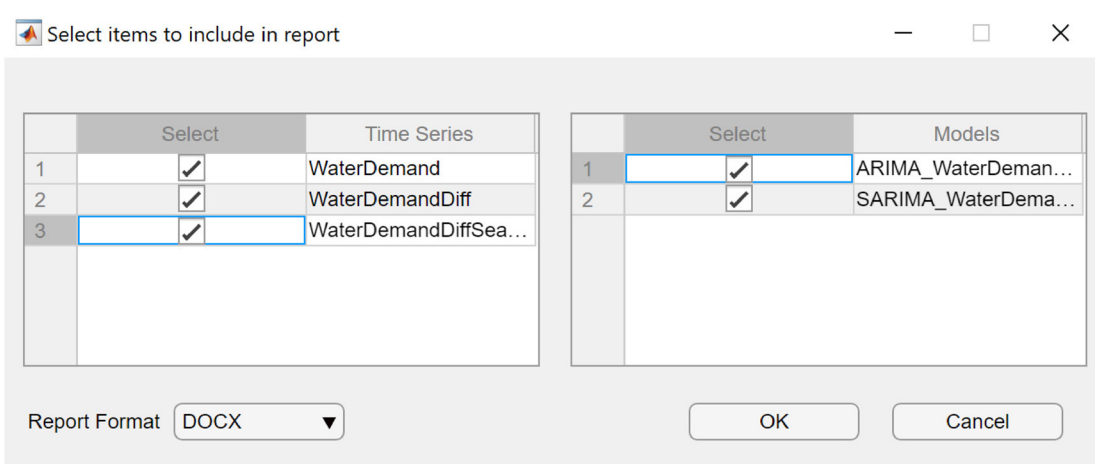

Figure 4.15 Generating a report.

4.2.4 Forecasting

MATLAB's forecast function uses an observed time series as a presample data (to initialize the forecasts) and a fitted regression model such as an ARIMA or SARIMA model to generate minimum mean square error (MMSE) forecasts denoted in Equation (4.12):

$$\hat{y}_{t+1} = E(y_{t+1}|H_t, X_{t+1}) \tag{4.12}$$

where H_t is the history of the process up to time t *and* X_{t+1} is the exogenous covariate series up to time $t+1$ (Mathworks, 2021a).

Equation (4.13) shows an s-step ahead forecast mean square error (MSE) corresponding to the MMSE forecasts (Mathworks, 2021b):

$$MSE = E(y_{t+s} - \hat{y}_{t+s} \mid H_{t+s-1}, X_{t+s})^2 \tag{4.13}$$

The performance of ARIMA and SARIMA models can be evaluated using either the MSE or the root mean squared errors (RMSE) given in Equation (4.14):

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(Yt - Yo)^2}{n}} \tag{4.14}$$

where Y_t is the forecasted observation, Y_o is the actual observation, and n is the number of observations. The ARIMA and SARIMA models obtained in the example were used respectively to make a 12-months future forecast using the following procedures given in the two MATLAB codes:

ARIMA FORECAST MATLAB CODE:

```
% Forecast ARIMA Model
% This example shows how to forecast an ARIMA (11,2) model for a
% hypothetical water demand data using MATLAB's forecast function.

% Step 1: Load the water demand data and prepare it for analysis.

[~, ~, data] = xlsread('C:\Users\User\Documents\ waterdemand.xlsx'); % change this to your file location
data = data(:,1); % corresponds to the 1st column in the excel file (column A)
data = data(2:37); % corresponds to the data range from row 2 to 37 in the excel file
data = [data{:}];
data = data';
y = data;
T = length(y);

% Step 2: Estimate an ARIMA (11,2) model for the water demand time series
% data.

Mdl = arima('Constant',NaN,'ARLags',1,'D',1,'MALags',1:2,'Distribution','Gaussian'); % the ARIMA model
function on the right hand side of the equal to sign was copied directly from the model estimate equation
given in the saved MATLAB function that was generated from the Econometric Modeler.

EstMdl = estimate(Mdl,y,'Display','off');

% Step 3: Forecast future water demand for the next 12 months using
% the fitted ARIMA model and the observed water demand time series as
% presample data to generate MMSE forecasts and their corresponding MSE and RMSE
```

```
[yF,yMSE] = forecast(EstMdl,12,'Y0',y);
upper = yF + 1.96*sqrt(yMSE);
lower = yF - 1.96*sqrt(yMSE);

mse = mean((lower-yF).^2) % calculate the MSE
rmse = sqrt(mse) % calculate the RMSE

figure
plot(y,'Color',[.75,.75,.75])
hold on
h1 = plot(T + 1:T + 12,yF,'r','LineWidth',2);
h2 = plot(T + 1:T + 12,upper,'k--','LineWidth',1.5);
plot(T + 1:T + 12,lower,'k--','LineWidth',1.5)
xlim([0,T + 12])
title({'Forecast and 95% Forecast Interval using ARIMA (11,2)', 'RMSE = ' + rmse})

legend([h1,h2],'Forecast','95% Interval','Location','NorthWest')
xlabel('Month')
ylabel('Water Demand')
hold off
```

The results of the ARIMA forecast are shown in Figure 4.16.

SARIMA FORECAST MATLAB CODE:

```
% Forecast SARIMA Model
% This example shows how to forecast a seasonal ARIMA (11,2) × (01,1)12 model
% for a hypothetical water demand data using MATLAB's forecast function.
```

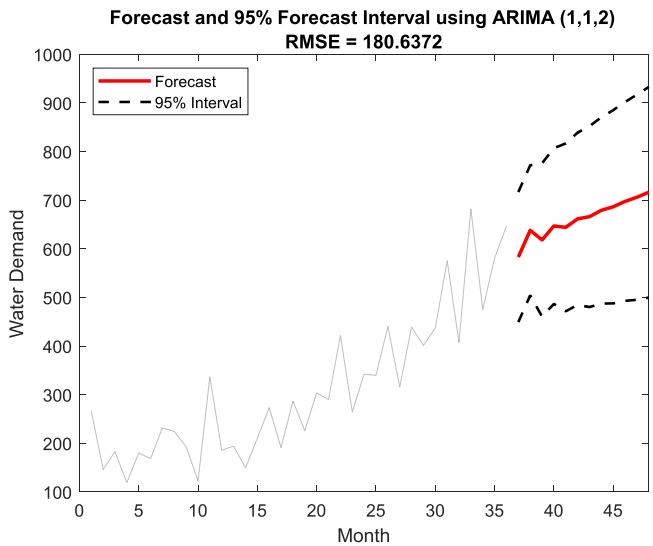

Figure 4.16 Forecast and 95% forecast interval using ARIMA (1,1,2).

```
% Step 1: Load the water demand data and prepare it for analysis.

[~, ~, data] = xlsread('C:\Users\User\Documents\NYC 311 Water Complaints\waterdemand.xlsx'); %
change this to your file location

data = data(:,1); % corresponds to the 1st column in the excel file (column A)
data = data(2:37); % corresponds to the data range from row 2 to 37 in the excel file
data = [data{:}];
data = data';
y = data;
T = length(y);
% Step 2: Estimate an ARIMA (11,2) × (01,1)12 model for the water demand time series data.

Mdl = arima('Constant',NaN,'ARLags',1,'D',1,'MALags',1:2,'SARLags',[],'Seasonality',12,'SMALags',12,
'Distribution','Gaussian'); % the seasonal ARIMA model function on the right hand side of the equal to
sign was copied directly from the model estimate equation given in the saved MATLAB function that was
generated from the Econometric Modeler. However, the seasonality term was changed from '0' to '12'
to include the seasonal difference, which was not included in the estimation as discussed Step 15 in the
previous section.
EstMdl = estimate(Mdl,y,'Display','off');
% Step 3: Forecast future water demand for the next 12 months using
% the fitted ARIMA model and the observed water demand time series as
% presample data to generate MMSE forecasts and their corresponding MSE and RMSE.

[yF,yMSE] = forecast(EstMdl,12,'Y0',y);
upper = yF + 1.96*sqrt(yMSE);
lower = yF - 1.96*sqrt(yMSE);

mse = mean((lower-yF).^2) % calculate the MSE
rmse = sqrt(mse) % calculate the RMSE

figure
plot(y,'Color',[.75,.75,.75])
hold on
h1 = plot(T + 1:T + 12,yF,'r','LineWidth',2);
h2 = plot(T + 1:T + 12,upper,'k--','LineWidth',1.5);
plot(T + 1:T + 12,lower,'k--','LineWidth',1.5)
xlim([0,T + 12])
title({'Forecast and 95% Forecast Interval using ARIMA (11,2) × (01,1)12', 'RMSE = ' + rmse})

legend([h1,h2],'Forecast','95% Interval','Location','NorthWest')
xlabel('Month')
ylabel('Water Demand')
hold off
```

The results of the SARIMA forecast are shown in Figure 4.17.

4.2.5 Limitations

Although ARIMA and SARIMA can be used to model a wide range of time series problems, one of the major limitations of these models is their inability to capture nonlinear patterns due to their linear structure (Kofinas *et al.*, 2014). Machine learning-based time series models such as artificial neural networks (ANNs) can capture both linear and non-linear patterns, therefore hybrid ARIMA and ANN models have been proposed to tackle the nonlinearity deficiencies (Kofinas *et al.*, 2014). Faruk (2010)

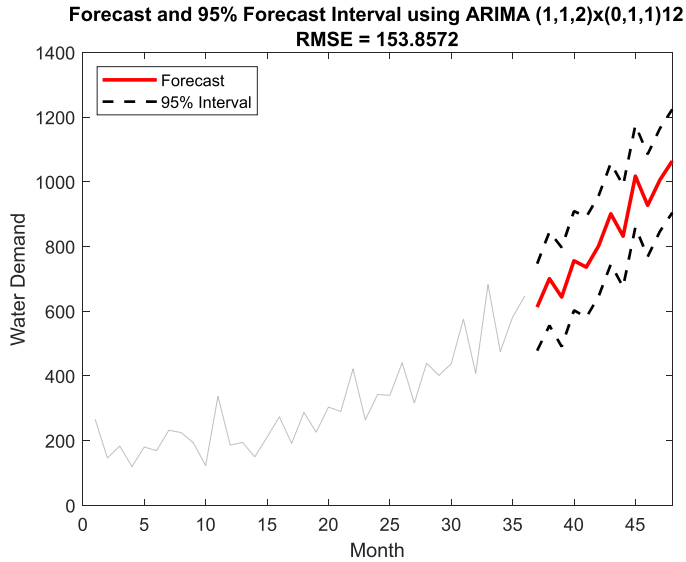

Figure 4.17 Forecast and 95% Forecast Interval using ARIMA (1,1,2)x(0,1,1)12.

used a hybrid neural network and ARIMA model for water quality time series prediction by using water quality data such as water temperature, and boron and dissolved oxygen concentrations collected at the Buyuk Menderes river in Turley from 1996 to 2004. The hybrid model provided accurate results by tackling both the linear and nonlinear patterns of the complex water quality time series (Faruk, 2010).

4.3 MACHINE LEARNING TIME SERIES

4.3.1 Machine learning

4.3.1.1 Artificial neural network

Artificial neural networks (ANNs) mimic the biological neural structure of the brain and form interconnected groups of artificial neurons which are organized in layers. It is a supervised machine learning technique that can be used to forecast water demand patterns over time. ANNs consist of three layers: input layer, hidden layer, and output layer. The inputs or predictors are inserted into the input layer as the bottom layer. The hidden layer is an intermediate layer with hidden neurons. The output layer forms the top layer as forecasts. Among the various architecture of ANN, the feedforward, back propagation (BP) neural network is the most popular, effective model to recognize patterns. A multilayer feedforward network is shown in Figure 4.18. There are four inputs, one hidden layer with three hidden neurons. Each layer of nodes receives inputs from previous layers.

Suppose the input of an ANN is $\boldsymbol{x}=[x_1, x_2, \ldots, x_n]'$ and its output is $\boldsymbol{y}(\boldsymbol{x})=[y_1, y_2, \ldots, y_n]'$. There exists a mapping M from the input space X:$\{x \in X|\ x$ is the input to the system$\}$ to output space Y:$\{y \epsilon Y\ |y$ is the output of the system for given input $x\}$. So, the mapping M is as follows:

$$M : X \rightarrow Y \tag{4.15}$$

The training process can be considered a process of gradually adjusting the network internal parameters, for example, the weight w in the weight space ω, that is $w \in \omega$, so that the error between the expected outputs $\hat{y}(x,\ w)$ and the real outputs $y(x)$of the network are minimal:

$$error = \min \left\| \hat{y}(x,w) - y(x) \right\|^2 \tag{4.16}$$

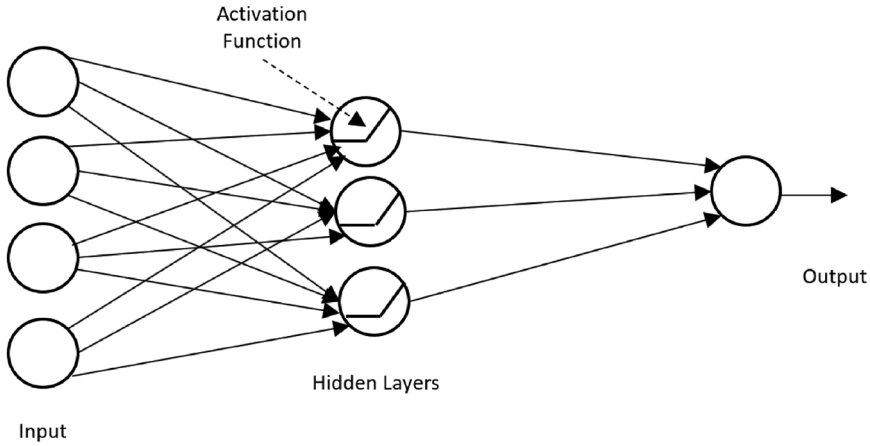

Figure 4.18 Artificial neural networks (ANNs).

