An investigation into the use of machine learning techniques for forecasting inventory stock

Neethu Samuel

1892082

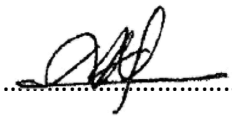
Supervisor: Dr. Joke Bührmann

A research report submitted to the Faculty of Engineering and the Built Environment, University of the Witwatersrand, Johannesburg, in partial fulfilment of the requirements for the degree of Master of Science in Engineering.

Johannesburg, 2020

Candidate's declaration

I declare that this research report is my own unaided work. It is being submitted for the Degree of Master of Science to the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination to any other university.



……

26th day of July, 2020

Abstract

Demand forecasting is the field of study that aims to predict customer demand. In the past, demand forecasting was achieved using traditional stochastic methods like the Holtz Winters method, ARIMA and moving averages. Machine learning techniques are techniques that can capture the characteristics of data more efficiently. Thus; machine learning techniques were explored for the purpose of demand forecasting in this research report.

The dataset used for this research is Kaggle's Historical Product Demand. This dataset consists of various product demand categories arranged in monthly logs from 2011 to 2017. There are 2172 product categories in the original dataset. After processing, there are 1803 product categories. The dataset is essentially a time series dataset; so, machine learning and statistical methods can be applied to it.

The machine learning techniques utilized for the research are Artificial Neural Networks (ANNs) and Support Vector Regression (SVR). The results of the forecasting using ANNs was compared to SVR; which was then validated against the forecast obtained from an ARIMA method.

It was discovered that for series with no trend and seasonality and only irregular and/or cyclical behaviour, the SVR Gaussian model is the clear performer in 92% of the product series. The remaining 8% have the ANN traincgf algorithm as the model with the smallest MAD on the overall dataset. For series with slight trend and no seasonality, the Gaussian SVR kernel outperforms the other kernels for 85% of the products. The ANN trainlm algorithm performs the best for 9% of the products, followed by the SVR polynomial kernel for the remaining 6%. For series with clear/heavy trend and no seasonality, the ANN trainbfg algorithm outperforms the other training algorithms

Acknowledgements

I would like to extend my heartfelt gratitude towards my family and friends for giving me the support required to complete this research report.

I would also like to thank my supervisor, Dr Joke Bührmann for her support and encouragement during this period.

Contents

List of figures

List of Tables

Nomenclature/list of acronyms

ACF	Autocorrelation Function
ANN	Artificial Neural Network
AR	Autoregressive
ARIMA	Autoregressive Integrated Moving Average
ARMA	Autoregressive Moving Average
BFGS	BFGS Quasi Newton Backpropagation
CGF	Conjugate Gradient Backpropagation with Fletcher Reeves restarts
CGP	Conjugate Gradient Backpropagation with Polak/Ribiere restarts
CGB	Conjugate Gradient with Powell/Beale restarts
GD	Gradient Descent Backpropagation
GDA	Gradient Descent with Adaptive learning rate backpropagation
GDM	Gradient Descent with Momentum Backpropagation
GDX	Gradient Descent with momentum and adaptive learning rate backpropagation
LM	Levenberg Marquardt Backpropagation
MA	Moving Average
MAD	Mean Absolute Deviation
MAPE	Mean Average Percentage Error
MLP	Multi Layer Perceptron
MSE	Mean Squared Error
OSS	One Step Secant Backpropagation
PACF	Partial Autocorrelation Function
QPP	Quadratic Programming Problem
RBF	Radial Basis Function
RNN	Recurrent Neural Network
RP	Resilient backpropagation
SCG	Scaled conjugate gradient backpropagation
SLP	Single Layer Perceptron
SRM	Structural Risk Minimization
SVM	Support Vector Machine
SVR	Support Vector Regression

1. Introduction

1.1 Research background and context

A supply chain consists of the storage, dispatch and transportation of goods from source to client (Stefanovic, 2015). An efficiently run supply chain is the backbone of a well-organized and effective business. Figure 1 below shows the supply chain in greater detail from manufacturer to customer.

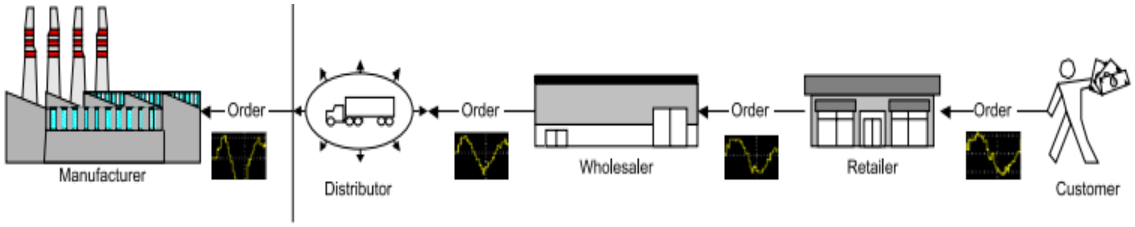


Figure 1: Supply chain (Carbonneau, et al., 2007)

From Figure 1, it can be seen that information from the customer is used to order products from the retailer. Orders from the retailer are fed back to the wholesaler. The wholesaler derives its orders from the manufacturer through a distribution network. Customer demand is what drives the supply chain from manufacturer to customer. The knowledge of what goods need to be produced and by what amounts is what propels the manufacturing process.

Demand forecasting is the field of study that aims to predict customer demand; which refers to the customer requirement for commodities or services (Shahrabi, et al., 2009). Insufficient goods in the stores will lead to financial loss for the organization. On the other hand, having surplus goods on hand will lead to space constraints in the warehouses and financial loss depending on the shelf life of the goods on hand (Stefanovic, 2015). It is thus essential to accurately predict customer demand patterns.

Demand forecasting falls under the broad genre of time series forecasting. Time series forecasting uses the historical patterns of the time series to generate future forecasts (Adhikari & Agrawal, 2013). Traditionally, demand forecasting was achieved using stochastic methods. These traditional stochastic/statistical methods are subject to limitations explained in Section 1.2 below.

Over time, there has been a drive to investigate new methods in the field of time series forecasting in order to overcome the challenges faced by traditional methods. One of the applications of machine learning focuses on the same problem as time series: to predict a

new outcome based on previously known outcomes (Bourne, 2016). Thus, machine learning methods can be applied to time series forecasting.

Historically, machine learning was born out of a need to derive patterns from data (SAS, 2018). In other words, machine learning is a branch of computer science in which algorithms learn from data (Bourne, 2016). These algorithms are constantly updated through iterations using input data and parameters. The patterns developed from these iterations can be used for various applications.

Some of commonly used machine learning methods are artificial neural networks, support vector machines, decision trees and Bayesian networks. Each algorithm has trade-offs in speed, effectiveness and accuracy of the results (Bourne, 2016).

This research project aims to utilize machine learning techniques to investigate the forecasting of inventory stock. The dataset used is Kaggle's "Historical Product Demand" dataset. The dataset consists of 2172 product categories arranged in monthly logs from 2011 to 2017 (Zhao, 2017). A traditional stochastic method will be used to benchmark the forecasts obtained from the machine learning techniques.

1.2 Research Motivation

Demand management and planning has been an essential part of supply chain and logistics from historical times (Nguyen, et al., 2017). Under economic stresses like war, recession etc., supply disruptions were common and thus lead to the necessity for forecasting and managing inventory levels. Even without unforeseen circumstances, there lies a need to manage customer demand without leading to overstocking or understocking (Stefanovic, 2015). Understocking leads to an inability to meet customer demand; which in turn leads to monetary loss (Stefanovic, 2015; Davis & Heineke, 2005). Overstocking leads to an increase in cost and a lack of space (Stefanovic, 2015).

An organization's position in the competitive space is greatly improved by properly understanding customer demand. Effective demand forecasting helps industries know what goods to have in stock and what quantities are required (Mupparaju, et al., 2018).

A well-managed inventory affects the performance of the whole supply chain positively. The ability to predict customer demand results in improved production planning, material planning and inventory space management (Kandananond, 2012). An adequately stocked

inventory improves customer satisfaction and enables management to properly respond to gaps in supply chain (Zhou, et al., 2015). At the same time, meeting the demands of the customer timeously can improve customer loyalty and increase savings for the long term. Demand forecasting for maintaining an optimal inventory can be applied in the manufacturing sector, automotive sector, as part of a large retail chain, or to any other applicable sector.

Traditionally, demand forecasting was achieved using statistical methods like moving-averages, same-time-period, multiple linear regression and the Holt Winters method (Athanasopoulos & Hyndman, 2018; Kotu & Deshpande, 2015). Studies have been performed in the past that show the accuracy and efficiency of traditional forecasts; thus, establishing them as effective forecasting tools. However, statistical methods are not able to capture the complex nature of the data. Some of the traditional techniques like moving-averages are elementary and are less capable of intelligent forecasting (Zhou, et al., 2015). Studies in the past have shown that simple forecasting methods can induce the bullwhip effect (Carbonneau, et al., 2007). The bullwhip effect is a phenomenon where the demand information is distorted as it moves up the supply chain (Lee, et al., 1997). Also, the performance of the forecasting algorithm can always be improved upon (Carbonneau, et al., 2007). The performance is evaluated in terms of the error function; which can always be bettered.

Time series forecasting can be performed either one-step-ahead or multi-step ahead. The difficulty of the forecasting task increases with the number of steps into the future that the forecast needs to be made (Athanasopoulos & Hyndman, 2018). This research report aims to forecast one step into the future using the past n time steps.

In order to achieve a good forecast, various factors come into play. Most machine learning techniques are able to incorporate the complex characteristics of time-series data in the forecasts (Zhou, et al., 2015). Using data mining and machine learning, demand forecasting can be achieved using algorithms that learn via trial-and-error instead of set programming rules (Makridakis et al, 2018). In a modern, dynamic supply chain landscape; machine learning tools can produce results without compromising on speed and effectiveness (Stefanovic, 2015). From the various machine learning techniques, artificial neural networks (ANNs) and support vector machines (SVMs) have shown to have great ability in modelling data patterns not usually captured by traditional statistical methods (Carbonneau, et al., 2007). These machine learning methods have been used in various fields like computer science, electrical engineering, sales and finance.

1.3 Problem Statement

The problem addressed in this study is: how to achieve forecasting of inventory stock using machine learning methods. Two different machine learning techniques will be investigated, namely, SVMs and ANNs. The forecasts produced will be validated against a traditional forecasting method, namely, ARIMA. The dataset to be utilized for this study is Kaggle's "Historical Product Demand"; which contains inventory stock data of various product categories over a time period from 2011 to 2017 (Zhao, 2017). It consists of 2172 product categories with data in monthly logs before processing. Since it is a public dataset, the data is sometimes sporadic and not filled entirely over this time period. Therefore; further processing is required before the dataset can be utilized.

1.4 Research questions

The critical research question can be summarized as follows:

How do machine learning methods perform when utilized to forecast inventory stock compared to traditional methods?

The objectives of this research are to:

- determine which method of machine learning provides more accurate forecast of inventory stock. The methods considered will be variations of ANNs and SVMs.
- quantify and compare the performance of the machine learning methods against ARIMA method for benchmarking

1.5 Summary of Research Method

The research method is a comparative study of the performance of SVMs versus ANNs in the field of time series forecasting. Various error measures will be used to measure the performance of the two machine learning forecasting methods as well as the ARIMA method used for benchmarking.

1.6 Limitations of Scope

The results are limited to the experiments conducted to Kaggle's Historical Product Demand dataset. This is a limited dataset with only 60 time points for each product demand category. The results might differ depending on the size of the dataset used. The reader is encouraged to apply the forecasting techniques to bigger datasets to explore their efficiency in a wider context.

1.7 Outline of Chapters

The remainder of this research can be summarised as follows. Chapter 2 contains a literature review of time series, classifications of time series and its components while Chapter 3 discusses traditional forecasting methods used in industry and academia. In Chapter 4, the background of ANNs and SVMs are explored. A brief introduction to classification and regression is also provided. Chapter 5 explores various data pre-processing methods commonly used in time series analysis while Chapter 6 discusses performance measures commonly used to measure error. Based on these findings, Chapter 7 focuses on the methodology behind the investigation. It also describes the data, research methods and the tools used in the investigation. In Chapter 8, the ANN, SVM and ARIMA methods are analysed and compared. The final chapter, Chapter 9, provides conclusions and identifies future areas of research.

Appendices A – D address the MATLAB code for the ANN, SVM and ARIMA methods.

2. Time series

The “Historical Product demand” dataset is essentially a time series i.e. it consists of monthly data points from January 2012 to December 2016. Thus; the following sections will explore the overview of time series, classifications of time series, time series analysis and components of time series. It will also explore the correlation and autocorrelation aspects of time series.

2.1 Time series overview

A time series consists of a collection of data points measured periodically (Adhikari & Agrawal, 2013). The basic format of a time series is *x(t),* t=0,1,2… where t is the time at which the data is measured. A series in which some of the time points are empty is not considered time series. Figure 2 shows an example of a time series.