The activation of the artificial neuron is conducted through the following equations:

$$\phi(\mathrm{z}) = \phi\left(\sum_i w_i x_i + b\right) \tag{4.17}$$

where i stands for the independent variables that we are considering. The activation function is a non-linear function. Three activation functions that we will consider are the sigmoid function (sigmoid), the hyperbolic tangent function (tanh) and the rectified linear function (ReLU) shown below:

$$sigmoid(z) = \frac{1}{1+e^{-z}} \tag{4.18}$$

$$\tanh(z) = \frac{e^{2z}-1}{e^{2z}+1} \tag{4.19}$$

$$\mathrm{ReLU}(\mathrm{z}) = \max(0, z) \tag{4.20}$$

The training process of feedforward backpropagation ANN is summarized as follows: (1) *Initialize*: construct the feedforward neural network by choosing the input units and output units; (2) *Feedforward*: the input value is propagated from the input layer via the hidden layer to the output layer using the weight and offset value of the network. Compute the output and the error until a stopping criterion is met; (3) *Backpropagation*: the weight is continuously updated and modified so that the error is minimized.

4.3.1.2 Support vector machine

SVM is a supervised machine learning algorithm (Candelieri, 2017; Msiza *et al.*, 2007; Sengupta *et al.*, 2018). The goal of SVM is to separate a given set of binary labeled training data with a hyperplane that is maximally distant from them, that is with maximized margin. However, a hyperplane cannot separate the training data if they are non-linearly separable. Hence, kernel function is introduced to map the training data from its original input space to a high dimensional space where a linear separation can be achieved. In this case, the hyper-plane found by the SVM in the feature space corresponding to a non-linear decision boundary in the original input space. Several common kernel functions are linear kernel, Gaussian radial basis kernel and Sigmoid kernel, and so on.

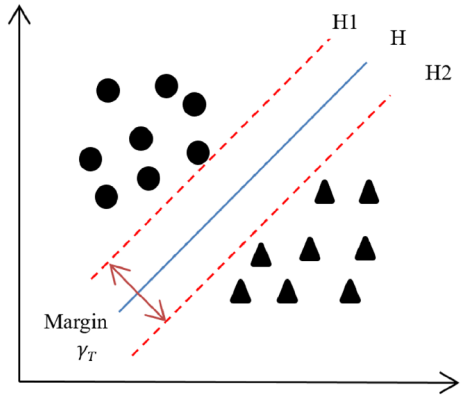

Figure 4.19 Support vector machines (SVMs).

As shown in Figure 4.19, the decision boundary of SVMs is a hyperplane H: (w, b), where w is a normal vector, or a weight vector, perpendicular to the hyperplane with initial value $w_0 = 0$. It is adjusted iteratively each time when training examples are misclassified by current w. b is intercept or bias. The hyperplane equation is defined as:

$$w^T x_i + b = 0 \tag{4.21}$$

To assign class labels to each class for test data, another two hyperplane H1 and H2 are used to determine their classification labels:

$$\begin{cases} H1 : w^T x_i + b \geq 1, & if\ y_i = +1 \\ H2 : w^T x_i + b \leq -1, & if\ y_i = -1 \end{cases} \tag{4.22}$$

Therefore, the final goal is to find the hyperplane with the largest margin. The points on H1 and H2 are called support vectors. Margin of the hyperplanes are the distance from support vectors to the hyperplane γ_T, namely the distance between H1 and H2. To solve the minimization problem, Lagrange multiplier method and Karush-Kuhn-Tucker (KTT) conditions are used to transform this problem to its dual problem. An equivalent dual problem of minimizing $\|w\|_2$ is a maximization problem solving by QP (Quadratic Programming) below:

$$\begin{aligned} &maximize\, W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j x_i \cdot x_j \\ &subject\, to \quad \sum_{i=1}^{m} y_i \alpha_i = 0 \\ &\qquad\qquad W = \sum_{i=1}^{m} \alpha_i y_j x_i \\ &0 \leq \alpha_i \leq C, i = 1, \ldots, m. \end{aligned} \tag{4.23}$$

where α_1, …, α_m is the Lagrangian multiplier associated with each training example (x_i, y_i). The Lagrangian multipliers are bounded by C, called a box constraint. α_i is the Lagrangian multipliers for support vectors.

The training process of SVM is summarized as follows: (1) Initialize: construct the SVM by entering input and output pairs of the training data sets. Compute the support vectors. (2) Sequential minimal optimization (SMO) is used to solve the QP problem. The goal of this problem to find the hyperplane with the largest margin.

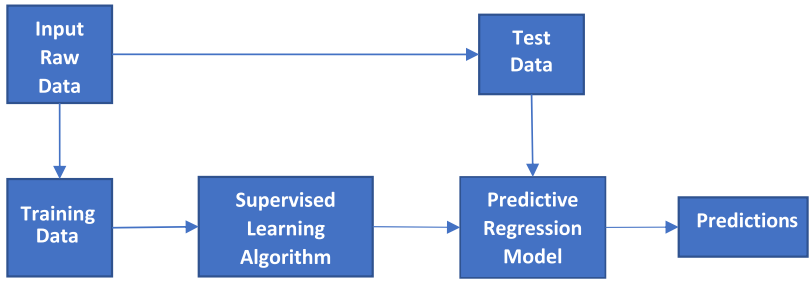

Figure 4.20 Machine learning for water demand forecasting.

4.3.1.3 Forecasting

The water demand forecasting problems can be formalized as supervised machine learning tasks. Supervised learning builds a predictive model that relies on the availability of a finite set of observations. These observations are the mapping or relation between a set of input variables and one or more output variables of the forecast problem.

The flow of a supervised machine learning forecasting task is presented in Figure 4.20. A raw dataset is divided into two subsets: a training set and test set. Data points in the training set are excluded from the test set. The training set is a collection of the input and output pairs. The training set is fed to a supervised learning algorithm to build a predictive regression model. Then, the test set validates the model using its output, that is predictions. In this case, the test set can also be referred to as the validation set. In some literatures, validation set is different from test set. Validation set is a third part of raw data which is used to tune the model's parameters to minimize the overfitting.

Water demand forecast can be solved using machine learning regression models. The input of the model is non-linear water demand time series. The output is real values depicting the water demand on a specific date. The regression problem will find a function $f(x)$ that can map the training inputs to the training outputs.

4.3.2 Practice problems

In this section, we present a simple forecasting problem using SVM regression. The data set we used is from hourly inflow/outflow data of production and storage facilities of the south-central water distribution network in Hillsborough County, FL, Apr 2012–Dec 2012 (Chen, 2018). The first 500 data points were selected for our example below for illustration purposes.

Step 1: Import the data. Separate the data as training and test set. Plot the training set as shown in Figure 4.21.

```
%Import the data from the data file 'Water demand data set 2_Unit_MLD.mat'. This file includes 500 data
points, where 450 data points (90% of the data) is chosen as training data set. The 50 data points are chosen
as the test data set. Plot the training datasets.

rawdata=importdata('Water demand data set 2_Unit_MLD.mat');
rawdata=rawdata';

data1=rawdata(1:450,:);
data1=data1'
figure
plot(data1)
xlabel('Hour')
ylabel('Million gallons')
title('System-wide water demands aggregated in 1-hour intervals in million gallons per day')
```

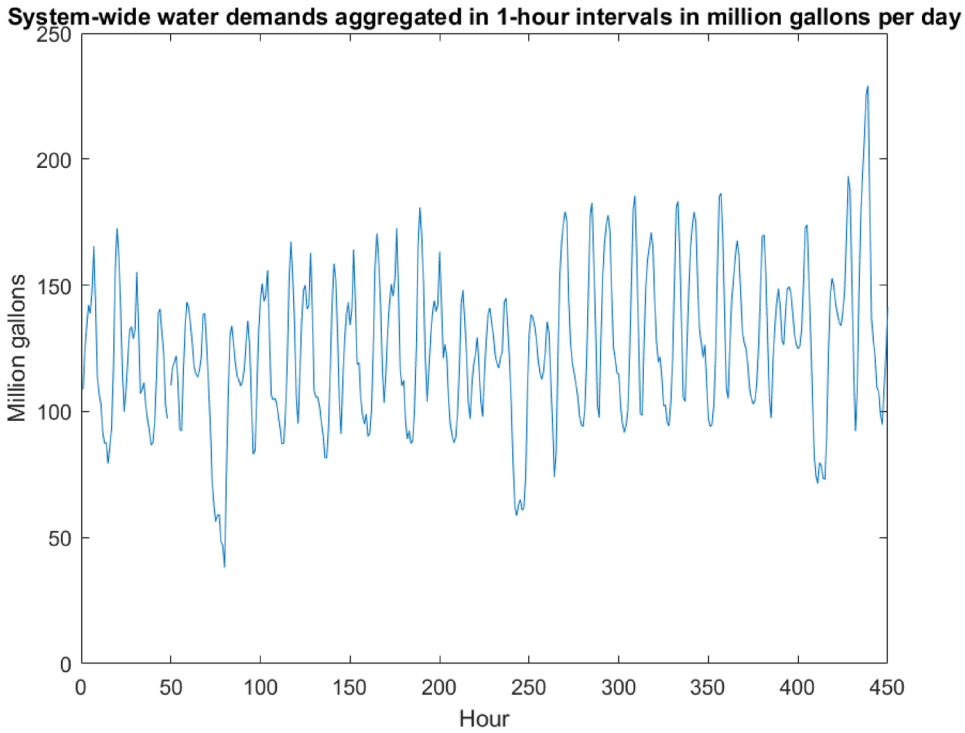

Figure 4.21 SVM Training data set.

Step 2: Construct training and testing data sets. Ninety per cent of the data (450 data points) is chosen as training data set. The remaining 10% of the data (50 data points) is chosen as the test data set.

```
data = rawdata(1:500,:);

numTimeStepsTrain = 450;

dataTrain = data(1:numTimeStepsTrain + 1);
dataTest = data(numTimeStepsTrain + 1:end);

numTimeStepsTest = numel(dataTest(1:end−1));

%XTrain is training data set
%YTrain is the response values of the training data set

XTrain = dataTrain(1:end−1);
YTrain = dataTrain(2:end);

YTest = dataTest(2:end);
```

Step 3: Configure and train the SVM.

```
%Use 'fitrsvm' function to train the SVM. List the kernel function as 'gaussian' kernel, and set the
'standardize' as true. The function will standardize the training data set.

svm_Mdl=fitrsvm(XTrain,YTrain, 'KernelFunction','gaussian','Standardize',true);
```

Step 4: Validate the trained SVM model. The forecasting results are showed in Figure 4.22 and compared with the observed results shown in Figure 4.23. The RMSE (root mean square error) values for SVM forecast model are shown in Figure 4.24.

```
%Use 'predict' function to validate the SVM predictive model svm_Mdl, with input test data set YTest.
YPred stores the forecast results.

YPred=predict(svm_Mdl,YTest);

%Plot the forecast results
figure
plot(dataTrain(1:end−1))
hold on
idx=numTimeStepsTrain:(numTimeStepsTrain+numTimeStepsTest);
plot(idx,[data(numTimeStepsTrain) YPred'],'.-')
hold off
xlabel('Hourly water demands')
ylabel('Million gallons')
title('Forecast 50 red data points in the future')
legend(['Observed' 'Forecast'])

%Plot the forecast results versus observed results

figure

plot(YTest)
hold on
plot(YPred,'.-')
hold off
legend(['Observed' 'Forecast'])
ylabel('Million gallons')
title('Forecast vs Observed')

% Quantitative evaluation of forecast results using RMSE

rmse=sqrt(mean((YPred-YTest).^2));
figure(),

stem(YPred – YTest);
xlabel('Hourly water demands')
ylabel('Error')
title('RMSE='+rmse)
```

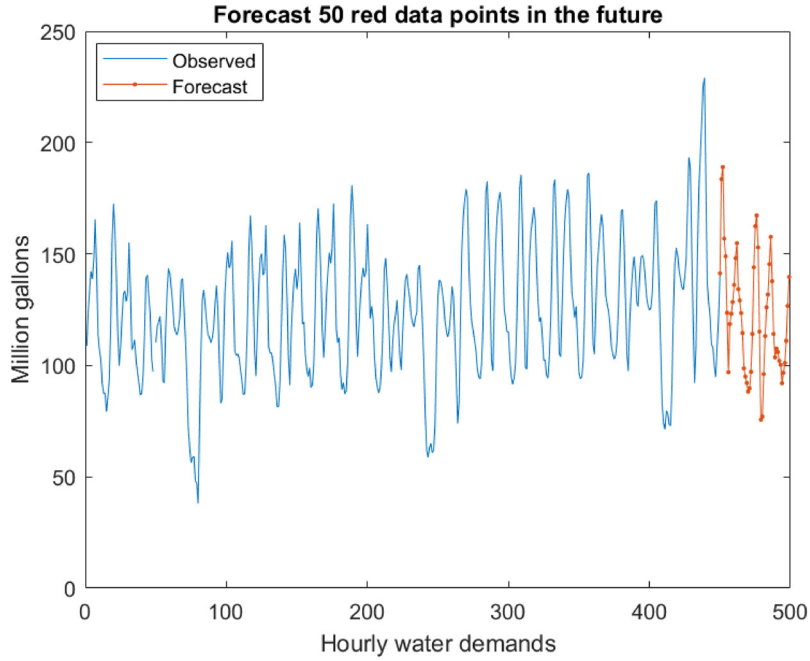

Figure 4.22 SVM forecast results.