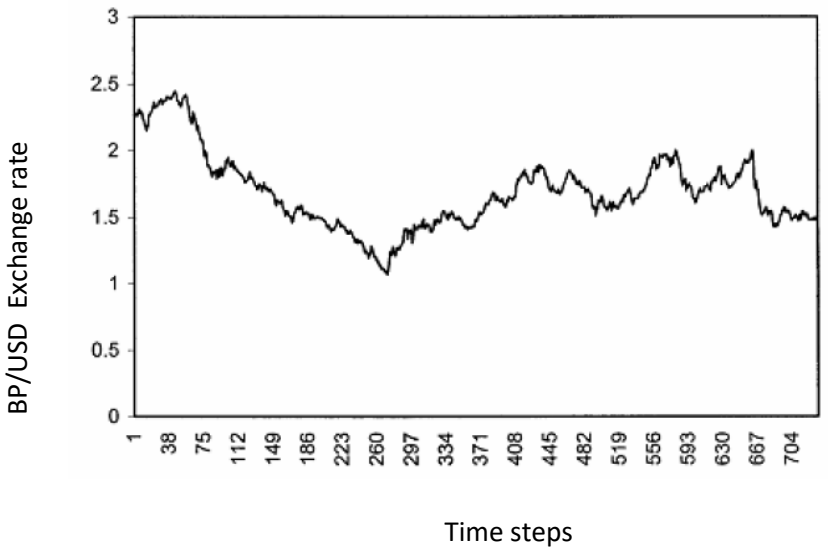


Figure 2 : Weekly BP/USD exchange rate (Adhikari & Agrawal, 2013)

In Figure 2, a time series of the weekly British Pound (BP) to US Dollar (USD) exchange rate is displayed. The X-axis displays the number of time steps while the Y-axis displays the exchange rate.

2.1.1 Time series classifications

Different criteria are used to classify time series. These are usually the number of variables, continuity of measurement, length of time step, stationarity and memory.

Using the criteria of number of variables, time series can either be

- univariate

- multivariate (Adhikari & Agrawal, 2013; Box, et al., 2015).

A univariate time series consists of one variable whereas a multivariate time series contains more than one variable (Adhikari & Agrawal, 2013).

Another way of distinguishing time series is by continuity of measurement, by which it is classified into:

- discrete time series
- continuous time series

A discrete time series consists of data measured periodically (Athanasopoulos & Hyndman, 2018). Thus, the time series will consist of discrete time steps. For example, monthly production of a factory is a time series in which data is measured and recorded periodically. A continuous time series consists of data measured continuously. For example, flow of a river, electrical signals are time series which are measured continuously (Box, et al., 2015).

Using the distance between the time steps, time series can be classified into:

- equidistant time series
- non-equidistant time series

As the name suggests, equidistant time series are series in which data is measured and recorded at constant time steps (Kotu & Deshpande, 2015). On the other hand, in non-equidistant time series, data is measured and recorded at differing time steps.

Based on stationarity, time series can be classified into:

- stationary time series
- non-stationary time series

Series in which statistical properties like mean and variance do not change over time are known as stationary time series (Adhikari & Agrawal, 2013; Box, et al., 2015; Athanasopoulos & Hyndman, 2018). On the other hand, in non-stationary time series, statistical properties change over time. Time series commonly used in industry belong to the non-stationary category. According to Box, et al. (2015) and Athanasopoulos & Hyndman (2018); non-stationarity in time series is normally caused by the presence of trend and seasonal components in the time series.

2.1.2 Components of time series

A time series consists of four components (Adhikari & Agrawal, 2013; Athanasopoulos & Hyndman, 2018):

- Trend $T(t)$,
- Cyclical $C(t)$,
- Irregular I (t) and
- Seasonal components $S(t)$.

Trend refers to the long-term movement of the time series, for example, increasing trend of population growth (Kotu & Deshpande, 2015). Seasonality refers to the seasonal variations within a time series such as time of the month or day of the week (Adhikari & Agrawal, 2013). The cyclical variation in time series refers to the medium-term variations in the series. Irregular component of time series is caused by random and unpredictable influences, for instance, war and recession (Adhikari & Agrawal, 2013; Box, et al., 2015). Figure 3 shows the components of time series in greater detail.

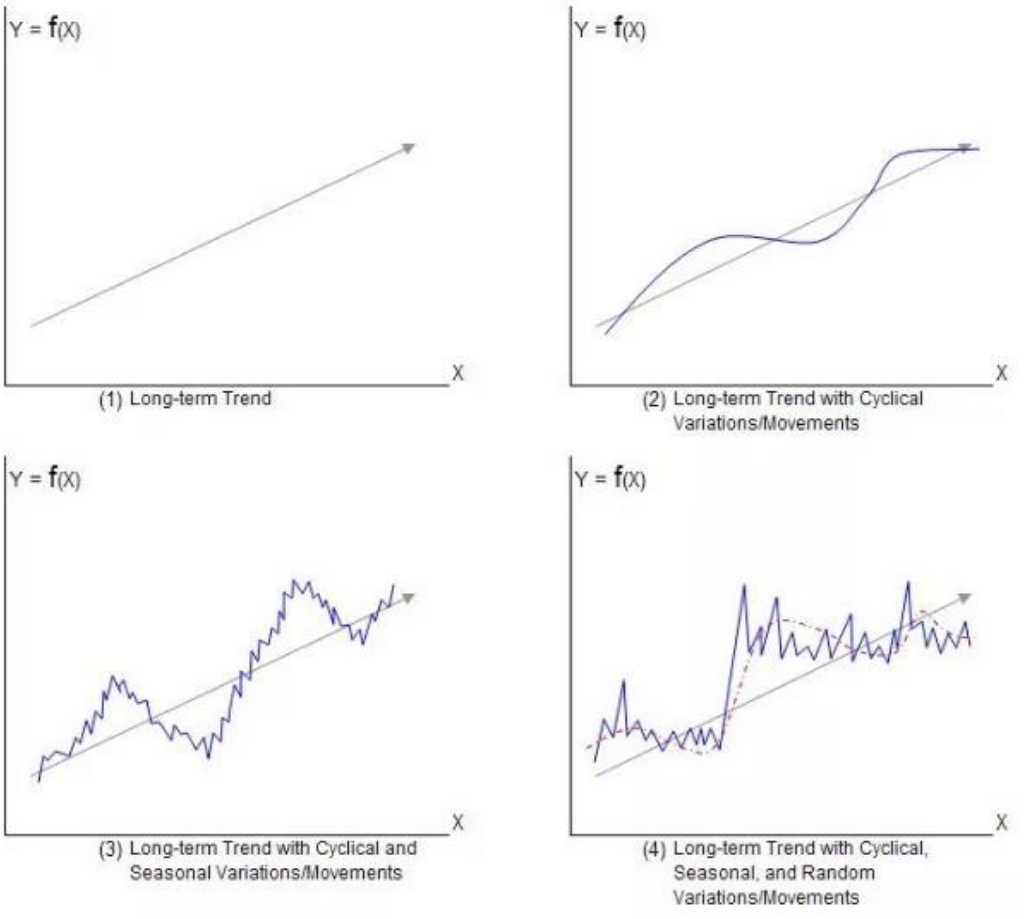


Figure 3 : Components of time series (Ullah, 2014)

In Figure 3 (1), long term trend is displayed while Figure 3 (2) displays long term trend with cyclical variations. Seasonal variation along with trend and cyclicity is displayed in Figure 3 (3) while Figure 3 (4) displays random variations along with the other components of the series.

It is important to make the distinction between seasonality and cyclicity. According to Box, et al. (2015), seasonality in time series displays repetitive behaviour at fixed intervals. Cyclical behaviour is displayed by irregular intervals and are not a fixed length.

There are two main ways in which these four components associate with each other:

- Additive model:

$$Y(t) = T(t) + S(t) + C(t) + I(t) \quad (1)$$

  In this model, the four components are independent of each other (Box, et al., 2015; Athanasopoulos & Hyndman, 2018; Adhikari & Agrawal, 2013)

- Multiplicative model:

$$Y(t) = T(t) \times S(t) \times C(t) \times I(t) \quad (2)$$

  In this model the four components depend on each other (Box, et al., 2015; Athanasopoulos & Hyndman, 2018; Adhikari & Agrawal, 2013)

2.1.3 Time series analysis

Time series analysis is the process of building a model from a time series and determining its characteristics (Kotu & Deshpande, 2015). One subset of time series analysis is time series forecasting. It is the process of predicting the future based on past events of a time series. In other words, a series' past value may be used to predict the future values (Mellit, et al., 2012).

Forecasting can either be one-step ahead or multi-step ahead (Athanasopoulos & Hyndman, 2018). In one-step ahead forecasting, past values are used to forecast just one step into the future whereas multistep forecasting can be used to predict several steps into the future (Shumway & Stoffer, 2010).

Time series forecasting can be performed in two ways. The first one is where the only the past values of the current series are used to predict future values. The second method is where external factors/other time series are used to predict into the future (Kotu & Deshpande, 2015).

2.2 Correlation

Correlation coefficients are used to determine the relationship between two variables (Athanasopoulos & Hyndman, 2018).

The correlation coefficient between two variables x and y is determined by

$$r = \frac{\sum(x_t - \overline{x})(y_t - \overline{y})}{\sqrt{\sum(x_t - \overline{x})^2}\sqrt{\sum(y_t - \overline{y})^2}} \quad (3)$$

where x_t = x_1, x_2, x_3…. , y_t = y_1, y_2, y_3…., $\overline{x}$ is the average of x_t and $\overline{y}$ is the average of y_t (Athanasopoulos & Hyndman, 2018).

Positive values of *r* are indicative of a positive relationship while negative values indicate a negative relationship between the two variables.

2.3 Autocorrelation and partial autocorrelation

Autocorrelation and partial autocorrelations refer to the relationships between time series and its historical values (Adhikari & Agrawal, 2013; Athanasopoulos & Hyndman, 2018). These historical time points are also known as lags. Using the autocorrelations and partial autocorrelations of a time series enables the forecaster to utilize the correct number of lags for time series forecasting model.

2.3.1 Autocorrelation

An autocorrelation function (ACF) is the correlation between a time series at present and its historical values (Adhikari & Agrawal, 2013; Athanasopoulos & Hyndman, 2018). An autocorrelation function is important to detect dependencies between time series components (Box, et al., 2015).

For a given time series, there are several autocorrelation coefficients r_k; which are described by:

$$r_k = \frac{\sum_{t=k+1}^{T}(y_t - \overline{y})(y_{t-k} - \overline{y})}{\sum_{t=1}^{T}(y_t - \overline{y})^2} \quad (4)$$

where y_t = y_1, y_2, y_3…. are the time series values, $\overline{y}$ is the average of y_t and T is the length of the time series (Athanasopoulos & Hyndman, 2018).

Here, r_1 would depict the relationship between y_t and y_{t-1}, r_2 would depict the relationship between y_t and y_{t-2} and so on (Shumway & Stoffer, 2010).

The figure of the plotted autocorrelation functions is known as a correlogram.

Figure 4 below depicts an example of a correlogram.

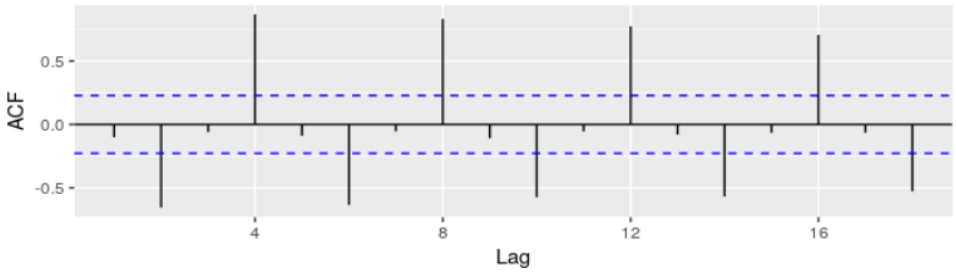


Figure 4: Sample ACF plot (Athanasopoulos & Hyndman, 2018)

In Figure 4, the spikes at various time lags depict the autocorrelation coefficients r_1, r_2, r_3 etc..

2.3.2 Partial autocorrelation

A partial autocorrelation is the relationship between the time series and its historical values without the influence of the time series values in between (Shumway & Stoffer, 2010; Athanasopoulos & Hyndman, 2018). For instance, a partial autocorrelation can demonstrate the dependence of a first value on the third value of a time series without the effect of the second value.

A time series' underlying trend, seasonality, cyclical and irregular behaviour can be determined from its ACF and PACF plots. A time series with seasonality will exhibit repeating behaviour in fixed time steps in its ACF plot. A gradual increase/decrease in ACF is an indicator of trend.

Figure 5 displays the ACF of a time series with trend and seasonality.