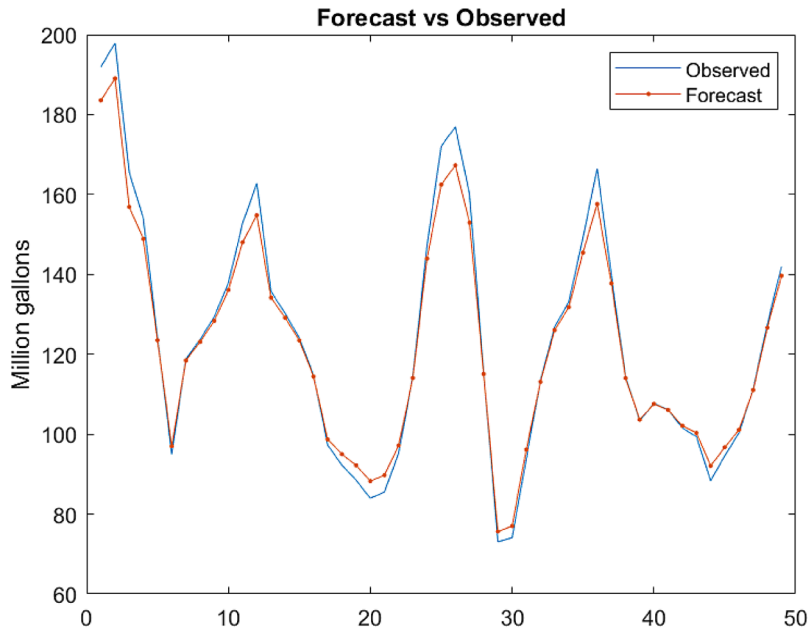

Figure 4.23 SVM forecast (testing) results compared with observed results.

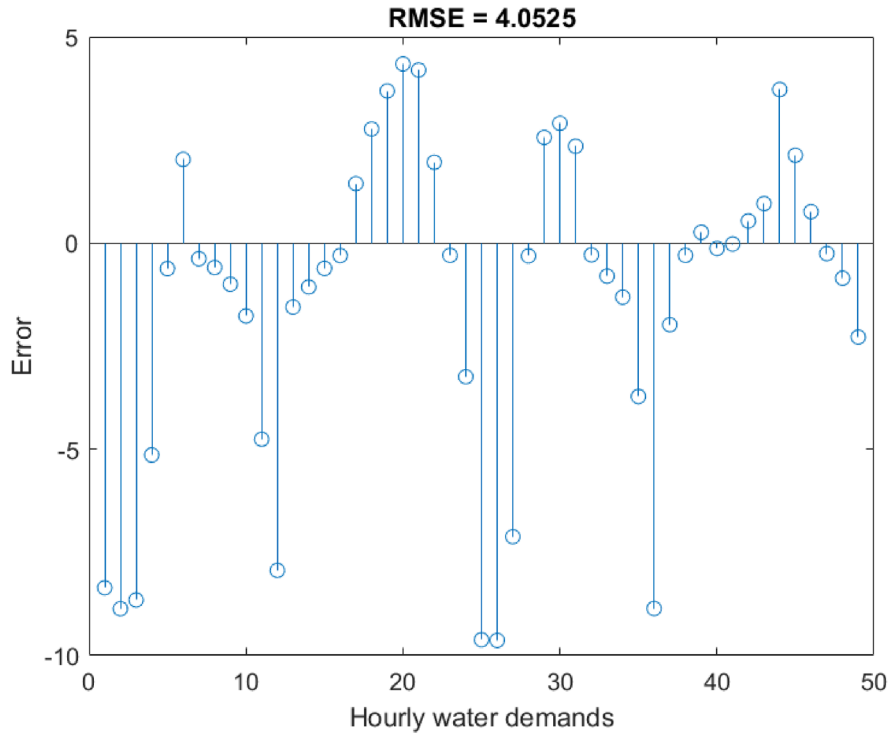

Figure 4.24 RMSE for SVM forecast model.

4.4 DEEP LEARNING TIME SERIES

Deep learning is a promising type of machine learning technique that has attracted much attention over the past few years. Deep learning has the advantages of processing big data, feature learning and strong generalization capability compared to shallow machine learning models. The deep learning time series model exhibits attractive performance in terms of accuracy, stability, and effectiveness (Bedi & Toshniwal, 2019; Du *et al.*, 2021; Guo *et al.*, 2018). We introduce two deep learning time series forecasting models in this section: Convolutional neural networks (CNN) and recurrent neural networks (RNN).

4.4.1 Deep learning models

4.4.1.1 Convolutional neural network

Convolutional neural network (CNN) is a neural network that has been successfully applied in image classification and feature mining. The main advantage of CNN is that it enables the most important features from the input to be extracted (Goldberg, 2016). CNN consists of three types of layers as building blocks: convolution layer, subsampling or pooling layer, as well as a fully connected layer as shown in Figure 4.25.

The convolution layer is a two-layer feed-forward neural network that includes a convolution operation that is designed to extract features from the input. CNN is designed to accept two-dimensional (2D) image data for feature extraction. Time series is one dimensional (1D) data in time domain, so a conversion from 1D to 2D data needs to be carried out before feeding into CNN for

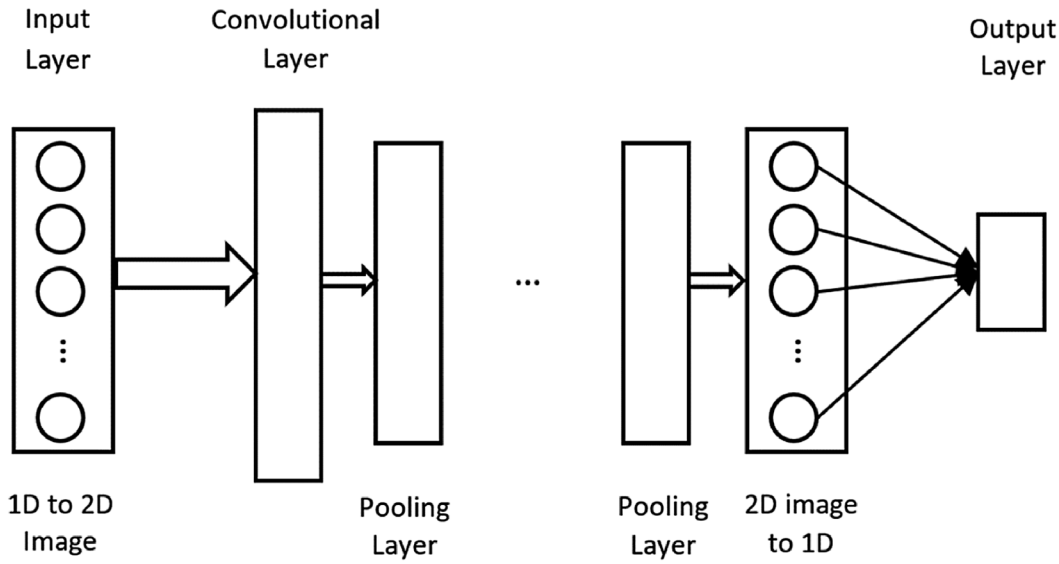

Figure 4.25 Convolutional neural networks (CNNs).

forecasting. Specifically, the input features x_i are convolved with shared weight w and bias term b and get the output y_j in the next layer as follows:

$$y_j = f(\sum x_i \otimes w_{i,j} + b_j) \tag{4.24}$$

where $\otimes$ is a convolutional operation and f is a sigmoid function.

The pooling layers are connected to convolutional layers to build up the high-level invariant structures in data. The pooling layer aims to reduce the dimensions of the data and create a down-sampled version of the input. The pooling operations include the max pooling and average pooling.

4.4.1.2 Recurrent neural network

Recurrent neural networks (RNNs) are designed to use the previous information in the sequence to produce the current input and gained popularity in time series forecasting with the recent advances of AI. Unlike ANN, it has forwarding connections in between the neuros and feedback loops. The main advantage of RNN is its acquisition of the internal sequential nature that remembers information through many timesteps, making it a powerful tool in forecasting long term trends from time series data. RNN is comprised of single rolled RNN units as shown in Figure 4.26.

Three kinds of RNN units are most popular for sequence modelling. They are the Elman RNN (ERNN) cell (Elman, 1990), the gated recurrent unit (GRU) cell (Cho *et al.*, 2014) and the long short-term memory (LSTM) cell (Hochreiter & Schmidhuber, 1997). The LSTM RNN network has been applied in time series prediction as a special kind of deep learning model.

The structure of RNN includes hidden state h, input X and an optional output Y. Given a time series input sequence $X = \{x_1, x_2, \ldots, x_t\}$, at time step t, RNN learns a mapping from x_t to h_t depending on the hidden state at h_{t-1}:

$$h_t = f(h_{t-1}, x_t), \tag{4.25}$$

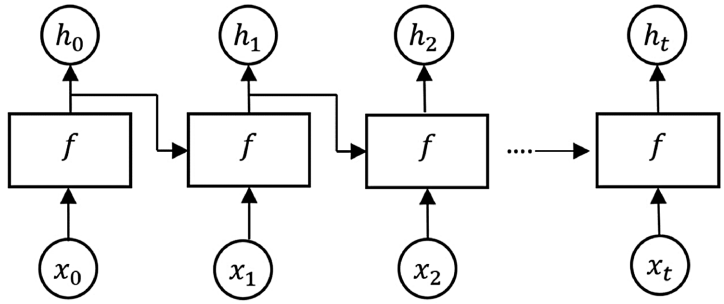

Figure 4.26 Recurrent neural networks (RNNs).

where f is a non-linear activation function. This function can be ERNN, GRN or LSTM, or as simple as a logic sigmoid function.

The training process of RNN suffers from problems of vanishing or exploding gradients which occur when backpropagating errors across many time steps. LSTM was introduced to overcome the above problem by replacing the hidden layer in the standard RNN by a memory cell. Each memory cell contains several gates and four interactive layers: forget gate layer, input gate layer, Tanh layer, and output gate layer.

4.4.2 Practice problems

In this section, we present a simple forecasting problem using LSTM regression. The data set we used is from hourly sewer flows monitored at Station S2 in Columbus, OH, Jun 1998–Dec 2013 (Chen, 2018). The first 500 data points was selected for our example below for illustration purpose. The task is to forecast the sewer flow in the 1-hour intervals.

Step 1: Import the data. Separate the data as training and test set. Plot the training set as shown in Figure 4.27.

```
%Import the data from the data file 'sewer_hourly.mat'. This file includes 500 data points, where 450 data
points (90% of the data) is chosen as training data set. The 50 data points are chosen as the test data set.
Plot the training datasets.

rawdata=importdata('sewer_hourly.mat');
data1=rawdata(1:450,:);
data1=data1'
figure
plot(data1)
xlabel('Hour')
ylabel('Million gallons')
title('Hourly Sewer flow aggregated in million gallons per day')
```

Step 2: Construct training and testing data sets. 90% of the data (450 data points) is chosen as training data set. The remaining 10% of the data (50 data points) is chosen as the test data set.