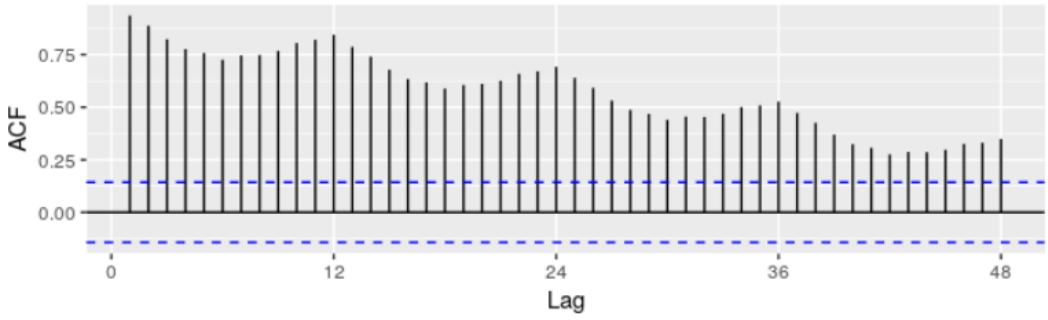


Figure 5: Sample ACF plot with trend and seasonality (Athanasopoulos & Hyndman, 2018)

In Figure 5, the time series displays repeating behaviour every 12 lags; this is an indication of seasonality. The curved shape is also due to seasonality. Trend is also present due to the gradual decrease in the ACF as the number of lags increases (Athanasopoulos & Hyndman, 2018).

3. Traditional Forecasting Methods

Since predicting the future is an important aspect of business, care needs to be applied in the proper selection and design of a forecasting method (Wei, 2006). Traditional forecasting methods can be broadly classified into qualitative/judgmental methods, causal relationship forecasting methods and time series methods (Davis & Heineke, 2005).

3.1 Qualitative/judgemental methods

Qualitative methods are used when there is no data available, and are usually based on opinion or guesstimates. Some of the popular qualitative methods are the Delphi method, market research and historical analogy (Davis & Heineke, 2005; Shumway & Stoffer, 2010).

3.2 Causal relationship forecasting methods

This forecasting method is based on the assumption that demand is dependent on external environmental factors (Athanasopoulos & Hyndman, 2018). Some examples of this method are regression analysis, input/output models and leading indicators (Davis & Heineke, 2005).

3.3 Time series methods

Time series methods are based on the premise that the future behaviour of a time series can be predicted from past behaviour (Adhikari & Agrawal, 2013; Wei, 2006; Davis & Heineke, 2005).

They can be further classified into data driven approaches and model-based approaches, which are described in Sections 3.3.1 and 3.3.2 below.

3.3.1 Data driven approaches

Data driven approaches are forecasting tools in which the forecast is based on the previous data-points in the time series (Kotu & Deshpande, 2015).

3.3.1.1 Naïve forecast

Naïve forecast is the simplest forecasting tool in which the forecast one step ahead is estimated by the last data point in the time series (Athanasopoulos & Hyndman, 2018; Kotu & Deshpande, 2015). In other words

$$F_{t+1} = y_t \tag{5}$$

where F_{t+1} is the forecast one step ahead and y_t is the current data point (Kotu & Deshpande, 2015).

3.3.1.2 Simple average

In this forecasting model, the forecast is given by the average of the previous *n* datapoints (Kotu & Deshpande, 2015). In other words

$$F_{t+1} = average(y_t, y_{t-1} \dots \dots y_{t-n}) \tag{6}$$

where F_{t+1} is the forecast one step ahead and $y_t, y_{t-1} \dots \dots y_{t-n}$ are the previous n data points (Kotu & Deshpande, 2015).

3.3.1.3 Moving averages:

Moving average is an expanded form of the simple average. In this model, the forecast is derived from a window consisting of a set of previous data points (Kotu & Deshpande, 2015). Suppose a window of 4 is chosen. Then the forecast would consist of the average of the previous four data points. In cases where there is seasonality in data, this model can result in distortions.

According to the moving average forecast, the formula for the value of the time series at time t is described by:

$$\begin{aligned} y_t &= \mu + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j} + \varepsilon_t \\ &= \mu + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots \dots \dots \theta_3 \varepsilon_{t-q} + \varepsilon_t \end{aligned} \tag{7}$$

where ε_t is the random error at time t, θ_j (j=1,2…. p) are the model parameters and *q* is the order of the model (Adhikari and Agrawal, 2013).

3.3.1.4 Weighted moving average

In some cases, in the moving average forecast, more relevance should be given to some data points compared to others (Kotu & Deshpande, 2015). This may be due to seasonality; for instance, in the retail industry certain months will have more consumerism than others.

The forecast will then be given by the equation below:

$$F_{t+1} = \frac{ay_t + by_{t-1} \dots \dots + fy_{t-n}}{a + b + \cdots .. f} \quad (8)$$

where typically *a>b> …f* (Kotu & Deshpande, 2015).

The coefficients *a, b, c… f* are usually based on intelligence obtained from the time series.

3.3.1.5 Exponential smoothing

The methods mentioned above use an amount of historical data. If only recent data is available for forecasting, exponential smoothing is the recommended method (Davis & Heineke, 2005; Kotu & Deshpande, 2015).

Exponential smoothing uses only the previously forecasted value to predict the succeeding value. In other words:

$$F_{t+1} = \alpha y_t + (\alpha - 1)F_t \quad (9)$$

where α is a constant between 0 and 1, F_t is the previously forecasted value and y_t is the previous data point (Kotu & Deshpande, 2015).

The exponential smoothing constant α is generally chosen to be small if the demand is stable over time. If the demand fluctuates wildly over time, the smoothing constant is chosen to be larger, i.e. closer to one (Adhikari & Agrawal, 2013; Davis & Heineke, 2005; Wei, 2006).

The exponential smoothing technique is the basis of many other data driven techniques and is one of the most commonly used forecasting techniques.

Using exponential smoothing, it is; however; not possible to make forecasts several steps ahead, due to the fact that the model relies on the forecast at the previous time step (Kotu & Deshpande, 2015). Additionally, simple exponential smoothing is not very efficient in capturing trend and seasonality. In order to make more sophisticated forecasts that incorporate trend and seasonality, Holts two parameter exponential smoothing and Holts three parameter exponential smoothing were developed (Kotu & Deshpande, 2015).

3.3.1.6 Holts two parameter exponential smoothing

The Holts two parameter exponential smoothing technique is highly efficient in capturing trends due to the incorporation of two parameters α and β into the formula for the model (Kotu & Deshpande, 2015). Simple exponential smoothing only provides an expression for the average or "level" of the series (Shumway & Stoffer, 2010). Holts two parameter exponential smoothing provides expressions for the level and trend of the series.

The forecast is expressed as a sum of the trend T_t and average value L_t. In other words,

$$F_{t+1} = T_t + L_t \tag{10}$$

where

$$L_t = \alpha * y_t + (1 - \alpha) * (L_{t-1} + T_{t-1}) \tag{11}$$

$$T_t = \beta * (L_t - L_{t-1}) + (1 - \beta) * (T_{t-1}) \tag{12}$$

where α, β are constants, L_t is the level or average of the series, T_t is the trend of the time series and y_t is the previous data point (Kotu & Deshpande, 2015).

3.3.1.7 Holt Winters three parameter exponential smoothing

For the Holt Winters three parameter exponential smoothing technique, there is an additional parameter γ to account for seasonality in the time series (Adhikari & Agrawal, 2013; Kotu & Deshpande, 2015). The mathematical expressions incorporate the parameter γ into them.

$$F_{t+1} = T_t + L_t + S_t \tag{13}$$

where

$$L_t = \alpha * (y_t - S_{t-m}) + (1 - \alpha) * (L_{t-1} + T_{t-1}) \tag{14}$$

$$T_t = \beta * (L_t - L_{t-1}) + (1 - \beta) * (T_{t-1}) \tag{15}$$

$$S_t = \gamma * (y_t - L_{t-1} - T_{t-1}) + (1 - \gamma) * (S_{t-m}) \tag{16}$$

Here, α, β and γ are parameters associated with the level, trend and seasonality of the series (Kotu & Deshpande, 2015).

3.3.2 Model driven approaches

Model driven approaches are forecasting techniques in which the forecast can be made several steps into the future (Kotu & Deshpande, 2015). The forecast is not entirely dependent on the previous data point in the series. The model is based on the global pattern of the time series and can be made at any time point into the future (Kotu & Deshpande, 2015; Shumway & Stoffer, 2010).

3.3.2.1 Linear regression

This is the simplest form of the model driven techniques where the forecast is attempted using a linear fit or straight line (Athanasopoulos & Hyndman, 2018). The disadvantage of this approach is that it does not capture the time series effectively (Kotu & Deshpande, 2015). This can be overcome using a more complex model like logistic regression. Figure 6 shows an example of linear regression.

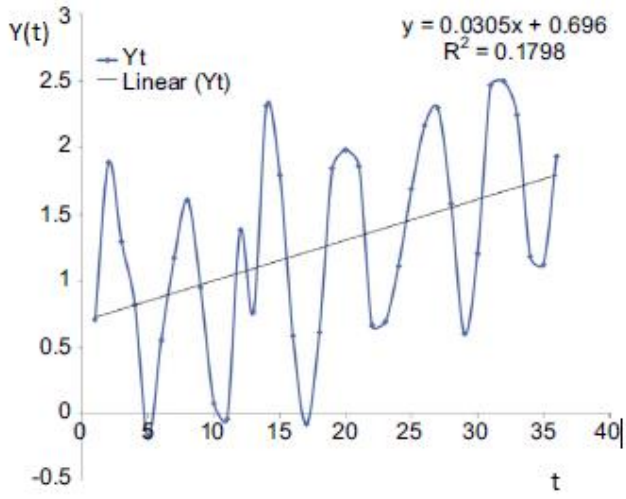


Figure 6 : Example of linear regression model (Kotu & Deshpande, 2015)

In Figure 6, the x-axis represents the number of time steps *t* while the y-axis represents the value of *Y(t).*The time series function is described by $Y(t) = 0.0305\,x + 0.696$ while the straight line represents the linear regression model.

3.3.2.2 Polynomial regression

In this regression model, the forecast is attempted by the utilization of a higher polynomial order fit (Kotu & Deshpande, 2015). This model does a better attempt at forecasting, but is considered inefficient as it still does not capture the time series perfectly (Kotu & Deshpande, 2015). Figure 7 shows an example of a polynomial regression model.

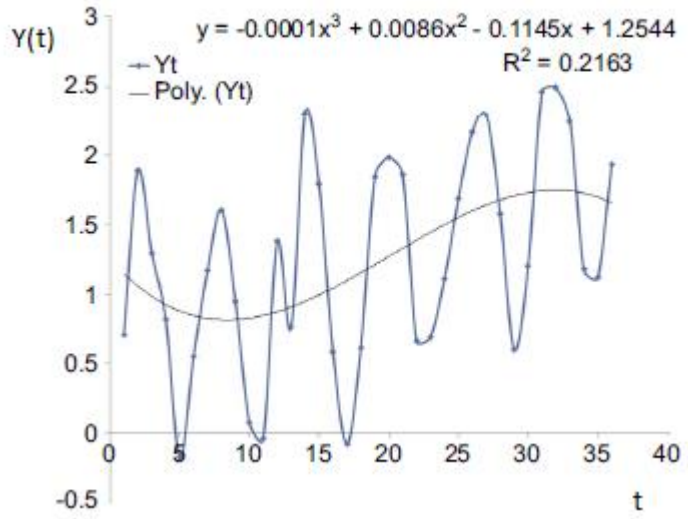


Figure 7 : Polynomial regression model (Kotu & Deshpande, 2015)

In Figure 7, the x-axis represents the number of time steps *t* while the y-axis represents the value of *Y(t).* The time series function is described by $Y(t) = -0.0001x^3 + 0.0086x^2 - 0.1145x + 1.2544$ while the curved line represents the polynomial regression model.

3.3.2.3 Linear regression with seasonality

The linear regression can be improved by introducing variables that capture seasonality. In this instance, the data points that do not influence the fit can be excluded by using dummy variables i.e. a 0 or a 1 (Athanasopoulos & Hyndman, 2018). This improves the fit of the linear regression model (Kotu & Deshpande, 2015).

3.3.2.4 Autoregressive Model

Autoregressive (AR) models are regression models where the prediction is based on the lag series of the original series (Kotu & Deshpande, 2015). The forecast is calculated from the *p* past observations as well as an error and a constant term (Adhikari & Agrawal, 2013). It can be expressed as:

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

$$= c + \varphi_1 y_{t-1} + \varphi_2 y_{t-2} + \cdots \ldots \varphi_p y_{t-p} + \varepsilon_t \qquad (17)$$

Here $\varphi_i (i = 1 \dots p)$ are the model parameters, *c* is a constant and ε_t is the error at t (Adhikari & Agrawal, 2013).

This is known as an AR(p) model, an autoregressive model of order p (Athanasopoulos & Hyndman, 2018).

3.3.2.5 Autoregressive Moving Average

The autoregressive moving average (ARMA) is a combination of autoregressive and moving average models (Adhikari & Agrawal, 2013).

Mathematically, it can be expressed as:

$$y_t = c + \varepsilon_t + \sum_{i=1}^{p} \varphi_i y_{t-i} + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j} \quad (18)$$

Here $\varphi_i (i = 1 \dots p)$ are the parameters of the autoregressive model, $\theta_j (j = 1 \dots q)$ are the parameters of the moving average model, *c* is a constant and ε_t is the error at t (Adhikari & Agrawal, 2013).

The ARMA method can only be applied to stationary processes, which is considered a disadvantage of this method.

3.3.2.6 Autoregressive Integrated Moving Average

As mentioned above, one of the major disadvantages of the ARMA model is that it can only be applied to stationary time series data (Adhikari & Agrawal, 2013; Brownlee, 2017). In order to accommodate non-stationary time series, the ARIMA was derived from ARMA.

The ARIMA stands for Autoregressive Integrated Moving Average and includes a term I for differencing the non-stationary series (Nau, 2014).

The acronym may be explained as follows:

AR (Autoregressive): A model that contains autoregressive parts of the time series and its lag components (Brownlee, 2017)

I (Integrated): The use of differencing in order to make the time series stationary (Brownlee, 2017)

MA (Moving Average): A model that contains a moving average term applied to lags (Brownlee, 2017)

The ARIMA model may be expressed as:

$$y_t{}' = c + \varphi_1 y'_{t-1} + \cdots + \varphi_p y'_{t-p} + \theta_1 \varepsilon_{t-1} + \cdots . + \theta_q \varepsilon_{t-q} + \varepsilon_t \quad (19)$$

where y_t' is the differenced series, p is the order of the autoregressive part and q is the order of the moving average part (Brownlee, 2017; Nau, 2014).

This model is known as an ARIMA (p, d, q) model where

p= order of autoregressive part which is the number of lag terms in the model

d= degree of differencing which is the number of times it is differenced in order to make it stationary

q=order of moving average part which is the size of the moving average window

The values of p and q are usually determined from the ACF and PACF plots of the time series. The value of p is determined from the PACF plot as it contains the correlations between time series components without the influences of the components in between. It is usually taken as the number of lags before the series decays to zero (Athanasopoulos & Hyndman, 2018). The value of q is usually determined from the ACF plots; it is usually the number of lags before the ACF plot decays to zero (Athanasopoulos & Hyndman, 2018)

The ARIMA model was chosen to benchmark the results from the machine learning models due to the following reasons:

- It is a complex model and is therefore able to provide a comparable forecast to the forecasts produced by complex methods like artificial neural networks and support vector machines.
- It is a lag model which is dependent on the lags of the time series. Thus, by choosing the same number of lags for the ANN and SVM models, an equal axis of comparison can be produced for all three models.

4. Machine Learning

4.1 Classification and regression

Machine learning problems can be broadly classified into either classification or regression problems.

Classification problems may be defined as follows: for a pattern x, which is an element of a domain χ, estimate what value a related value y will assume (Smola & Scholkopf, 2004; Smola & Vishwanathan, 2008). For instance, given a person's health record, eating habits, exercise habits, smoking and drinking habits, classification may be used to predict whether or not he/she is likely to develop cancer (Smola & Vishwanathan, 2008). Classification may be also be used to detect whether or not an email is spam. The overall applications of classification are broad and these are just a few examples of what may be achieved with classification.

Classification can either be binary or multiclass (Smola & Vishwanathan, 2008). Binary classification involves classifying into one of two classes whereas multiclass classification involves the use of more than two classes (Smola & Vishwanathan, 2008). Figure 8 displays binary and multiclass classification pictorially.

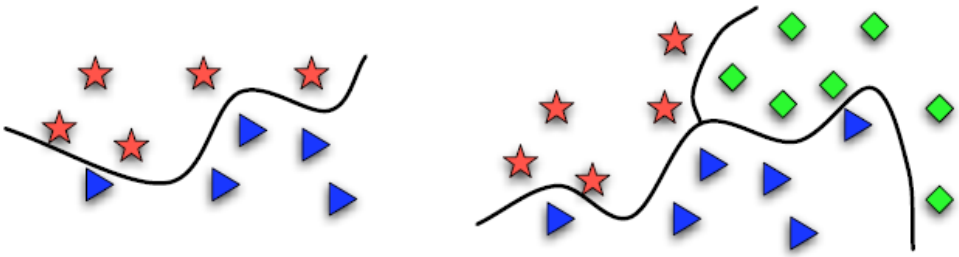


Figure 8 : Left: Binary classification and Right: Multiclass classification (Smola & Vishwanathan, 2008)

In Figure 8, the left diagram displays classification into two classes whilst the right diagram displays classification into three classes.

Regression problems focus on the estimation of a real variable $y \in \mathbb{R}$ from a pattern x. Practical examples of regression include the estimation of stock price for the following day and time series forecasting (Kumar, 2012).

Figure 9 shows regression in greater detail.

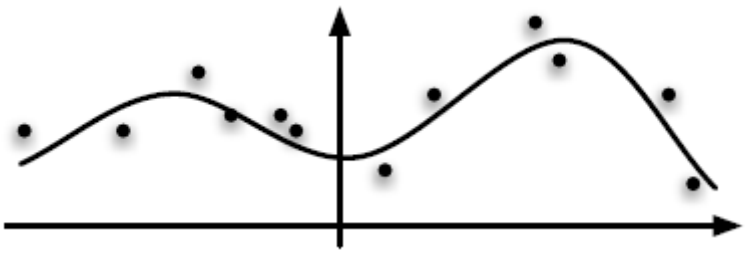


Figure 9 : Regression estimation (Smola & Vishwanathan, 2008)

In Figure 9, the black dots represent the observed values and the aim is to find a function f(x) so that the f(x) is close to the observed values (Smola & Scholkopf, 2004; Smola & Vishwanathan, 2008). Regression also requires a loss function to be defined so that the error between the actual and obtained value may be quantified.

4.2 Underfitting and overfitting for regression problems

Figure 10 depicts underfitting and overfitting for regression problems in machine learning.

Figure 10: Sample underfitting and overfitting (Tran, 2017)

In Figure 10, the left diagram depicts underfitting, the middle diagram depicts a fit just right and the right diagram depicts overfitting.

Underfitting is a phenomenon in which the regression line does not capture the data points of the time series at all (Tran, 2017). Instead, the regression line passes through the data without attempting to capture the data. Underfitting occurs when the model fails to capture the underlying trend of the data (Smola & Vishwanathan, 2008; Tran, 2017).

Overfitting occurs when the regression line captures the data overly. Instead of making a reasonable capture of the data, it tends to try and pass through each data point. This occurs when the model captures the noise of the data instead of the actual data (Tran, 2017).

Underfitting and overfitting can also be explained in terms of the bias-variance trade-off. The bias of a model is defined as the measure of how much the estimated values deviate from the actual values (Smola & Vishwanathan, 2008; Tran, 2017). The variance is defined as the error occurring due to sensitivities from deviations in the training dataset (Tran, 2017).

Generally, underfitted models have low variance but high bias whereas overfitted models have high variance but low bias (Tran, 2017).

For machine learning problems, overfitting and underfitting are countered by the use of cross validation (Beale, et al., 2014; Smola & Scholkopf, 2004). Cross validation is a technique by which the dataset is split into training, validation and test datasets (Beale, et al., 2014). Different configurations are built using the training dataset and are tested on the validation set. The model with the smallest validation error is chosen for the final model. Cross validation for machine learning problems is explained further in Section 4.3.7 and 4.4.7.

4.3 Artificial Neural Networks

Artificial neural networks (ANNs) can be used for both classification and regression tasks.

The advantage of ANNs over traditional forecasting methods is that they are data-driven and self-regulatory (Hill, et al., 1996). They are capable of adapting themselves according to the data that is presented to them. Another advantage of ANNs is that they are inherently able to capture the non-linear characteristics of the data; which make them more accurate than linear methods like ARIMA etc (Haykin, 2009). Artificial neural networks are also excellent in approximating functions; which sets them apart from other forecasting methods (Adhikari & Agrawal, 2013). These advantages of ANNs allow them to prevail over other traditional forecasting methods which are subject to linearity, bias and re-estimation (Hill, et al., 1996).

In spite of these advantages of ANNs, selecting an architecture for the artificial neural network is not an easy task. Selecting the best parameters of the ANN can only be achieved through trial and error.

The biological inspiration for an ANN comes from the neurons of the human brain (Haykin, 2009). Like the human brain is capable of learning, an ANN is capable of learning patterns in the data and generalizing the learned patterns on the test dataset (Adhikari & Agrawal, 2013). There are millions of neurons in the human brain; forming the backbone of the nervous system.

A neuron has a branched structure consisting of a dendrite (input), axon (output) and a nucleus (Haykin, 2009). The axons form connections known as synapses with dendrites of surrounding neurons; through which electrical signals are received (Haykin, 2009; Mahanta, 2017). These signals are processed in the cell body and transmitted to other neurons through the axons (Mahanta, 2017). The surrounding neurons may accept/ reject it based on the strength of the signals. A human neuron is capable of learning through the sending and receiving of these signals. Figure 11 shows the structure of the biological neuron in greater detail.

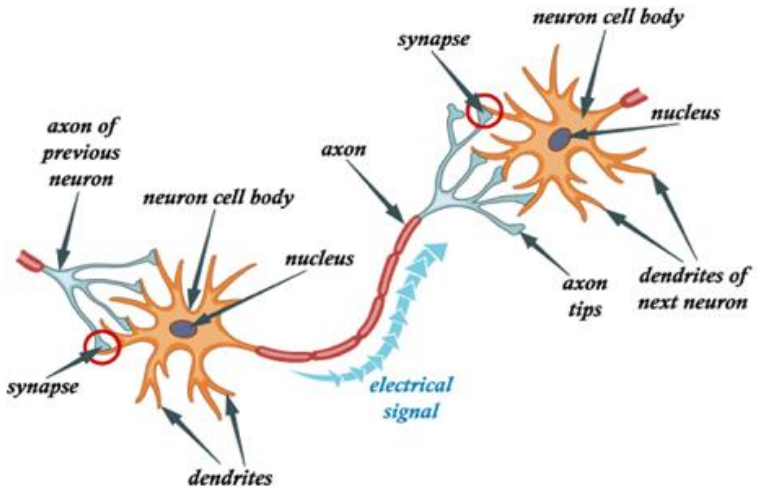


Figure 11 : Biological neuron (Mahanta, 2017)

An ANN is an artificial equivalent of the biological neuron. Figure 12 shows the structure of an ANN in greater detail.

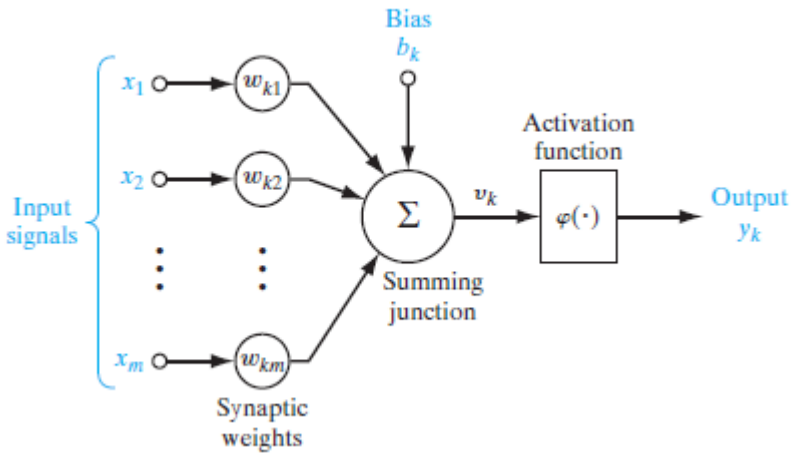


Figure 12: Model of an artificial neural network (Haykin, 2009)

As can be seen in Figure 12, the structure consists of 3 major components:

i) Inputs

The inputs represent the signals received by the ANN. They have synaptic weights associated with them, which they get multiplied by once they enter the ANN (Haykin, 2009). In Figure 12 above, x_1, x_2…. x_m represent the inputs and w_{k1}, w_{k2}…w_{km} represent the synaptic weights.

ii) Summing junction

As the name suggests, the summing junction adds up the weighted inputs and the bias term before sending the signal to the activation block. The bias signal is usually a threshold signal which is added /subtracted to the weighted sum of inputs (Haykin, 2009).

iii) Activation

The activation function generates an output y_k based on the input it receives and the type of activation specified (Haykin, 2009).

The ANN seen in Figure 13 is represented by the equations:

$$v_k = \sum_{i=1}^{m} w_{ki} x_i, \tag{20}$$

$$\text{and } y_k = \varphi(v_k) \tag{21}$$

where x_i represents the inputs, w_{ki} represents the weights, b_k represents the bias, v_k is the activation potential, φ is the activation function and y_k is the output of the ANN (Haykin, 2009).

4.3.1 Types of activation function

The activation function provides the output of the ANN. Different activation functions are available for different types of activations required. The major activation functions which can be utilized are discussed below:

4.3.1.1 Threshold function

This function generates an output of one if the input is greater than zero and an output of zero otherwise (Haykin, 2009). In mathematical terms, it can be expressed as:

$$\varphi(v) = \begin{cases} 1 \; if \; v \geq 0 \\ 0 \; if \; v < 0 \end{cases} \tag{22}$$

Figure 13 below displays the graph of the threshold function.