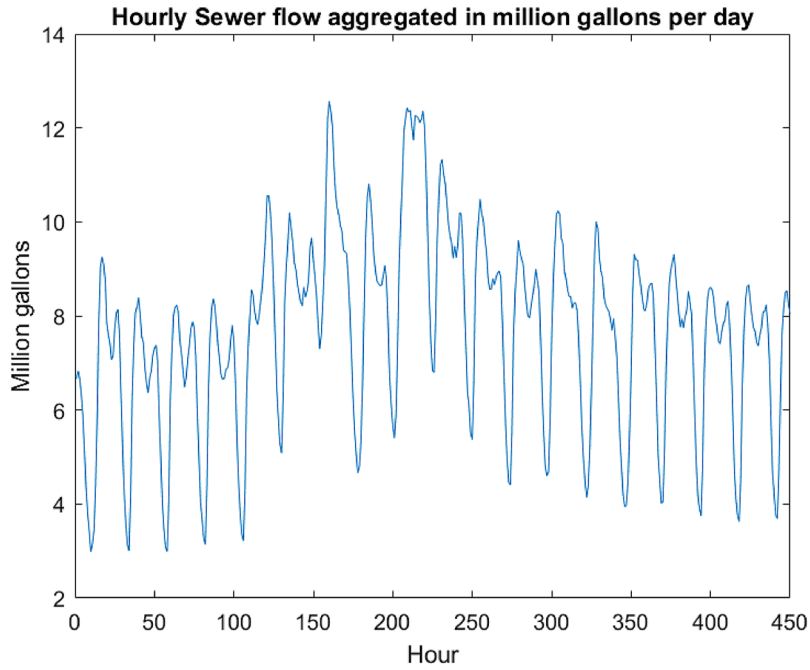

Figure 4.27 LSTM training data set.

```
data = rawdata(1:500,:);
data = data';

numTimeStepsTrain = 450;

% The data with index 1 to numTimeStepsTrain + 1 will be training set
% The data with index numTimeStepsTrain + 1 to end will be test set
dataTrain = data(1:numTimeStepsTrain + 1);
dataTest = data(numTimeStepsTrain + 1:end);

% Standardize the data by putting different data on the same scale. We calculate the mean and standard
deviation for each variable. Then, for each observed data, we subtract the mean and divide by the standard
deviation.

mu = mean(dataTrain);
sig = std(dataTrain);

dataTrainStandardized = (dataTrain – mu)/sig;

%XTrain is training data set
%YTrain is the response values of the training data set

XTrain = dataTrainStandardized(1:end–1);
YTrain = dataTrainStandardized(2:end);
```

Step 3: Configure the LSTM neural network.

```
% Set the LSTM regression network training option as follows: 250 hidden units.
numFeatures = 1;
numResponses = 1;
numHiddenUnits = 250;

layers = [...
 sequenceInputLayer(numFeatures)
 lstmLayer(numHiddenUnits)
 fullyConnectedLayer(numResponses)
      regressionLayer];

% Set the maximum epochs to 250. Gradient threshold to 1. Learn rate determines the step size at each
iteration while moving toward a minimum of a loss function. Initial learn rate 0.005. After 125 epochs, the
learn rate will be multiplied by a factor of 0.2.
options = trainingOptions('adam',...
 'MaxEpochs',250,...
 'GradientThreshold',1,...
 'InitialLearnRate',0.005,...
 'LearnRateSchedule','piecewise',...
 'LearnRateDropPeriod',125,...
 'LearnRateDropFactor',0.2,...
 'Verbose',0,...
 'Plots','training-progress');
```

Step 4: Train the LSTM neural network.

```
% Generate a trained recurrent neural network model in variable 'net'
net = trainNetwork(XTrain,YTrain,layers,options);
```

Step 5: Validate the trained LSTM model. The forecasting results are showed in Figure 4.28 and compared with the observed results shown in Figure 4.29. The RMSE values for LSTM forecast model are shown in Figure 4.30.

```
dataTestStandardized = (dataTest - mu)/sig;
XTest = dataTestStandardized(1:end-1);

% predictAndUpdateState function: Predict responses using a trained recurrent neural network 'net' and
update the network state

net = predictAndUpdateState(net,XTrain);

% YPred variable stores the forecast results of 50 data points

[net,YPred] = predictAndUpdateState(net,YTrain(end));

numTimeStepsTest = numel(XTest);
for i = 2:numTimeStepsTest
 [net,YPred(:,i)] = predictAndUpdateState(net,YPred(:,i-1),'ExecutionEnvironment','cpu');
end
```

```
YPred = sig*YPred + mu;

% YTest variable stores the observed results of 50 data points

YTest = dataTest(2:end);

% Plot the forecast results

rmse = sqrt(mean((YPred-YTest).^2));

figure
plot(dataTrain(1:end−1))
hold on
idx = numTimeStepsTrain:(numTimeStepsTrain + numTimeStepsTest);
plot(idx,[data(numTimeStepsTrain) YPred],'.-')
hold off
xlabel('Hourly sewer flows')
ylabel('Million gallons')
title('Forecast 50 red data points in the future')
legend(['Observed' 'Forecast'])

% Compare the forecast results with the observed results.

figure
plot(YTest)
hold on
plot(YPred,'.-')
hold off
legend(['Observed' 'Forecast'])
ylabel('Million gallons')
title('Forecast vs Observed')

%% Quantitative evaluation of forecast results using RMSE.

figure(),
stem(YPred – YTest)
xlabel('Hourly sewer flows')
ylabel('Error')
title('RMSE = ' + rmse)
```

4.5 OTHER POPULAR ML TECHNIQUES

4.5.1 Ensemble learning

In this section, we demonstrate how ensemble methods may be used to combine multiple MLT to improve the solution of regression and classification problems, with practical applications to a real case study, using high-resolution water-flow measures. All the applications reported in this paragraph are made available in the Github repository (https://github.com/Water-End-Use-Dataset-Tools/EL-WaterDemandTS). An ensemble includes a number of learners called base learners, usually generated from training data by a base learning algorithm which can be a decision tree, neural network or other kinds of learning algorithms. They try to build a set of learners from training data and combine them (Dong *et al.*, 2020). The use of ensemble methods is related to the possibility of achieving higher predictive performance than using an individual algorithm by itself (Zhou, 2012). In this section, the example code is given in Python for the variety of coding capacities (and it is also free!)

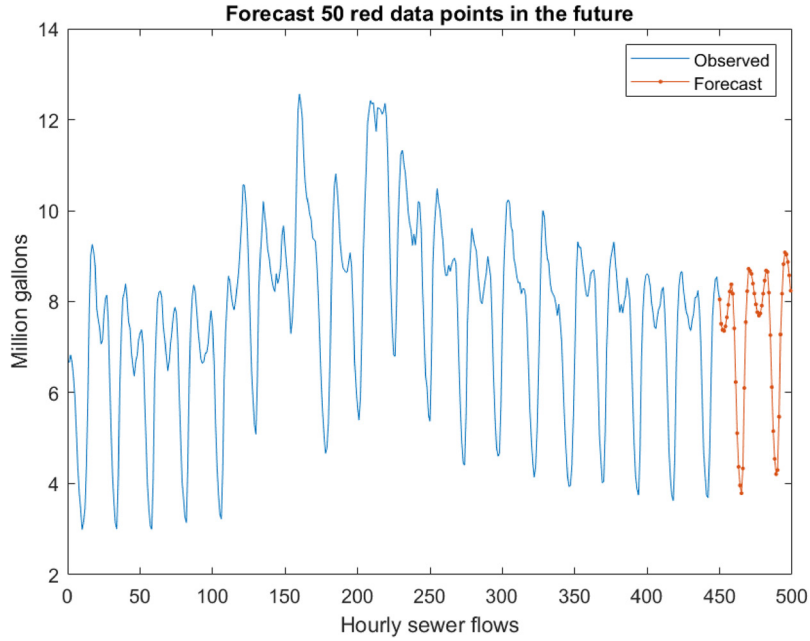

Figure 4.28 LSTM forecast results.

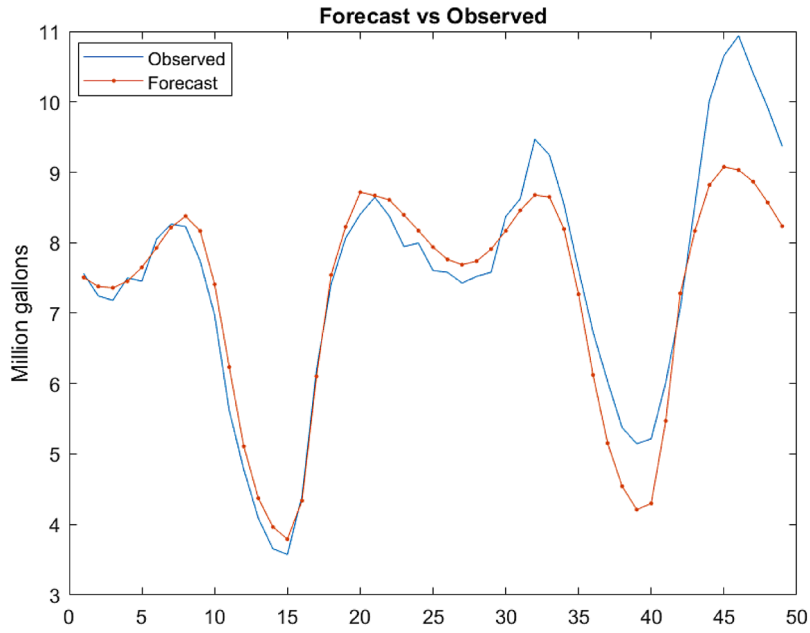

Figure 4.29 LSTM forecast (testing) results compared with observed results.

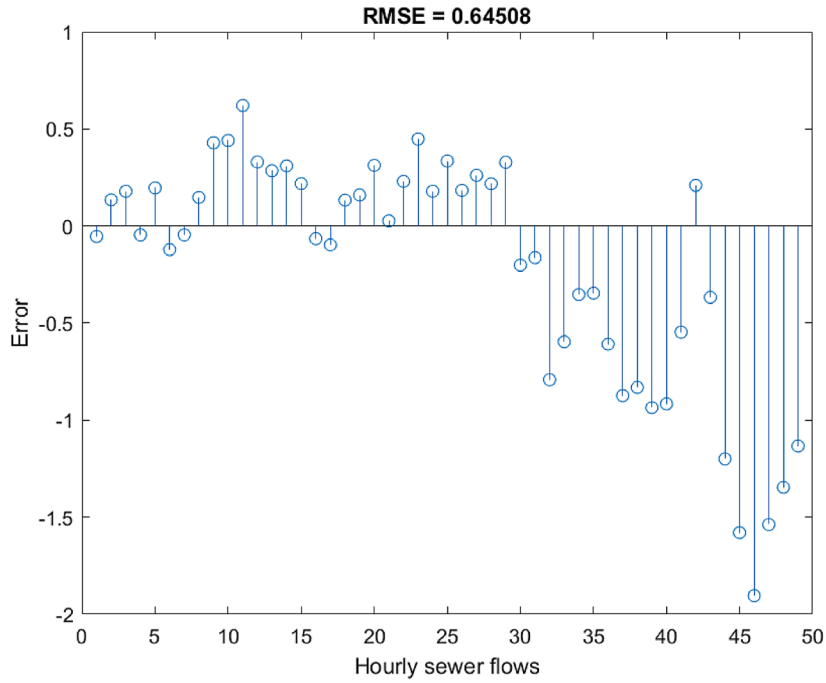

Figure 4.30 RMSE for LSTM forecast model.

4.5.1.1 Water end use dataset

For the applications reported in this paragraph, a dataset of water end use consumption is used. The dataset has been generated processing the water consumption measured at different fixtures of a domestic pilot and collected as water flow time-series. Each time-series contains the water-flow data in ml/sec with a sample period of 1 sec (Di Mauro *et al.*, 2019).

The water_usages dataset is a list of records provided as a CSV (comma separated values). Each record characterizes the occurrence of a water usage and is described by the following parameters:

- *start_date_time*: long [sec] it is the starting date-time of the usage as Unix epoch
- *duration*: int [ms], how long lasts the usage
- *liters*: int [mL], how many liters of water have been consumed
- *month*:int, month of occurrence
- *hour*:int, hour of the day
- *day*: int, day of the week {0,…,6}
- *max_flow:* int [mL/sec], maximum flow rate measured during the usage
- *av_flow_rate*: float [mL/sec], the average flow rate calculates for the usage
- *sec_from_midnight:* int, the number of seconds after the midnight
- *fixture*: string, the lable that identifies the fixture (e.g., shower, washbasin, etc.)
- *num_fixture:* int, an integer that identifies the fixture (e.g., 0: shower, 1: washbasin, …)

The original time-series have been split to identify every single usage, and then the usages have been clustered to identify similar water consumption profiles (e.g. hand washing, teeth brushing). The individual time-series excerpts will be also used later in this chapter. The complete dataset is available in a different GitHub repository (https://github.com/Water-End-Use-Dataset-Tools/WEUSEDTO)

4.5.1.2 Bootstrapping

Bootstrapping is a statistical method that resamples a single dataset to create many simulated samples. Applying the bootstrap method works like collecting many datasets. Increasing the dataset and computing the mean of the means estimates will eventually lead to a zero bias. In other words, it aims at computing an unbiased estimator of the population mean. The bootstrapping process allows us to evaluate statistics on a population which is obtained by sampling a dataset with replacement in order to make the selection procedure completely random. Bootstrapping is commonly useful to evaluate statistics such as the mean, standard deviation, construct confidence intervals and perform hypothesis testing for different types of statistics samples. It is used in applied machine learning to value the ability of machine learning models when making predictions on data not included in the training dataset. The importance of bootstrap sampling is related to their use as a basic step for several modern MLT, as for example the bagging technique used in various ensemble machine learning algorithms like random forests, gradient boost, and so on. Moreover, bootstrapping can be used to estimate the parameters of a population when the data sample available is not large enough to assume that the sampling distribution is normally distributed. Bootstrapping uses the distribution of the sample statistics among the simulated samples as the sampling distribution. The application reported below shows an example of mean evaluation on resampled datasets.