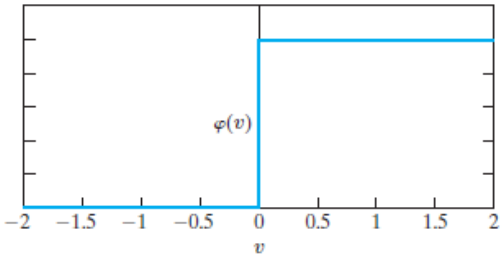


Figure 13: Threshold function (Haykin, 2009)

4.3.1.2 Linear function

The linear function is commonly used for function fitting and generates an output equivalent to the weighted sum of the inputs and biases (Beale, et al., 2014). Mathematically, it can be expressed as:

$$\varphi(v) = v \tag{23}$$

Figure 14 represents the linear function graphically.

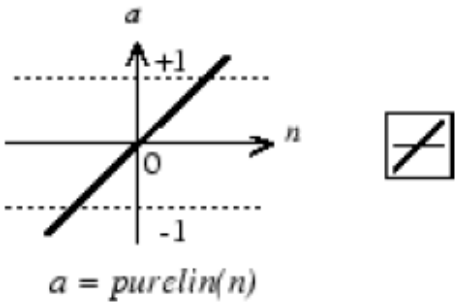


Figure 14 : Linear transfer function (Beale, et al., 2014)

4.3.1.3 Sigmoid function

The Sigmoid function's graph is "S" shaped and it is the most commonly used activation function (Haykin, 2009). The Sigmoid function can either be log-sigmoid or tan-sigmoid.

For the log-sigmoid function, the outputs vary between 0 and 1. The log-sigmoid function is represented by the equation:

$$\varphi(v) = \frac{1}{1 + e^{-av}} \tag{24}$$

where *a* is the slope parameter and *v* is the input (Haykin, 2009).

Figure 15 displays the sigmoid function graphically.

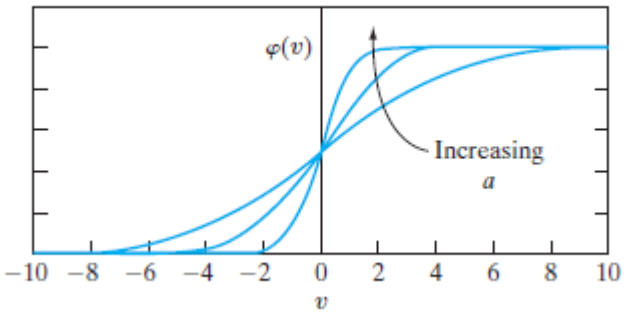


Figure 15 : Log-sigmoid activation function (Haykin, 2009)

Alternatively, networks can utilize the tan-sigmoid transfer function where the outputs vary between -1 and 1. Mathematically, it is expressed as:

$$\varphi(v) = \tanh(v) = \frac{2}{1 + e^{-2v}} - 1 \tag{25}$$

Figure 16 shows the tan-sig function in greater detail.

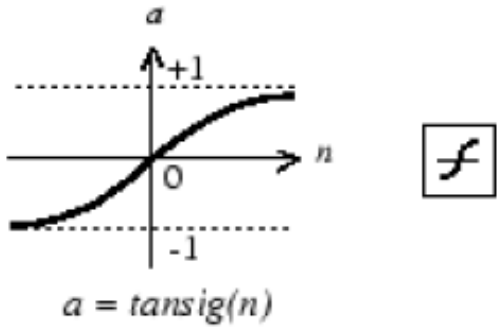


Figure 16: Tan-sigmoid transfer function

4.3.2 Network Topologies

In practise, the individual neurons of an ANN need to be configured in a way that enables learning. Usually, the neurons in an ANN are grouped into layers which are connected together.

A network topology refers to the way in which the ANN is configured; which can be in either of two ways: feedforward and recurrent (Smola & Scholkopf, 2004).

- Feedforward topology

In this type of network topology, the inputs and outputs are connected in a feedforward fashion and there are no feedback loops (Haykin, 2009). Figure 17 displays the feedforward topology in greater detail.

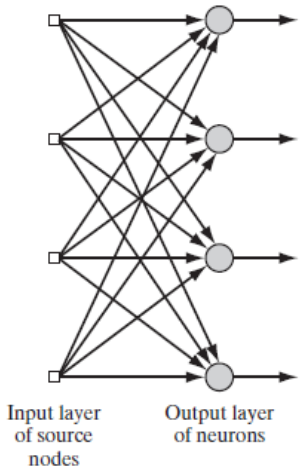


Figure 17: Feedforward neural network (Haykin, 2009)

As can be seen in Figure 17 above, the feedforward network consists of input layer and an output layer of neurons connected in a straightforward manner.

- Recurrent topology

In this type of neural network, the outputs of some of the neurons are fed back into neurons of the same layer or a previous layer (Carbonneau, et al., 2007). In other words, feedback is enabled by the use of a recurrent neural network (RNN). The feedback is possible either from the output of a neuron back to its own input, or to the input of another neuron.

Diagrammatically, an RNN can be represented as shown below in Figure 18. In this figure, the outputs of the hidden layer are used as inputs to the recurrent layer.

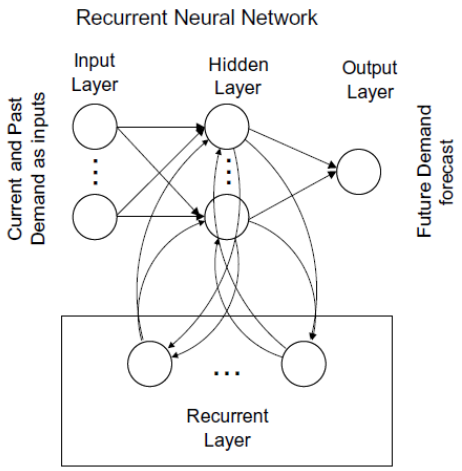


Figure 18: Recurrent neural network (Carbonneau, et al., 2007)

As can be seen in Figure 18, the RNN consists of an input layer, a hidden layer and an output layer of neurons connected with feedback loops.

4.3.3 Perceptron neural network

4.3.3.1 Single Layer Perceptron

These networks typically have a single connection between the inputs and outputs and are connected in a feedforward fashion (Haykin, 2009). Figure 19 displays this in greater detail.

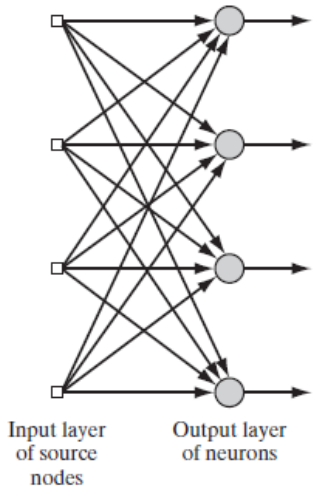


Figure 19 : Single Layer Perceptron (Haykin, 2009)

4.3.3.2 Multi Layer Perceptron

The most commonly used artificial neural network is the Multi Layer Perceptron (MLP) which consists of an input layer, a hidden layer and an output layer of neurons (Adhikari & Agrawal, 2013). In some cases, there may be more than one hidden layer. The hidden layer adds an extra layer of complexity to the neural network.

The input layer provides activation functions to the hidden layer. The outputs of the hidden second layer are used as inputs to the output layer (Adhikari & Agrawal, 2013; Haykin, 2009). The outputs provided by the final layer are seen as overall responses to the inputs provided to the neural network.

The structure of the MLP is shown in Figure 20 below.

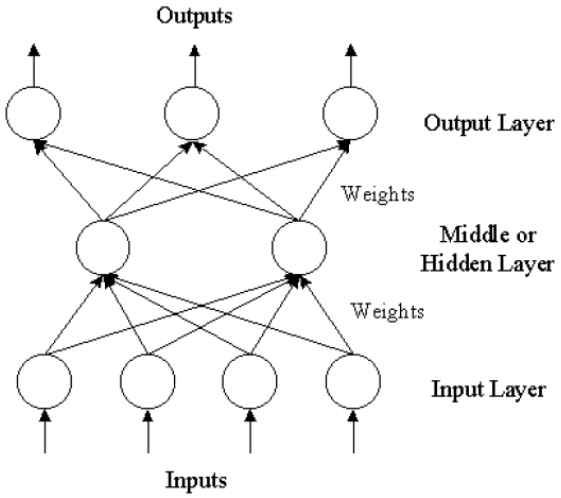


Figure 20: Multilayer perceptron (Adhikari and Agrawal, 2013)

The output of the model is given by the following expression:

$$y_t = \alpha_0 + \sum_{j=1}^{q} \alpha_j \, g\left(\beta_{0j} + \sum_{i=1}^{p} \beta_{ij} \, y_{t-i}\right) + \varepsilon_t \, , \forall t \tag{26}$$

where y_{t-i}, $i = 1,2 \ldots p$ are the inputs,

y_t is the output,

$\alpha_j, j = 1,2 \ldots . q$ and $\beta_{ij}, i = 1,2 \ldots p, j = 1,2 \ldots q$ are the connection weights,

ε_t is the random shock and

α_0 and β_{0j} are the bias terms (Adhikari and Agrawal, 2013).

The activation function for the hidden layer is usually the sigmoid function given by: $g(x) = \frac{1}{1+e^{-x}}$. The activation function for the output layer is usually the linear activation function.

4.3.4 Training of ANNs

The weights and bias of an ANN are adjustable parameters. The processing of changing the weights and bias until the neural network "learns" is known as training (Beale, et al., 2014).

During training, the network is presented with a target value and it aims to learn that target value (Beale, et al., 2014; Haykin, 2009). One iteration of the learning process involves initializing the weights and biases and generating an output (Beale, et al., 2014). The output is compared to the target and the process is repeated if the desired target is not obtained. This method of reinitializing weights and biases is repeated until the desired target is obtained (Haykin, 2009). There are various training algorithms that can be utilized, which are described in Section 4.3.5.

The performance is usually measured in terms of Mean Squared Error (MSE). Either the gradient of the network performance, or the Jacobian of the network errors is used for the optimization process (Beale, et al., 2014). A technique known as backpropagation is usually employed for calculating the gradient or the Jacobian.

4.3.5 Types of training algorithms

Neural networks can be trained using a variety of training algorithms; which use either the gradient method or the Jacobian method. They are described in greater detail in Table 1 below.

Table 1 : Training algorithms and groups associated with them (Comert & Kocamaz, 2017)

Group	Abbreviation	Name	Commonly used for	Size of network
1	GD	Gradient descent backpropagation	Regression/ Classification	Any
1	GDA	Gradient descent with adaptive learning rate backpropagation	Regression/ Classification	Any
1	GDM	Gradient descent with momentum backpropagation	Regression/ Classification	Any
1	GDX	Gradient descent with momentum and adaptive learning rate backpropagation	Regression/ Classification	Any
2	RP	Resilient backpropagation	Classification	Large networks
3	CGF	Conjugate gradient backpropagation with Fletcher Reeves restarts	Regression/ Classification	Any
3	CGP	Conjugate gradient backpropagation with Polak/Ribiere restarts	Regression/ Classification	Any
3	CGB	Conjugate gradient with Beale/Powell restarts	Regression/ Classification	Any
3	SCG	Scaled conjugate gradient backpropagation	Classification	Large networks
4	BFGS	BFGS Quasi Newton backpropagation	Regression/ Classification	Any

4	OSS	One step secant backpropagation	Regression/ Classification	Any
5	LM	Levenberg Marquardt backpropagation	Regression	Small networks

Training of ANNs focuses on the minimization of a loss function ***f***; which consists of an error term and a regularization term. The error term measures how effectively the ANN fits the dataset (Quesada, 2019). The regularization term prevents overfitting by regulating the intricacy of the ANN (Quesada, 2019; Haykin, 2009).

The weights and biases of the ANN can be grouped together into a vector **w.** Thus, the loss function ***f(w)*** is dependent on the weights and biases (Quesada, 2019). Figure 21 depicts the loss function in detail.

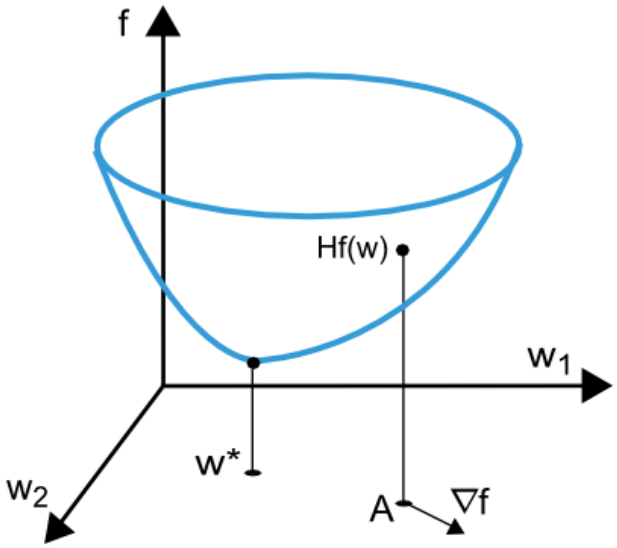


Figure 21: Loss function f(w) (Quesada, 2019)

From Figure 21, the minimum of the loss function can be obtained at w^*. At any point A, the first derivative and second derivates of ***f(w)*** can be calculated (Beale, et al., 2014; Haykin, 2009). The first derivatives are contained in the gradient vector:

$$\nabla \boldsymbol{f}(\boldsymbol{w}) = \frac{\partial f}{\partial w_i}, \tag{27}$$

for *i= 1, 2 … n* (Quesada, 2019).