***Bootstrap method formulation*:** Let there be a sample X of size N. We can make a new sample from the original sample by drawing N elements from the latter randomly and uniformly, with replacement. In other words, we select a random element from the original sample of size N and do this N times. All elements are equally likely to be selected, thus each element is drawn with the equal probability $1/N$. More details on the bootstrap method can be found in (Efron & Tibshirani, 1993).

***Bootstrapping example application and code*:** The water_usages data-set has been used here to demonstrate how bootstrapping works. The amount of water consumed on each usage is the feature that is used in the model. Let us visualize in Figure 4.31 the data and look at the distribution of this feature for two fixtures, which are the washbasin and the kitchen faucet.

```
import numpy as np
import pandas as pd
import seaborn as sns

sns.set()
from matplotlib import pyplot as plt

#Graphics in retina format are more sharp and legible %config InlineBackend.figure_format = 'retina'
water_data = pd.read_csv('.data/dataset.csv', delimiter = ' ')

water_data.loc[water_data['fixture'] == 'washbasin', 'liters'].hist(label = 'Washbasin')

water_data.loc[water_data['fixture'] == 'kitchen faucet', 'liters'].hist(label = 'kitchen faucet')

plt.xlabel('mL')
plt.ylabel('Density')
plt.legend();
```

It is straightforward to observe that the washbasin is used more often than the kitchen faucet. Moreover, a higher percentage of usages consume less water in the case of the washbasin with respect to the kitchen faucet. Now, it may be a good idea to estimate the average amount of water consumed for each fixture for designing predictive strategies of water management. Since our dataset is small,

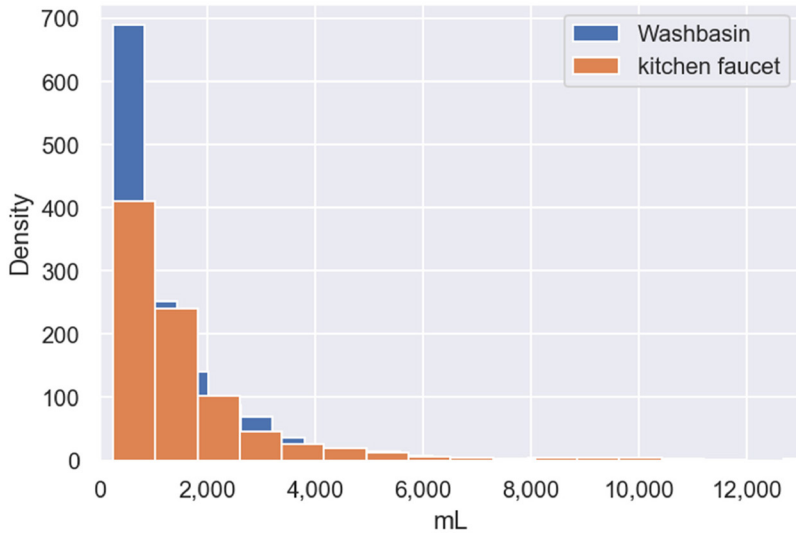

Figure 4.31 Data distribution of washbasin and kitchen faucet fixtures.

and the number of samples is different for the two fixtures (washbasin: 1354, kitchen faucet: 895), we would not get a good estimate by simply calculating the mean of the original sample. With a small dataset the estimation of the mean value could be different from the mean value of the population. Such a difference is called bias. We will be better off applying the bootstrap method. Let us generate 5000 new bootstrap samples from our original population and produce an interval estimate of the mean.

```
def get_bootstrap_samples(data, n_samples):
    '''Generate bootstrap samples using the bootstrap method.''' indices=
    np.random.randint(0, len(data), (n_samples, len(data))) samples=
    data[indices]
    return samples

def stat_intervals(stat, alpha):
    '''Produce an interval estimate.'''
    boundaries=np.percentile(stat, [100 * alpha / 2.0, 100 * (1 – alpha / 2.0)])
    return boundaries

#Save the data about the washbasin and kitchen faucet to split the dataset
wb_liters=water_data.loc[water_data['fixture']== 'washbasin', 'liters' ].values
kit_liters=water_data.loc[ water_data['fixture']== 'kitchenfaucet', 'liters'].values

#Set the seed for reproducibility of the results
np.random.seed(0)

#Generate the samples using bootstrapping and calculate the mean for each of them
wb_liters_mean_scores=[

   np.mean(sample) for sample in get_bootstrap_samples(wb_liters, 5000)]
kit_liters_mean_scores=[
```

```
    np.mean(sample) for sample in get_bootstrap_samples(kit_liters, 5000)]

#Print the resulting interval estimates
print('mliters consumed by washbasin: mean interval',
stat_intervals(wb_liters_mean_scores, 0.05))
print('mliters consumed by kitchenfaucet: mean interval',
            stat_intervals(kit_liters_mean_scores, 0.05))
```

As a result, the mean interval for the milliliters consumed by washbasin and kitchen faucet are respectively: [1344, 1547] and [1645, 1953].

In Figure 4.32, the same procedure is applied to compare different fixtures as kitchen faucet and shower. It is straightforward to observe that the shower is used less often than the kitchen faucet. Moreover, shower usages usually consume more water respect to the kitchen faucet, with a reduced variance.

As a result, the mean interval for the milliliters consumed by washbasin and shower are respectively: [20056, 24218] and [1344, 1547])

4.5.1.3 Bagging

The bagging is a machine learning ensemble algorithm realized to improve the stability and accuracy of algorithms used for statistical classification and regression. Bagging, also called bootstrap aggregating, trains multiple models of the same learning algorithm on bootstrapped samples of the original dataset, and then aggregates their individual predictions to produce a final prediction as shown in Figure 4.33. Bagging is typically used with decision trees and this kind of MLT prevents overfitting, reducing the variance of a classifier by decreasing the difference in error when the model is trained on different datasets. Besides the use of the bagging technique to reduce model overfitting, it is used in the case of high-dimensional data due to its good performance. Furthermore, possible missing values in the dataset do not alter the execution of the algorithm. More details on the bagging method can be found in Bühlmann and Yu (2002).

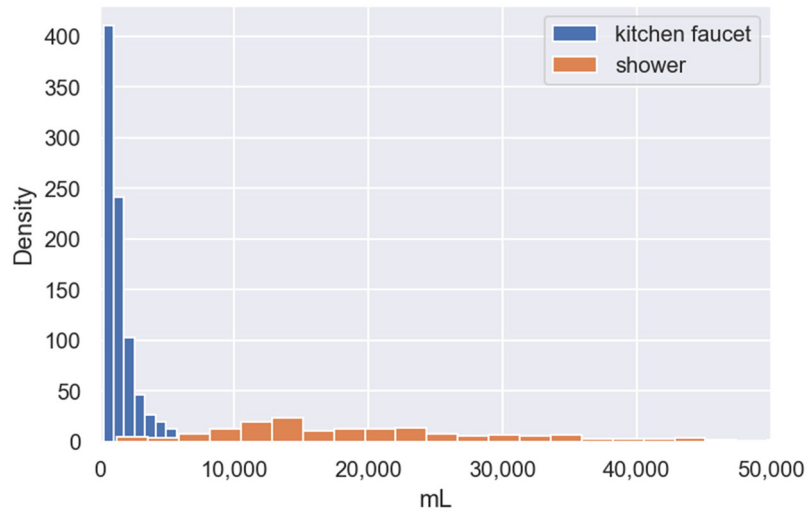

Figure 4.32 Data distribution of kitchen faucet and shower fixtures.

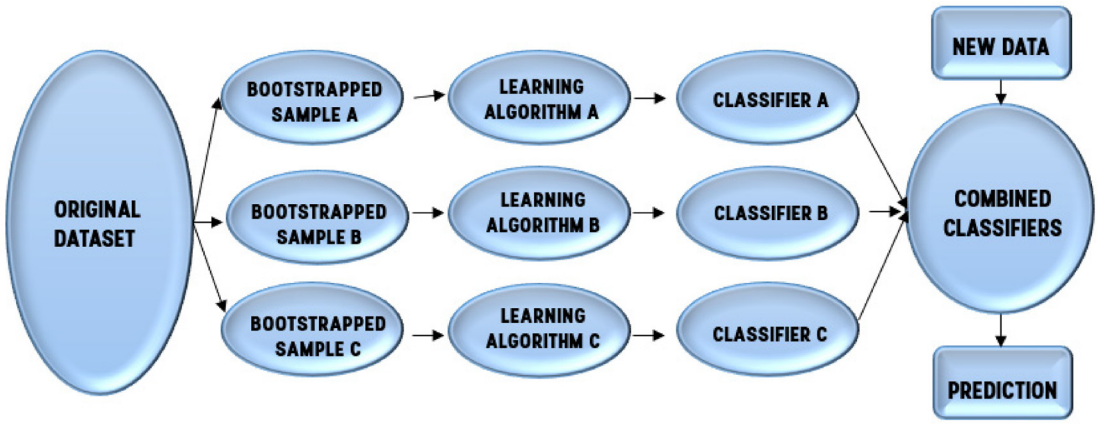

Figure 4.33 Diagram of bagging technique.

***Bagging method formulation*:** Bagging method formulation was presented by Breiman as reported in Breiman (1996). Consider a training set X then $X_1, \ldots, X_M$ are generated using bootstrapping. Now, a classifier $a_i(x)$ is trained for each bootstrap sample. The final classifier will average the outputs from all these individual classifiers:

$$a(x) = \frac{1}{M}\sum_{i=1}^{M} a_i(x) \tag{4.26}$$

Figure 4.33 illustrates the bagging algorithm.

Let us consider a regression problem with base algorithms $b_1(x), \ldots, b_n(x)$. Assume that there exists an ideal target function of true answers $y(x)$ defined for all inputs and that the distribution $p(x)$ is defined. Then the error can be expressed for each regression function as follows:

$$\varepsilon_i(x) = b_i(x) - y(x), \quad i = 1, \ldots, n \tag{4.27}$$

and the expected value of the mean squared error:

$$E_x[(b_i(x) - y(x))^2)] = E_x\left[\varepsilon_i^2(x)\right] \tag{4.28}$$

Then, the mean error over all regression functions will look as follows:

$$E_1 = \frac{1}{n} E_x\left[\sum_{i}^{n} \varepsilon_i^2(x)\right] \tag{4.29}$$

Assuming that the errors are unbiased and uncorrelated, that is:

$$E_x\left[\varepsilon_i(x)\right] = 0, \tag{4.30}$$

$$E_x\left[\varepsilon_i(x)\varepsilon_j(x)\right] = 0, \quad i \neq j \tag{4.31}$$

Now, let us construct a new regression function that will average the values from the individual functions:

$$a(x) = \frac{1}{n}\sum_{i=1}^{n} b_i(x) \tag{4.32}$$

Let us find its mean squared error:

$$E_n = E_x\left[\frac{1}{n}\sum_{i=1}^{n} b_i(x) - y(x)\right]^2 \tag{4.33}$$

$$= E_x\left[\frac{1}{n}\sum_{i=1}^{n} \varepsilon_i\right]^2 \tag{4.34}$$

$$= \frac{1}{n^2} E_x\left[\sum_{i=1}^{n} \varepsilon_i^2(x) + \sum_{i\neq j} \varepsilon_i(x)\varepsilon_j(x)\right] \tag{4.35}$$

$$= \frac{1}{n} E_1 \tag{4.36}$$

Thus, by averaging the individual answers, the mean squared error can be reduced by a factor of n. Let us recall the components that make up the total out-of-sample error:

$$Err(\vec{x}) = E\left[(y - \hat{f}(\vec{x}))^2\right] \tag{4.37}$$

$$= \sigma^2 + f^2 + Var(\hat{f}) + E[\hat{f}]^2 - 2fE[\hat{f}] \tag{4.38}$$

$$= (f - E[\hat{f}])^2 + Var(\hat{f}) + \sigma^2 \tag{4.39}$$

$$= Bias(\hat{f})^2 + Var(\hat{f}) + \sigma^2 \tag{4.40}$$

***Bagging example application and code*:** In this example, similarly to what is presented in section 7.3 of Hastie *et al.* (2009), we show and compare the variance of the expected mean squared error of a single estimator against a bagging ensemble in a regression problem applied to time-series of real data. A cluster of washbasin usages has been used as they were noisy measures of the same water consumption profile (e.g., hand washing), and the spline that approximates all measures of the cluster as the true profile. Figure 4.34 shows the results of the application: the upper left figure illustrates the predictions (in dark dashed line) of a single decision tree that has been trained over a down-sampled time-series of one usage profile. It also illustrates the predictions (in light dashed line) of other single decision trees trained over the down-sampled consumption profiles of the cluster. The variance term in this application corresponds to the width of the bundle of predictions (in light dashed line) of the individual estimators. The predictions for x are more sensitive. The lower left figure plots the pointwise decomposition of the expected mean squared error of a single decision tree. It shows the variance in the rectangular marker line and also illustrates the noise part of the error which, as expected, appears to be comparable to the variance as we considered real profiles as a noisy version of the cluster centroid. The figures on the right reported to the same plots using a bagging ensemble of decision trees. In terms of variance, the bundle of predictions is narrower, which indicates that the variance is lower. Moreover, as shown by the lower right figure, the variance term (rectangular marker line) is lower than for single decision trees.

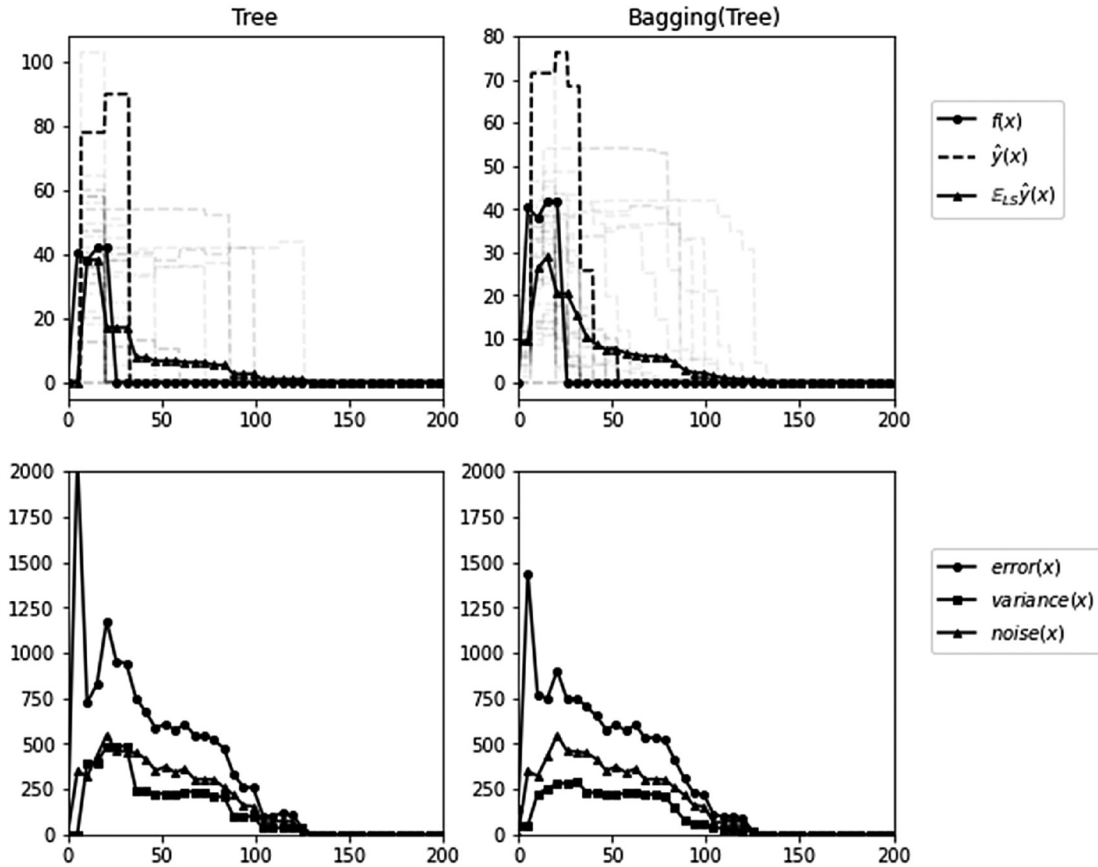

Figure 4.34 Bagging application to washbasin time series.

Comparing the circle marker line in the lower graphics, it is worth noticing that for bagging the overall mean squared error is lower. It depends on the fact that for bagging, averaging several decision trees fit on bootstrap copies of the dataset allows for a reduction of the variance.

The total error of the bagging ensemble is a wee bit lower than the total error of a single decision tree, this difference hinges on the reduced variance.

In Figure 4.34, $f(x)$ is the true function $y(x)$ are the estimators, $\mathcal{E}l_s\hat{y}(x)$ is the average of the estimators, error(x) is the mean square error between the true value and one estimator, noise(x) is the variance of the measured timeseries (it is evaluated on the test set that represent noisy measures).

4.5.1.4 Random forest

Random forest (RF) is one of the most popular machine learning algorithms. It was introduced by Breiman as an ensemble tree learner (Breiman 2001). The algorithm consists of many decision trees, each with the same nodes, built using a different bootstrap sample of the data from the original training dataset. RF merges the prediction result from every decision tree in order to find an answer, which represents the average of all the decision trees. It selects the best solution by means of voting, the most voted is chosen as the final prediction, as shown in Figure 4.35. One of the advantages of random forest is its flexibility, in fact, it is used to solve both regression and classification problems. It is used mostly because it is not influenced by noise, and due to the presence of several trees in the

```
from scipy.interpolate import interp1d
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import BaggingRegressor
from sklearn.tree import DecisionTreeRegressor
import glob

# Settings
n_repeat=50                    # Number of iterations for computing expectations
n_train=50                     # Size of the training set
n_test=1000                    # Size of the test set
np.random.seed(0)

estimators=[('Tree', DecisionTreeRegressor()),
                        ('Bagging(Tree)', BaggingRegressor(DecisionTreeRegressor()))]

n_estimators=len(estimators)
ts_files=glob.glob('data/csv_Washbasin/cluster_0/*.csv')
f_true='data/csv_Washbasin/1_spline.csv'

def f(x, iteration):
 if iteration==-1:
  ts=np.genfromtxt(ts_files[4], delimiter=' ')
 else:
  ts=np.genfromtxt(ts_files[iteration], delimiter=' ')
  start_time=ts[0,0]
  ts[:,0] -=start_time
  ts[0,1]=0
  if ts[-1,0] < 650:
                        ts=np.vstack((ts,[ts[-1,0]+1, 0]))
                        ts=np.vstack((ts,[650, 0]))
            for i in range(1,len(ts)-1):
                        if ts[i,1]==0 and ts[i+1,1]!=0:
                            ts[i,1]=(ts[i-1,1]+ts[i+1,1])*0.5
  linfunc=interp1d(ts[:,0], ts[:,1])
  return linfunc(x)

def generate(n_samples, n_repeat=1):
 max_duration=650
 X=np.linspace(0, 650, n_samples)
 if n_repeat== 1:
  y=f(X, np.random.randint(1,len(ts_files)))
 else:
  y=np.zeros((n_samples, n_repeat))
 for i in range(n_repeat):
  y[:, i]=f(X, np.random.randint(1,len(ts_files)))
 X=X.reshape((n_samples, 1))
 return X, y

X_train []
y_train=[]
```

```
for i in range(n_repeat):
    X, y=generate(n_samples=n_train)
    X_train.append(X)
    y_train.append(y)
X_test, y_test=generate(n_samples=n_test,n_repeat=n_repeat)
plt.figure(figsize=(10, 8))
# Loop over estimators to compare
for n, (name, estimator) in enumerate(estimators):
    # Compute predictions
    y_predict=np.zeros((n_test, n_repeat))
    for i in range(n_repeat):
        estimator.fit(X_train[i], y_train[i])
        y_predict[:, i]=estimator.predict(X_test)

    y_error=np.zeros(n_test)
    for i in range(n_repeat):
        for j in range(n_repeat):
            y_error+=(y_test[:, j] – y_predict[:, i]) ** 2

    y_error /=(n_repeat * n_repeat)
    y_noise=np.var(y_test, axis=1)
    y_var=np.var(y_predict, axis=1)

# Plot figures
plt.subplot(2, n_estimators, n+1)
plt.plot(X_test, f(X_test,–1), 'b', label='$f(x)$')

for i in range(n_repeat):
 if i== 0:
  plt.plot(X_test, y_predict[:, i], 'r', label=r'$\^y(x)$')
 else:
  plt.plot(X_test, y_predict[:, i], 'r', alpha=0.05)

plt.plot(X_test, np.mean(y_predict, axis=1), 'c', label=r'$\mathbb{E}_{LS} \^y(x)$')
plt.xlim([0, 350])
plt.title(name)

if n== n_estimators – 1:
 plt.legend(loc=(1.1, 0.5))

plt.subplot(2, n_estimators, n_estimators+n+1) plt.plot(X_test,
y_error, 'r', label='$error(x)$')        plt.plot(X_test,  y_var, 'g',
label='$variance(x)$'),  plt.plot(X_test,          y_noise,     'c',
label='$noise(x)$')
pltxlim([0, 350])
plt.ylim([0, 2000])
if n== n_estimators – 1:
 plt.legend(loc=(1.1, 0.5))

plt.subplots_adjust(right=.75)
plt.show()
```

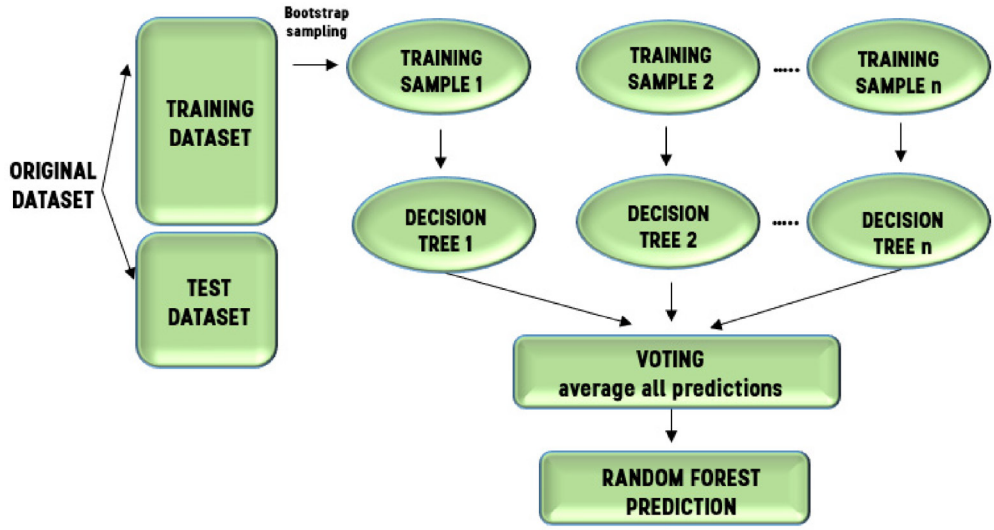

Figure 4.35 Diagram of random forest technique.

forest, it will not overfit the model. RF also has some limitations in terms of computation, which becomes slower as the number of trees in the model is larger. More details on random forest method can be found in Hastie *et al.* (2009).

***Random forest method formulation*:** The random forest method formulation was presented by Breiman as reported in Breiman (2001) and Hastie *et al.* (2009).

The algorithm for building a random forest of N trees goes as follows:

For each $b = 1, \ldots, N$;

- Draw a bootstrap sample Xb;
- Build a decision tree T_b on the bootstrap sample Xbrepeating the following steps:
  - Pick the best feature according to the given criteria. Split the sample by this feature to create a new tree level. Repeat this procedure until the sample is exhausted;
  - Building the tree until any of its leaves contains no more than n_{min} instances;
  - For each split, first randomly pick m features from the original ones and then search for the next best split only among the subset.

Output the ensemble of trees $\{T_b\}_1^N$

The final prediction at a new point x is defined:

For Regression by: $f(x) = \frac{1}{N}\sum_{b=1}^{N} T_b(x)$

For Classification by: Let $C_b(X)$ be the class prediction of the bth random forest tree.

Then $c(x) = majorityvote\{C_b\}_1^N$

When the RF algorithm is used for regression problems, the mean squared error (MSE) is used to evaluate the distance of each node from the predicted value in order to select which branch represents the best decision for the forest:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(f_i - y_i)^2 \tag{4.41}$$

where N is the number of data points, f_i is the value returned by the decision tree and y_i is the value of the data point you are testing at a certain node.

When the RF algorithm is used for classification problems, the Gini index is used to determine how nodes are on a decision tree branch. The class and probability are used to determine the Gini of each branch on a node, establishing which of the branches is more likely to occur:

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2 \tag{4.42}$$

where *pi* is the relative frequency of the class observed in the dataset and *c* is the number of classes.