The second derivatives are contained in the Hessian matrix:

$$H_{i,j}\,\boldsymbol{f}(\boldsymbol{w}) = \frac{\partial^2 f}{\partial w_i \partial w_j} \tag{28}$$

for *i, j= 0, 1,….* (Beale, et al., 2014) .

During training, the minimum w^* of the loss function ***f(w)*** is attempted to be reached. This is achieved by searching through the parameter space and consequently decreasing the loss function through each iteration (Haykin, 2009). At each iteration, the ANN weights and biases will be changed. The training algorithm continues decreasing the loss function until the stopping criterion is reached (Beale, et al., 2014; Quesada, 2019).

From Table 1, there are five groups of training algorithms.

4.3.5.1 Group 1: Gradient descent algorithms

Gradient descent algorithms use the first derivative of the gradient vector to reduce network error as quickly as possible (Comert & Kocamaz, 2017; Quesada, 2019). The standard algorithm is Gradient Descent (GD).

One iteration of the gradient descent algorithm is defined by:

$$w_{k+1} = w_k - a_k g_k, \tag{29}$$

where w_k is a vector of current weights and biases,

a_k is the learning rate,

g_k is the gradient of the error,

w_{k+1} is the new weight vector and

k is the number of iterations (Comert & Kocamaz, 2017).

In order to overcome limitations associated with the learning rate, different variations of the Gradient Descent algorithm; like Gradient descent with adaptive learning rate backpropagation (GDA), Gradient descent with momentum backpropagation (GDM) and Gradient descent with momentum and adaptive learning rate backpropagation (GDX) were implemented (Haykin, 2009; Comert & Kocamaz, 2017).

The Gradient descent with momentum backpropagation algorithm requires its weights, inputs and transfer functions to have derivative functions (Comert & Kocamaz, 2017). The derivatives of weights, inputs and transfer functions are calculated using backpropagation. It uses the following equation for computation:

$$dX = mc * dXprev + lr * (1 - mc) * \frac{dperf}{dX}, \quad (30)$$

where dX is the previous change to weight/bias,

mc is the momentum constant,

and lr is learning rate (Comert & Kocamaz, 2017).

One of the major drawbacks of the GD algorithms is that it requires many iterations and is generally recommended for very large networks (Comert & Kocamaz, 2017; Quesada, 2019).

4.3.5.2 Group 2: Resilient Backpropagation

Resilient backpropagation, also known as Newton's method, uses the Hessian matrix; therefore, it is a second order method. Resilient backpropagation uses sign of the derivative to update weights instead of the magnitude (Comert & Kocamaz, 2017). It significantly improves speed and reduces the number of iterations.

The error function is calculated by

$$\Delta x_k = -sign\left(\frac{\Delta E_k}{\Delta x_k}\right)\Delta k \quad (31)$$

where Δx_k is the difference in the weight vectors,

ΔE_k is the change in error function and

Δk is the change in bias (Comert & Kocamaz, 2017).

4.3.5.3 Group 3: Conjugate Gradient Algorithms (CGAs)

In Conjugate gradient algorithms, the loss function is searched for in conjugate directions (Quesada, 2019). Conjugate gradient algorithms provide better speeds and memory requirements than other methods but are not efficient in large scale problems (Quesada, 2019; Comert & Kocamaz, 2017).

The weight update is performed according the following equations

$$w_{k+1} = xw_k + a_k p \quad (32)$$

$$p_k = -g_k + \beta_k p_{k-1} \quad (33)$$

where w_{k+1}, w_k are the weight vectors, g_k is the gradient, p_k is the negative of the gradient and β_k is a constant known as the conjugate parameter (Comert & Kocamaz, 2017). Two of

the most common ways to calculate it are the Fletcher and Reeves method and Polak Ribiere method.

4.3.5.4 Group 4: Quasi Newton Algorithms

Quasi Newton Algorithms are computationally more extensive than CGAs but have similar fast optimization speeds (Comert & Kocamaz, 2017). The weights are updated according to the Newton method given below:

$$x_{k+1} = x_k - H_k^{-1} g_k \tag{34}$$

where H_k is the Hessian matrix or second derivation of the performance index,

x_{k+1}, x_k are the weight vectors,

and g_k is the gradient (Haykin, 2009).

4.3.5.5 Group 5: Levenberg Marquardt

The Levenberg Marquardt training algorithm is based on the computation of a Jacobian matrix through a standard backpropagation technique (Mathworks, 2019). The performance function is normally the mean/sum of squared errors. The Jacobian matrix contains the first derivations of the network weights and biases (Mathworks, 2019).

The Hessian matrix can be described by

$$H = J^T J, \tag{35}$$

where *J* is the Jacobian matrix (Quesada, 2019).

This training algorithm uses the following approximation of the Hessian matrix:

$$w_{k+1} = w_k - [\boldsymbol{J}^T \boldsymbol{J} + \mu \boldsymbol{I}]^{-1} \boldsymbol{J}^T \boldsymbol{e}, \tag{36}$$

where w_k is a vector of current weights and biases,

J is the Jacobian matrix,

μ is a damping factor ,

w_{k+1} is the new weight vector,

$\boldsymbol{e}$ the vector consisting of all error terms and

I is the Identity matrix (Haykin, 2009; Quesada, 2019).

When the value of μ approaches 0, Equation (36) becomes an approximation of Newton's method. When μ increases, it becomes gradient descent method. The aim is to approximate Newton's method; as it is faster and more accurate (Mathworks, 2019). Therefore, the value of μ is decreased progressively, which leads to a decrease in performance function as well (Mathworks, 2019).

As mentioned in Table 1 above, Resilient backpropagation and Scaled conjugate gradient backpropagation are commonly used for classification problems and for larger networks (Beale, et al., 2014). Hence, they will not be considered for this study.

The Levenberg Marquardt training algorithm is best suited for small/medium sized networks. It has the fastest performance even if that performance comes at the cost of memory (Mathworks, 2019).

4.3.6 ANN architecture

A feed forward multi layer perceptron (MLP) was chosen for this research. The reason being it is the most commonly used in time series analysis.

Figure 22 shows the architecture of the feedforward MLP.

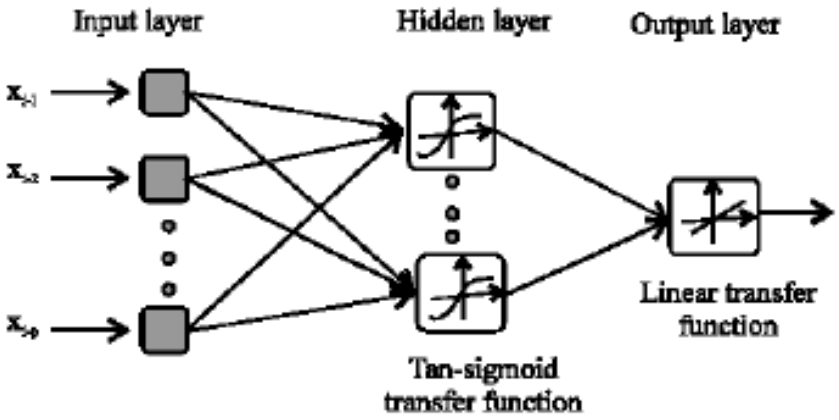


Figure 22: Three-layer feedforward network (Samsudin, et al., 2010)

The following parameters need to be specified for this architecture:

1) The number of inputs
   The number of inputs relates to the problem in context. In the case of time series forecasting, the number of inputs corresponds to the number of time steps used to make

the forecast. The ideal number of lags can be determined from the ACF/PACF plots which demonstrate the relationships between a time series and its lag values.

2) The number of outputs
   The number of outputs required is directly related to the information required. Since this research focuses on single step forecasting, only one output is required. Single step forecasting, as opposed to multi-step forecasting, forecasts one step into the future.

3) The number of layers
   The number of layers in a neural network usually correlates to the complexity of the task at hand. A feedforward neural network with a single hidden layer is usually able to learn most problems. If more complex tasks are required, then the number of layers can be gradually increased.

4) The number of neurons in the hidden layer
   The number of neurons in the hidden layer is chosen mostly through trial and error. There are no rules to determine the number of neurons in hidden layer. If the number of neurons chosen induces overfitting, then they should be gradually reduced until a good fit is obtained (Beale, et al., 2014).

5) The activation functions
   Usually, for the feedforward MLP, a tan-sig activation function is used for the hidden layer and a linear one for the output layer (Beale, et al., 2014).
   The linear activation function is commonly used for function fitting problems whilst the sigmoid function is required for pattern recognition problems (Beale, et al., 2014). In other words, the sigmoid function only generates an output between 0 and 1, whilst the linear function can generate an output linear to the input. The time series regression problem requires a real number output; hence a linear activation function is utilized for the output layer.

   The non-linear sigmoid activation function is used in the hidden layer as it is required to constrain the outputs of the layer to -1 and 1. The given time series data is highly non-linear; thus, a non-linear activation function was chosen for the hidden layer.

4.3.7 Cross validation

For forecasting using ANNs, the dataset needs to be subdivided into three subsets. These subsets are known as the training, validation and testing datasets. The training dataset is the largest one which is used for gradient computation and network parameter calculation and updating (Beale, et al., 2014; Haykin, 2009). The error typically decreases during the training

process. The validation set is the second subset of data; whose error is observed during the training process. Ideally, the validation error should decrease with the training process (Beale, et al., 2014). However, if the model overfits the data; the validation error increases (Haykin, 2009). The training is stopped when this occurs. The last dataset is known as the test dataset. It is a set of data unseen by the model during the training process; so, it used to test the effectiveness of the neural network model for forecasting accuracy (Beale, et al., 2014; Haykin, 2009).

4.4 Support Vector Machines

In addition to neural network methods, a new theory known as the support vector machine (SVM) has gained ground in recent years in the fields of classification and regression.

In 1995, Vapnik and his co-workers developed the SVM in the laboratories of A and T Bell. The basic principle behind SVM is Structural Risk Minimization (SRM) (Adhikari & Agrawal, 2013). SRM is a set of criteria for model selection which is used for generalizing from limited training datasets (Sewell, 2008). It provides a trade-off between model complexity and the quality of the fit of the higher dimensional feature space (Sewell, 2008). Figure 23 shows this principle in greater detail.

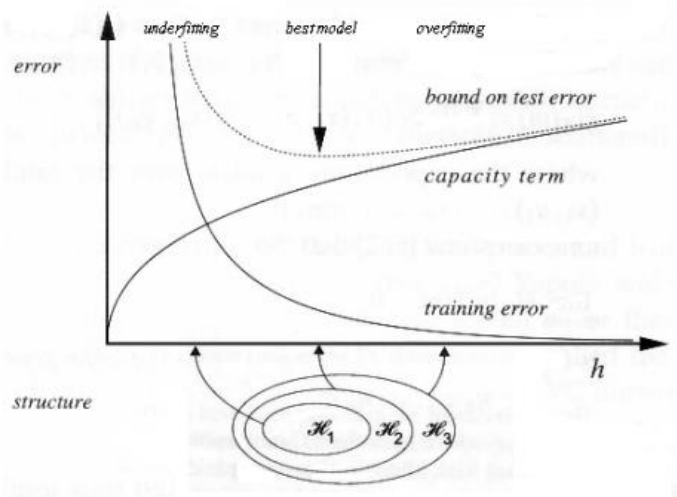


Figure 23: Structural Risk Minimization (Sewell, 2008)

4.4.1 SVM for Classification

When applied to classification problems, the main idea behind SVM is to find a hyperplane that separates the two classes of data in such a way that maximises the distance between them (Adhikari & Agrawal, 2013). There can be an infinite number of hyperplanes separating

the two classes. However, the idea is to find the optimal one. This can be seen in Figure 24 (a) below.