***Random forest regression example application and code*:** In the following example machine learning techniques have been used to predict the water consumption profiles of a daily usage from a down-sampled time-series of water-flow measures. In particular, the examples start from the high-resolution time-series of the daily water consumption of a kitchen faucet. The training set is composed of 100 samples, which are randomly selected from the original time-series.

```
import numpy as np
from scipy.interpolate import interp1d
from matplotlib import pyplot as plt

import seaborn as sns
from sklearn.datasets import make_circles
from sklearn.ensemble import (BaggingClassifier,
    BaggingRegressor, RandomForestClassifier, RandomForestRegressor)
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
import glob
import random

n_train=100
n_test=779

#Generate data
def generate(n_samples):
        ts=np.genfromtxt('data/oneday_kitchen.csv', delimiter=' ')
        start_time=ts[0,0]
        ts[:,0] -=start_time
        X=random.sample(range(0, len(ts)), n_samples)
        X.sort()y=ts[X,1]
        X=np.reshape(X, (n_samples, 1))
        return X, y

X_train, y_train=generate(n_samples=n_train)
X_test, y_test=generate(n_samples=n_test)
```

In the first example, a decision tree is used to predict all 779 samples of the original one, as shown in Figure 4.36.

```
# One decision tree regressor
dtree = DecisionTreeRegressor().fit(X_train, y_train)
d_predict = dtree.predict(X_test)
plt.figure(figsize = (10, 6))
plt.plot(X_test, y_test, color = '0.5',linestyle = 'dashed')
plt.scatter(X_train, y_train, c = 'b', s = 20)
plt.plot(X_test, d_predict, 'g', lw = 2)
plt.title('Decision tree, MSE = %.2f' % np.divide(np.sum((y_test – d_predict) ** 2),n_test))
```

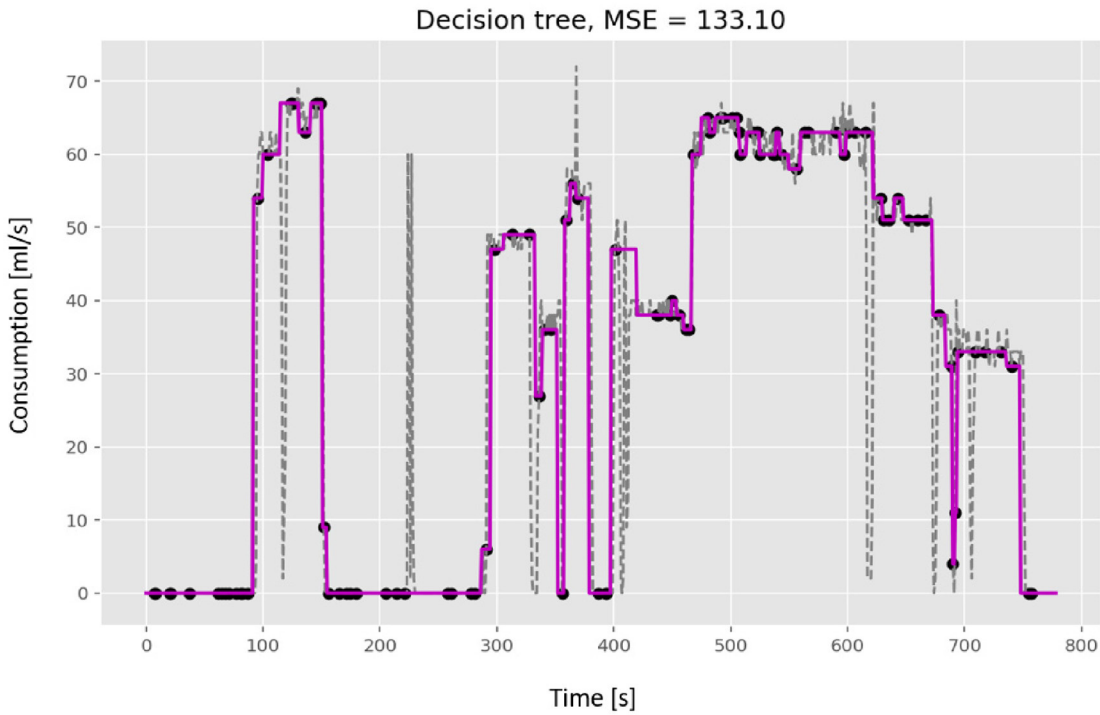

Figure 4.36 Decision tree to daily water consumption of a kitchen faucet.

In the second example, the bagging regressor uses ten trees to generate the solution, presenting a lower MSE, as shown in Figure 4.37.

```
# Bagging with a decision tree regressor
bdt = BaggingRegressor(DecisionTreeRegressor()).fit(X_train, y_train)
bdt_predict = bdt.predict(X_test)

plt.figure(figsize = (10, 6))
plt.plot(X_test, y_test, color = '0.5',linestyle = 'dashed')
plt.scatter(X_train, y_train, c = 'b', s = 20)
plt.plot(X_test, bdt_predict, 'y', lw = 2)
plt.title('Bagging for decision trees, MSE = %.2f'
% np.divide(np.sum((y_test – bdt_predict) ** 2),n_test));
```

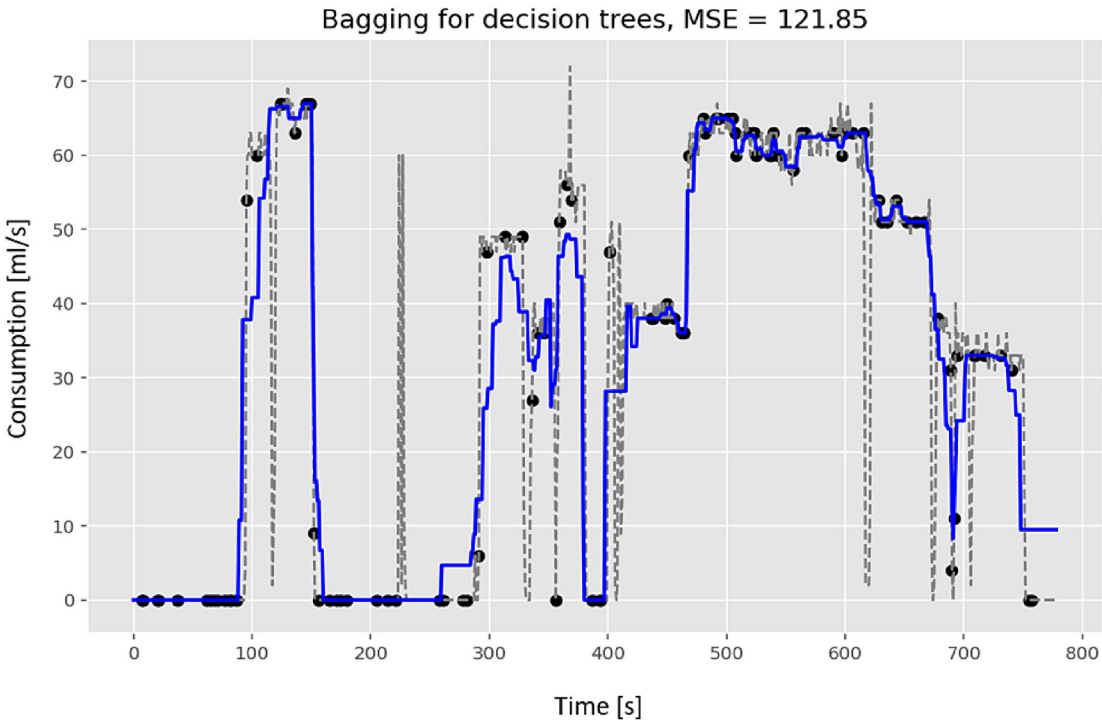

Figure 4.37 Bagging to daily water consumption of a kitchen faucet.

Finally, the random forest regressor is used to solve the same problem with the same number of decision trees. The example shows a comparison between random forests and bagging. It can be observed that, in a random forest, the best feature for a split is selected from a random subset of the available features, while in bagging all features are considered for the next best split. This represents the main difference between the two methods. The effect is, at least in this example, a slight improvement of the MSE as shown in Figure 4.38.

```
# Random Forest
rf=RandomForestRegressor(n_estimators=10).fit(X_train, y_train)
rf_predict=rf.predict(X_test)

plt.figure(figsize=(10, 6))
plt.plot(X_test, y_test, color='0.5',linestyle='dashed')
plt.scatter(X_train, y_train, c='b', s=20)
plt.plot(X_test, rf_predict, 'r', lw=2)
plt.title('Random forest, MSE=%.2f' % np.divide(np.sum((y_test – rf_predict) ** 2),n_test));
```

***Random forest classification example application and code*:** The following example looks at the advantages of random forests and bagging in classification problems. The goal is to classify the corresponding fixture of each water usage using two features. In this example, random forest classification has been applied to three fixtures: washbasin, shower and kitchen faucet. In particular, the following code reads from the dataset of water usages, the time of the day (seconds), the volume in liters and the related fixture for washbasin, shower and kitchen faucet.

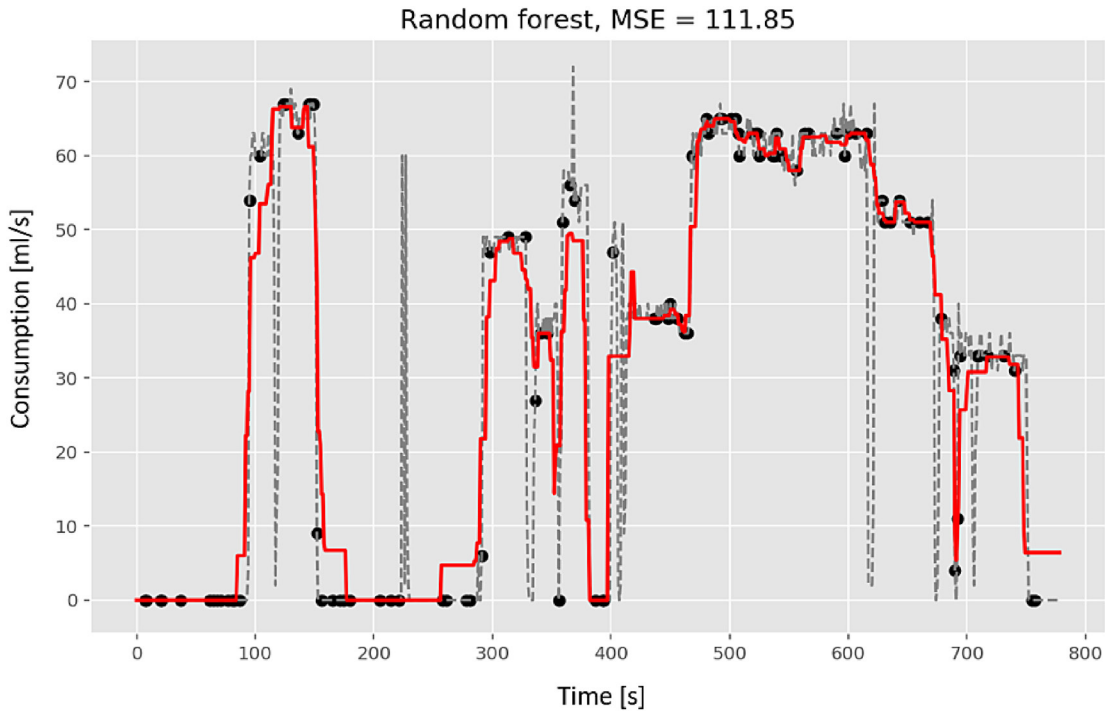

Figure 4.38 Random forest to daily water consumption of a kitchen faucet.

The fixtures are represented as an integer from 0 to 2. Twenty per cent of usages are used as the training set.

```
from matplotlib import pyplot as plt
import seaborn as sns
from sklearn.datasets import make_circles
from sklearn.ensemble import (BaggingClassifier, BaggingRegressor,
        RandomForestClassifier, RandomForestRegressor)
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
import pandas as pd
import sklearn

# Load data
df=pd.read_csv('./data/dataset.csv', delimiter=' ')
fixtures=['washbasin', 'shower', 'kitchenfaucet']
# Choose the numeric features
df=df[['sec_from_midnight','liters','fixture', 'num_fixture']]
df=df[(df['fixture']==fixtures[0]) | (df['fixture']==fixtures[1]) | (df['fixture']==fixtures[2])] plt
df.head()
df=sklearn.utils.shuffle(df)
X=np.asarray(df[['sec_from_midnight','liters']],dtype=float)
max_dur=max(X[:,0])
```

```
max_lit=max(X[:,1])
X[:,0]=X[:,0] /3600
X[:,1]=X[:,1]/1000
Y=df['num_fixture']-2

X_train_circles, X_test_circles, y_train_circles, y_test_circles=\
train_test_split(X, Y, test_size=0.2)
```

The next code applies three decision trees, a bagging and a random forest with 300 estimators to address such classification problem. The learned model is used to classify the points of a 2D grid.

```
def plot_class (X,Y, xx1,xx2,y_hat, title):
  fig, ax=plt.subplots()
  plt.contourf(xx1, xx2, y_hat, alpha=0.2)
  plt.scatter(X[:,0], X[:,1], c=Y, cmap='viridis', alpha=.7)
  handles, labels=scatter.legend_elements(prop='colors', alpha=0.6)
  legend2=ax.legend(handles, fixtures, loc='upper right')
  ax.add_artist(legend2)
  plt.title(title)
  ax.set_xlabel('hours')
  ax.set_ylabel('liters')
  ax.legend()
  plt.show()

x_range=np.linspace(X[:,0].min(), X[:,0].max(), 100)
y_range=np.linspace(X[:,1].min(), X[:,1].max(), 100)
xx1, xx2=np.meshgrid(x_range, y_range)

dtree=DecisionTreeClassifier()
dtree.fit(X_train_circles, y_train_circles)

y_hat=dtree.predict(np.c_[xx1.ravel(), xx2.ravel()])
y_hat=y_hat.reshape(xx1.shape)
plot_class(X,Y,xx1,xx2,yhat, 'Decision tree')

dtree=BaggingClassifier(DecisionTreeClassifier(),
          n_estimators=300, random_state=42)
b_dtree.fit(X_train_circles, y_train_circles)
y_hat=b_dtree.predict(np.c_[xx1.ravel(), xx2.ravel()])
y_hat=y_hat.reshape(xx1.shape)
plot_class(X,Y,xx1,xx2,yhat, 'Bagging (Decisione tree)')

rf=RandomForestClassifier(n_estimators=300, random_state=42)
rf.fit(X_train_circles, y_train_circles)
y_hat=rf.predict(np.c_[xx1.ravel(), xx2.ravel()])
y_hat=y_hat.reshape(xx1.shape)

plot_class(X,Y,xx1,xx2,yhat, 'Random Forest')
```

Figure 4.39 shows the results of the classification problems performed with the three methods described before. The models are used to classify a mesh-grid of 100×100 points in a 2D space whose dimensions range from the minimum to the maximum values of the two features of the training set.

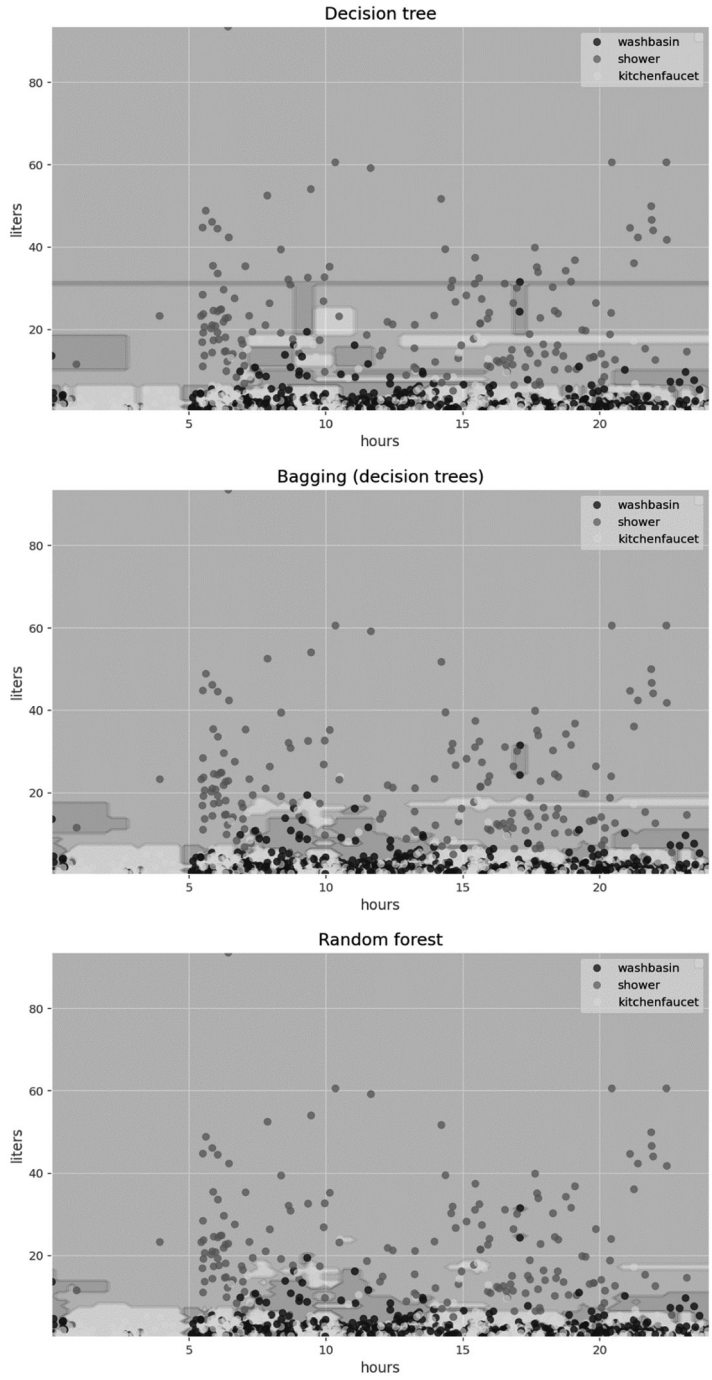

Figure 4.39 Random forest classification of washbasin, shower and kitchen faucet fixtures.

Each chart shows how the points of the mesh-grid have been clustered, coloring the associated region with a gradient of the corresponding fixture. For example, a point in a strip represented by darker gray shade then it will be classified as belonging to the dark gray fixture. Figure 4.39 shows that the decision boundary of the decision tree is serrated, suggesting the presence of overfitting and a not clear definition of the class. This means that it is difficult to make reliable predictions for new test data. The bagging and random forest algorithms, on the other hand, show more regular bounds and no evident signs of overfitting.

REFERENCES

Arandia E., Ba A., Eck B. and McKenna S. 2015 Tailoring seasonal time series models to forecast short-term water demand. *Journal of Water Resources Planning and Management*, **142**(3), 04015067, https://doi.org/10.1061/(ASCE)WR.1943-5452.0000591

ArunKumar K. E., Kalaga D. V., Mohan Sai Kumar C., Chilkoor G., Kawaji M. and Brenza T. M. (2021). Forecasting the dynamics of cumulative COVID-19 cases (confirmed, recovered and deaths) for top-16 countries using statistical machine learning models: auto-regressive integrated moving average (ARIMA) and seasonal auto-regressive integrated moving average (SARIMA). *Applied Soft Computing Journal*, **103**, 107161, https://doi.org/10.1016/j.asoc.2021.107161

Bedi J. and Toshniwal D. (2019). Deep learning framework to forecast electricity demand. *Applied Energy*, **238**, 1312–1326, https://doi.org/10.1016/j.apenergy.2019.01.113

Billings R. B. and Jones C. V. (2008). Forecasting Urban Water Demand, 2nd edn. American Water Works Association, Denver, CO.

Box G., Jenkins G., Reinsel G. and Ljung G. (2016). Time Series Analysis: Forecasting and Control, 5th edn. John Wiley and Sons, Inc., Hoboken, NJ.

Breiman L. (1996). Bagging predictors. *Machine Learning*, **24**(2), 123–140.

Breiman L. (2001). Random forests. *Machine Learning*, **45**(1), 5–32, https://doi.org/10.1023/A:1010933404324

Bühlmann P. and Yu B. (2002). Analyzing bagging. *Annals of Statistics*, **30**(4), 927–961, https://doi.org/10.1214/aos/1031689014

Candelieri A. (2017). Clustering and support vector regression for water demand forecasting and anomaly detection. *Water*, **9**(3), 224, https://doi.org/10.3390/w9030224

Chen J. (2018). *Water Demand Datasets* (Publication no. 10.17632/4yhprsgjrf.1). Retrieved from: https://data.mendeley.com/datasets/4yhprsgjrf/1 (last accessed July 2021)

Cho K., Van Merriënboer B., Gulcehre C., Bahdanau D., Bougares F., Schwenk H. and Bengio Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv*, 1406.1078.

Cominola A., Giuliani M., Piga D., Castelletti A. and Rizzoli A. E. (2015). Benefits and challenges of using smart meters for advancing residential water demand modeling and management: a review. *Environmental Modelling and Software*, **72**, 198–214, https://doi.org/10.1016/j.envsoft.2015.07.012

Di Mauro A., Di Nardo A., Santonastaso G. F. and Venticinque S. (2019). An IoT system for monitoring and data collection of residential water end-use consumption. Proceedings – International Conference on Computer Communications and Networks (ICCCN), IEEE, pp. 1–6.

Dong X., Yu Z., Cao W., Shi Y. and Ma Q. (2020). A survey on ensemble learning. *Frontiers of Computer Science*, **14**(2), 241–258, https://doi.org/10.1007/s11704-019-8208-z

Du B., Zhou Q., Guo J., Guo S. and Wang L. (2021). Deep learning with long short-term memory neural networks combining wavelet transform and principal component analysis for daily urban water demand forecasting. *Expert Systems with Applications*, **171**, 114571, https://doi.org/10.1016/j.eswa.2021.114571

Efron B. and Tibshirani R. J. (1993). An Introduction to the Bootstrap. Springer, Berkeley, CA.

Elman J. L. (1990). Finding structure in time. *Cognitive Science*, **14**(2), 179–211, https://doi.org/10.1207/s15516709cog1402_1

Faruk D. (2010). A hybrid neural network and ARIMA model for water quality time series prediction. *Engineering Applications of Artificial Intelligence*, **23**, 586–594, https://doi.org/10.1016/j.engappai.2009.09.015

Goldberg Y. J. J. o. A. I. R. (2016). A primer on neural network models for natural language processing. **57**, 345–420.

Guo G., Liu S., Wu Y., Li J., Zhou R. and Zhu X. (2018). Short-term water demand forecast based on deep learning method. *Journal of Water Resources Planning and Management*, **144**(12), 04018076, https://doi.org/10.1061/(ASCE)WR.1943-5452.0000992

Hastie T., Tibshirani R. and Friedman J. (2009). Elements of statistical learning. *The Mathematical Intelligencer*, **27**(2), 267–268.

Hochreiter S. and Schmidhuber J. (1997). Long short-term memory. *Neural Computation*, **9**(8), 1735–1780.

Kofinas D., Mellios N., Papageorgiou E. and Laspidou C. (2014). Urban water demand forecasting for the island of skiathos. *Procedia Engineering*, **89**, 1023–1030, https://doi.org/10.1016/j.proeng.2014.11.220

Lee J. and Chae S. (2016). Hourly water demand forecasting for micro water grids. *Journal of Water Supply Research and Technology – AQUA*, **65**(1), 12–17.

Mathworks. (2021a). Forecast Multiplicative ARIMA Model. Available at: https://www.mathworks.com/help/econ/forecast-airline-passenger-counts.html?searchHighlight=forecast%20multiplicative&s_tid=srchtitle (last accessed 9 March 2022)

Mathworks. (2021b). MMSE Forecasting of Conditional Mean Models. Available at: https://www.mathworks.com/help/econ/mmse-forecasting-for-arima-models.html#btbqqom (last accessed 9 March 2022)

Monks I., Stewart R. A., Sahin O. and Keller R. (2019). Revealing unreported benefits of digital water metering: literature review and expert opinions. *Water (Switzerland)*, **11**(4), 838–870, https://doi.org/10.3390/w11040838

Msiza I. S., Nelwamondo F. V. and Marwala T. (2007). Artificial neural networks and support vector machines for water demand time series forecasting. Paper presented at the 2007 IEEE International Conference on Systems, Man and Cybernetics, IEEE, pp. 638–643.

NCSS Statistical Software. The Box Jenkins Method, pp. 470-4 to 470-9. Available at: https://ncss-wpengine.netdna-ssl.com/wp-content/themes/ncss/pdf/Procedures/NCSS/The_Box-Jenkins_Method.pdf (last accessed 9 March 2022)

Pesantez J. E., Berglund E. Z. and Kaza N. (2020). Smart meters data for modeling and forecasting water demand at the user-level. *Environmental Modelling and Software*, **125**, 104633, https://doi.org/10.1016/j.envsoft.2020.104633

Rahim M. S., Nguyen K. A., Stewart R. A., Giurco D. and Blumenstein M. (2020). Machine learning and data analytic techniques in digital water metering: a review. *Water (Switzerland)*, **12**(1), 294–320.

Redondo E., Zafra-Mejía C. and Gutiérrez-Malaxechebarría A.-M. (2018). Operational analysis in a drinking water treatment plant using ARIMA models. *International Journal of Applied Engineering Research*, **13**, 16093–16099.

Sengupta A., Hawley R. J. and Stein E. D. (2018). Predicting hydromodification in streams using nonlinear memory-based algorithms in southern California streams. *Journal of Water Resources Planning and Management*, **144**(1), 04017079, https://doi.org/10.1061/(ASCE)WR.1943-5452.0000853

Villarin M. C. and Rodriguez-Galiano V. F. (2019). Machine learning for modeling water demand. *Journal of Water Resources Planning and Management*, **145**(5), 04019017, https://doi.org/10.1061/(ASCE)WR.1943-5452.0001067

Xenochristou M., Kapelan Z., Hutton C. and Hofman J. (2018). Smart water demand forecasting: learning from the data. *EPiC Series in Engineering*, **3**, 2351–2358, https://doi.org/10.29007/wkp4

Zhou Z. H. (2012). Ensemble methods: foundations and algorithms. In: Ensemble Methods: Foundations and Algorithms. Chapman and Hall/CRC Press, Taylor & Francis Group, New York, pp. 1–218.