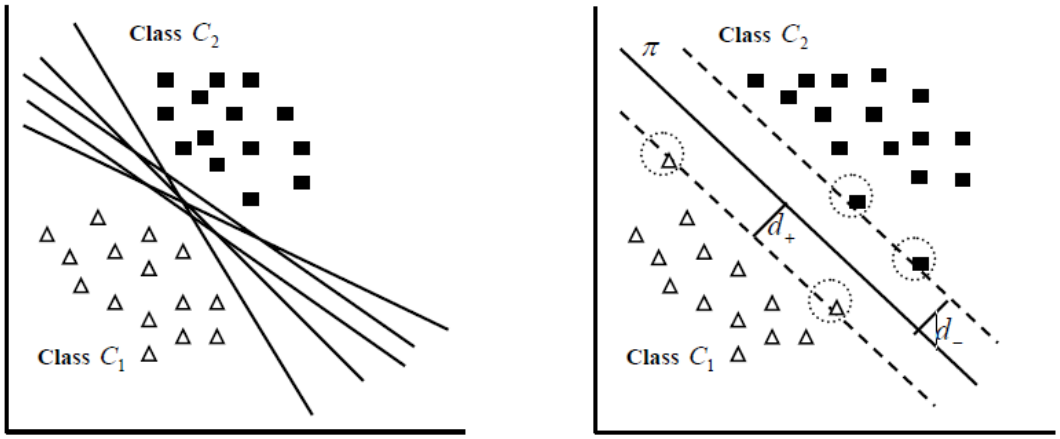


Figure 24 : The maximum margin hyperplane (Adhikari & Agrawal, 2013)

In Figure 24 , the distances *d+* and *d-* are known are the margins and the total margin is *M=(d+) + (d-).* The name of the hyperplane that maximises the distance between the vectors is known as the Maximum Margin Hyperplane (Adhikari & Agrawal, 2013). Support Vectors are the name given to the data points that lie on the maximum margin hyperplane. These support vectors can be seen circled in Figure 24 (b) above.

The functions $f(x, w, b) = sgn(w^T\ x + b)$, where w is the weight, b is the bias term and $x \in \mathbb{R}^n$ represent the hypothesis space (Smola & Scholkopf, 2004; Adhikari & Agrawal, 2013). Figure 25 depicts the maximum margin hyperplane in greater detail.

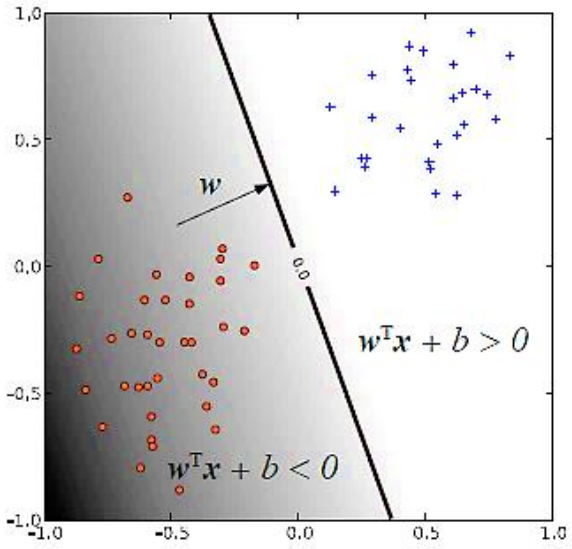


Figure 25: Representation of maximum margin hyperplane (Ben-Hur & Weston, 2010)

The hyperplane can be represented by $w^T x + b = 0$ where $x \in \mathbb{R}^n, w \in \mathbb{R}^n$ and $b \in \mathbb{R}$. The hyperplane divides the space into two, positive and negative spaces. The positive space is represented by $w^T x + b > 0$ and the negative space by $w^T x + b < 0$ (Ben-Hur & Weston, 2010). This essentially reduces the problem of finding the Maximum margin hyperplane to solving a quadratic programming problem (QPP) (Adhikari & Agrawal (2013), Smola & Vishwanathan (2008)).

4.4.2 SVM for classification for linearly separable data

Figure 25 above represents points that are linearly separable i.e. they can be separated by a linear decision boundary (Adhikari & Agrawal, 2013).

Let x_+ and x_- represent the closest points on the positive and negative sides of the hyperplane.

Then the margin of the hyperplane can be represented by:

$$m_D(f) = \frac{1}{2}\widehat{\boldsymbol{w}}^{\boldsymbol{T}}(x_+ - x_-) = \frac{1}{\|w\|} \tag{37}$$

where $\widehat{\boldsymbol{w}}^{\boldsymbol{T}}$ is a unit vector in the direction of w, $\|w\|$ is its length and $x_+ \; and \; x_-$ are equidistant from the decision boundary (Adhikari & Agrawal, 2013).

The aim is to maximise the geometric margin $1/\|w\|$ or minimise $\|w\|^2$. Thus, the QPP for linearly separable data can be expressed as (Adhikari & Agrawal, 2013):

$$Minimize\, J(w) = \frac{1}{2}w^T w = \frac{1}{2}\parallel w \parallel^2; \tag{38}$$

$$subject\ to\ y_i(w^T x_i + b) \geq 1; \forall i = 1,2 \ldots \ldots . N. \tag{39}$$

The solution to the QPP would yield optimized Lagrange multipliers $\boldsymbol{\alpha} = [\alpha_1, \alpha_2 \ldots \ldots \alpha_N]^T, \alpha_i > 0; \forall i = 1,2 \ldots \ldots . N$ and the optimal bias b_{opt} (Adhikari & Agrawal, 2013; Smola & Scholkopf, 2004). The support vectors are the data vectors for which $\alpha_i > 0$ and the total number of support vectors amount to N_s.

Then, the optimal weight vector is expressed by:

$$w_{opt} = \sum_{i=1}^{N_s} \alpha_i y_i x_i \,, \tag{40}$$

and the optimal hyperplane decision function is expressed by:

$$y(x) = sgn\left(\sum_{i=1}^{N} \alpha_i y_i (x^T x_i) + b_{opt}\right), \tag{41}$$

where α_i are the optimized Lagrangian multipliers, $x_i, y_i \in \mathbb{R}^n$ are data points in the hypothesis space and b_{opt} is the optimal bias (Adhikari & Agrawal, 2013).

4.4.3 SVM for classification for non-linearly separable data

The assumption for linearly separable data was that the optimisation problem classifies each data point correctly (Ben-Hur & Weston, 2010). If some amount of misclassification is allowed, then slack variables $\xi_i \geq 0$ are introduced into the QPP. Slack variables are variables that allow the data point to be in the margin $(0 \leq \xi_i \leq 1)$ or to be misclassified $(\xi_i > 1)$ (Ben-Hur & Weston, 2010).

When the data is not linearly separable, then a soft margin hyperplane is constructed and the QPP can be described as below (Adhikari and Agrawal, 2013):

$$Minimize\ J(w, \xi) = \frac{1}{2} \parallel w \parallel^2 + C\left(\sum_{i=1}^{N} \xi_i\right); \tag{42}$$

$$subject\ to\ y_i(w^T x_i + b) \geq 1 - \xi_i;\ \forall i = 1,2 \ldots \ldots . N; \xi_i \geq 0.$$

In this case, $C \geq 0$ is the regularization constant that applies a consequence to misclassified datapoints and ξ_i are the slack variables that ease the hard margin restrictions (Adhikari and Agrawal, 2013), $\|w\|$ is the length of the unit vector, $x_i, y_i \in \mathbb{R}^n$ are data points in the hypothesis space, w is the weight and b is the bias.

4.4.4 Support vector kernels

In the case where support vectors are not linearly separable, then they are mapped into a higher dimensional feature space (Adhikari & Agrawal, 2013). This will enable them to be separable by means of a linear decision boundary π_f. This can be seen diagrammatically in Figure 26 below.

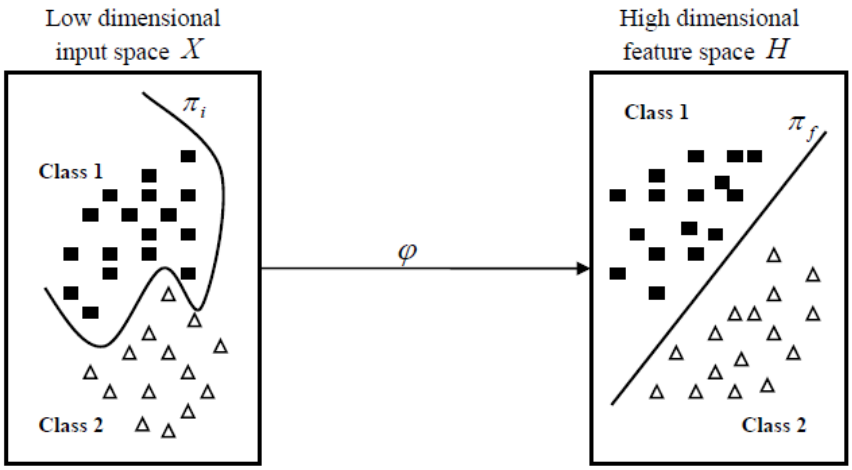


Figure 26: Mapping of support vectors into higher dimensional space (Adhikari & Agrawal, 2013)

In Figure 26, the data in the input space $X \in \mathbb{R}^n$ is charted on the feature space using the mapping $\varphi: X \rightarrow H$. Now the data points can be separated by a linear boundary π_f in the feature space (Ben-Hur & Weston, 2010).

The optimal hyperplane decision function described in Equation (41) can now be represented by:

$$y(x) = sgn\left(\sum_{i=1}^{N} \alpha_i y_i (\varphi(x)^T \varphi(x_i)) + b_{opt}\right) \tag{43}$$

The aim is now to find a kernel function $K(x_i x_j) = \varphi(x_i)^T \varphi(x_j)$ that can be used in the SVM training (Adhikari and Agrawal, 2013). This will enable SVM training without performing the φ mapping.

The most common kernel functions are described below:

- The linear kernel $K(x, y) = x^T y$ (Adhikari and Agrawal, 2013).
- The polynomial kernel $K(x, y) = (1 + x^T y)^d$ (Adhikari and Agrawal, 2013).
- The Radial Basis Function (RBF)/ Gaussian kernel $K(x, y) = exp(-\| x - y \|^2 / 2\sigma^2)$ (Adhikari and Agrawal, 2013).

4.4.5 Support vector regression

Similar to classification, support vector machines can be used for regression purposes.

In the case of support vector regression (SVR), the aim is to find a function *f(x)* that has a maximum of ε deviation from the maximum margin hyperplane (Smola & Scholkopf, 2004).

The function f(x) is described by:

$$f(x) = \langle w, x \rangle + b \; where \; w \in \chi, b \epsilon \mathbb{R}, \tag{44}$$

where χ is the input space, w is the weight, b is the bias term and $x \in \mathbb{R}^n$ represent the data points in the hypothesis space (Adhikari & Agrawal, 2013; Samsudin, et al., 2010).

Similar to how Equation (41) was derived, this can be summarized as a convex optimisation problem:

$$minimize \; \frac{1}{2} \parallel w \parallel^2, \tag{45}$$

$$subject \; to \begin{cases} y_i - \langle w, x_i \rangle - b \leq \varepsilon \\ \langle w, x_i \rangle + b - y_i \leq \varepsilon \, . \end{cases}$$

The assumption is that Equation (45) is feasible. In the event that the expression is not feasible or room is to be made for error, slack variables $\xi, \xi^*$are introduced to cope with the constraints of the optimisation problem (Smola & Scholkopf, 2004; Samsudin, et al., 2010).

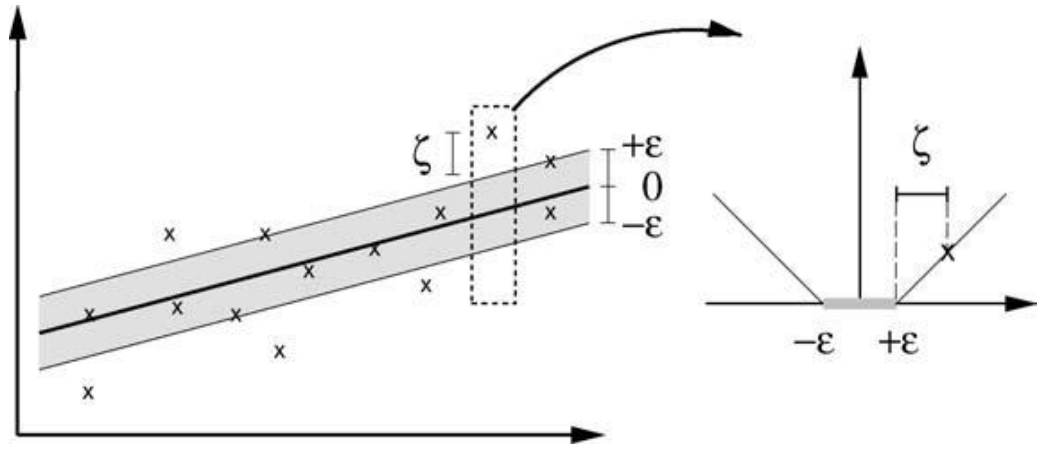


Figure 27 : Support vector regression (Smola & Scholkopf, 2004)

With the introduction of the slack variables, the convex optimisation problem becomes (Crone, et al., 2006; Adhikari & Agrawal, 2013; Samsudin, et al., 2010):

$$minimize \; \frac{1}{2} \parallel w \parallel^2 + C \sum_{i=1}^{N} (\xi_i + \xi_i^*), \tag{46}$$

$$subject \; to \begin{cases} y_i - \langle w, x_i \rangle - b \leq \varepsilon + \xi_i \\ \langle w, x_i \rangle + b - y_i \leq \varepsilon + \xi_i^* \, . \\ \xi_i, \xi_i^* \geq 0 \end{cases}$$

An ε – insensitive loss function can be defined by (Adhikari & Agrawal, 2013):

$$L_{\varepsilon}\big(y, f(x,w)\big) \quad = 0 \; if \; |y - f(x,w)| \leq \varepsilon \qquad (47)$$

$$= |y - f(x,w)| - \; \varepsilon \; otherwise$$

Equations (46) and (47) represent the main problem of support vector regression. The structural risk minimization principle is used in the objective fuction, which places emphasis on generalization ability and accuracy in the training set (Crone, et al., 2006). Here the parameter C represents the compromise between generalization ability and accuracy and ε represents the lenience to error (Crone, et al., 2006).

4.4.6 SVM Architecture

When applied to time series forecasting, the SVM architecture consists of a numbers of inputs transformed by a kernel function into a higher dimensional feature space. The output layer in addition to a bias provides the output of the regression. Figure 28 shows an SVM architecture in greater detail.

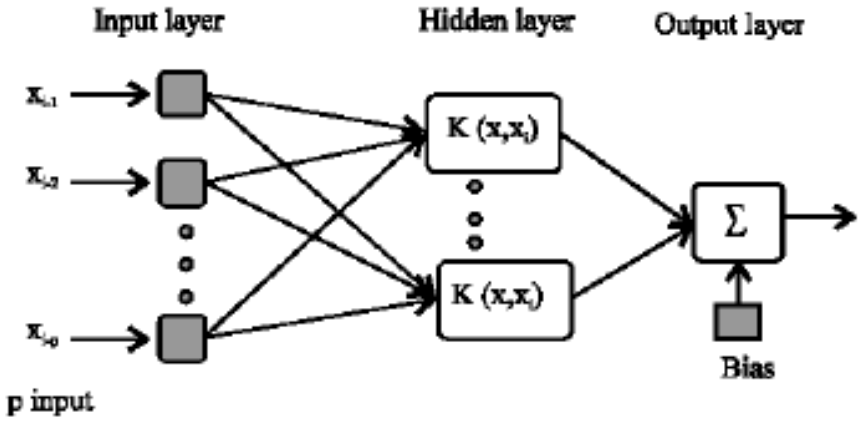


Figure 28 : SVM Architecture (Samsudin, et al., 2010)

The following need to be specified in an SVM architecture

1) Number of predictors

The goal of support vector regression is to predict one target value from a number of features or attributes (Hsu, et al., 2016). Thus, the number of predictors is an important element of the SVM architecture. In the case of time series forecasting, the number of predictors is equivalent to the number of inputs/number of lags.

2) Kernel function
A kernel function needs to be selected for transformation into a higher dimensional feature space (Crone, et al., 2006). The kernel functions available are linear, Gaussian/ RBF and the polynomial functions.

4.4.7 Cross validation

There are several hyperparameters which are required for each kernel function. For the linear kernel function, C and ε are the hyperparameters required (Hsu, et al., 2016). For the Gaussian/ RBF kernel function, C, ε and an additional hyperparameter γ are required. In the case of the polynomial kernel function, C, ε, γ and the polynomial order (either 1, 2 or 3) is required.

Unlike neural networks, cross validation in support vector regression is not an iterative process (Crone, et al., 2006). The process of cross-validation involves the division of the dataset into three subsets; training, validation and test datasets.

Cross validation is performed to reduce the possibility of overfitting. The process normally involves the selection of hyperparameters through a grid search (Ben-Hur & Weston, 2010). It is not known which combination of hyperparameters provide a good generalization capability; therefore, several different combinations are tested on the validation dataset; which is representative of the unseen data (Hsu, et al., 2016).

The model with the lowest validation error is selected and a new model is built from the combination of the training and validation sets. This model is tested on the hold-out sample/test dataset.

5. Data pre-processing

Data pre-processing consists of transforming the raw data so that it can be applied to the forecasting models. Sometimes the raw data is of the format that needs to be "cleaned" so that anomalies can be removed from it. Some of the commonly used data pre-processing methods are described below

5.1 Outlier removal

Outliers consist of data points that are usually differ substantially from the remaining data points in a dataset (Frost, 2020). Outliers need to be removed as they tend to misrepresent the dataset as a whole.

Figure 29 displays an example of an outlier.

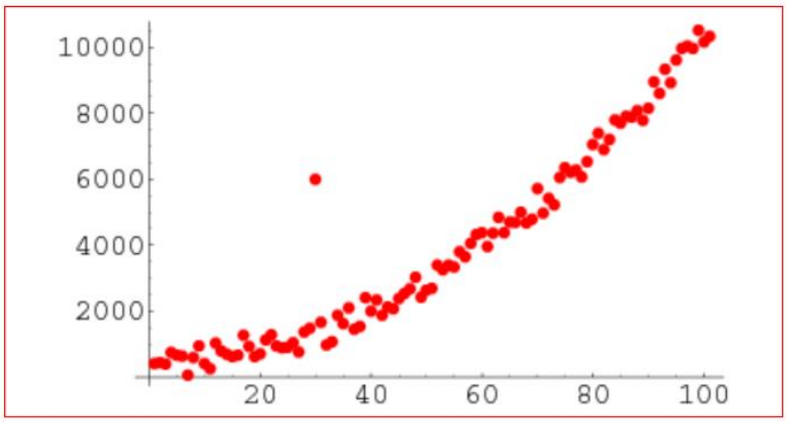


Figure 29: Scatter plot with outlier (Khandelwal, 2018)

5.2 Scaling

Scaling is the process of transforming the data series to fit within certain boundaries (Shumway & Stoffer, 2010). Data is usually scaled to fit within the [0, 1] range (Brownlee, 2016). Scaling is performed because some of the forecasting techniques are not equipped to deal with values outside of the boundaries

If Y(t) is a time series then the equation for the scaled time series Z(t) is as follows:

- Scaling [0, 1]

$$Z(t) = \frac{Y(t) - min}{max - min} \tag{48}$$

where *max* and *min* are the maximum and minimum values of the time series *Y(t)* (Dodge, 2003; Brownlee, 2016).

Scaling is normally applied for the machine learning techniques so that the series does not get trapped in local minima when training is performed.

5.3 Normalization

Normalization produces a normal distribution of the series, i.e. the time series is transformed into a series with mean equal to zero and standard deviation equal to one (Brownlee, 2016).

If Y(t) is a time series then the equation for the normalized time series Z(t) is (Dodge, 2003; Brownlee, 2016):

$$Z(t) = \frac{Y(t) - \mu}{\sigma} \tag{49}$$

where μ and σ are the mean and standard deviation of the time series $Y(t)$.

The equations for μ and σ are given by (Dodge, 2003):

$$\mu = \frac{1}{n}\sum_{t=1}^{n} Y(t) \tag{50}$$

$$\sigma = \sqrt{\frac{1}{n}\sum_{t=1}^{n} (Y(t) - \mu)^2} \tag{51}$$

5.4 Gaps in data

Sometimes data is missing from the dataset obtained from a public repository. In these situations, it is common to use smoothing techniques to fill in the gaps.

Some of the most common smoothing techniques are:

- Average: The average of the surrounding data points is used to fill in the missing data (Buckley & Butler, 2017).

- Moving average: The average of previous number of data points is used to fill in the missing data point (Zaiontz, 2020; Wolfram, 2020).

- Median: The median of a previous number of data points is used to fill in the missing data (Buckley & Butler, 2017; Wolfram, 2020).

- Local regression: The gaps are filled by a curve that best captures the data series (Wolfram, 2020).

5.5 Differencing

Differencing is a method of converting non-stationary time series into stationary time series (Athanasopoulos & Hyndman, 2018; Wei, 2006; Box, et al., 2015).

Majority of time series forecasting methods can only work on stationary time series; which have stochastic properties which do not change over time (Box & Jenkins, 1990). Thus non stationary time series need to be converted into stationary time series.

The various methods of differencing which may be applied are (Athanasopoulos & Hyndman, 2018; Box, et al., 2015):

- Simple differencing

$$D(t) = Z(t) - Z(t-1) \tag{52}$$

where $Z(t)$ is the time series and $Z(t-1)$ is the time series one step prior (Athanasopoulos & Hyndman, 2018).

- Second order differencing
  Often the differenced values need to be differenced once more to remove all traces of non-stationarity (Athanasopoulos & Hyndman, 2018). This can be expressed as

$$R(t) = Z(t) - Z(t-1) - (Z(t-1) - Z(t-2)) \tag{53}$$

$$= Z(t) - 2Z(t-1) + Z(t-2))$$

where $Z(t)$ is the time series, $Z(t-1)$ is the time series one step prior and $Z(t-2)$ is the time series two steps prior (Athanasopoulos & Hyndman, 2018).

- Transformative differencing
  A transform like logarithm, square root or power is applied to the time series before differencing is applied (Singh, 2018). Logarithmic differencing can be seen in equation (54) below.

$$LR(t) = \log(Z(t) - \log(Z(t-1)) \tag{54}$$

$$= log \frac{Z(t))}{Z(t-1)}$$

where $Z(t)$ is the time series and $Z(t-1)$ is the time series one step prior (Singh, 2018).

6. Performance Measures

There are several methods to measure performance of the chosen forecasting algorithm. They are described in greater detail below.

6.1 Root Mean Squared Error

The root mean squared error (RMSE) is usually applied to regression; and is a measure of how much the errors deviate from the regression line (Athanasopoulos & Hyndman, 2018; Davis & Heineke, 2005).

The formula for the RMSE is given by:

$$S_{YX} = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}} \tag{55}$$

where y_i is the original dataset,

$\hat{y}_i$ is the forecast and

n is the number of data points (Davis & Heineke, 2005).

6.2 Mean Squared Error

Mean squared error (MSE) is also known as the variance. It is the square of the standard error (Athanasopoulos & Hyndman, 2018; Davis & Heineke, 2005).

The formula for MSE is given by:

$$MSE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n} \tag{56}$$

where y_i is the original dataset,

$\hat{y}_i$ is the forecast and

n is the number of data points (Davis & Heineke, 2005).

6.3 Mean Absolute Deviation/Mean Absolute Error

Mean absolute deviation (MAD) is the sum of the absolute deviations of forecasted demand

from actual demand divided by the number of data points (Davis & Heineke, 2005).

In equation form, MAD is given by:

$$MAD = \frac{\sum_{i=1}^{n} |A_t - F_t|}{n} \tag{58}$$

where A_t is the actual demand,

F_t is the forecasted demand and

n is the number of data points (Davis & Heineke, 2005).

6.4 Mean Average Percentage Error

The Mean Average Percentage Error establishes the forecast errors as a percentage of the actual. This is useful if a relative forecast is required (Davis & Heineke, 2005).

The MAPE is calculated using the following formula

$$MAPE = \frac{\sum_{t=1}^{n} |A_t - F_t|}{n} \times 100 \tag{59}$$

where

A_t is the actual demand,

F_t is the forecast, and

n is the number of time steps (Davis & Heineke, 2005).

7. Research Method

7.1 Enabling software and hardware

Microsoft Excel Professional Plus 2010 was used for the data pre-processing.

MATLAB version R2017a was used for the implementation and forecasting using ANNs, SVR and ARIMA.

The study was conducted on a personal computer, using a 64-bit Microsoft Windows software operating system with 16GB ram and an *Intel core i3* processor.

7.2 Data description

This section describes the data used for the investigation. The data was obtained from data repository website "Kaggle" where various datasets are available for public use. The specific dataset used for this study was "Historical Product Demand", which consists of product demand for goods for a company in the manufacturing sector (Zhao, 2017). The first five product demand categories were selected for the purposes of this research.

The raw data as extracted from the website consists of columns of data, which are described in further detail in Table 2 below (Zhao, 2017).

Table 2 : Description of Kaggle dataset

Column	Variable	Description	Data Type
1	Product Code	Product identifier	Text
2	Warehouse	Warehouse identifier	Text
3	Product Category	Category identifier	Integer
4	Order Demand	Demand of the product	Number
5	Date	Date at which the demand occurred	Date

There are 2172 product codes belonging to 25 different categories which can be in one of three different warehouses. The original dataset has four line items per product code as can be seen in Table 2 above.

For the purposes of this research, the warehouse and product category are not significant. Hence, they are not included in the final dataset which consists of the product code, order demand and date.

The dataset was sorted into columns consisting of product name and product demand from January 2012 to December 2016 using Excel's data management tools.

7.3 Ethical clearance

The ethics clearance number is MIAEC 039/19W and the clearance was approved by the School of Mechanical, Aeronautical and Industrial Engineering.

7.4 Assumptions

The assumptions made for this study are:

- The ANN, SVR and ARIMA methods can be used on datasets of a limited size of 60 points.
- All the different ANN, SVR and ARIMA methods can be utilized on highly variable time series data.

7.5 Methodology Overview

The research method can be summarised as per Figure 30 below:

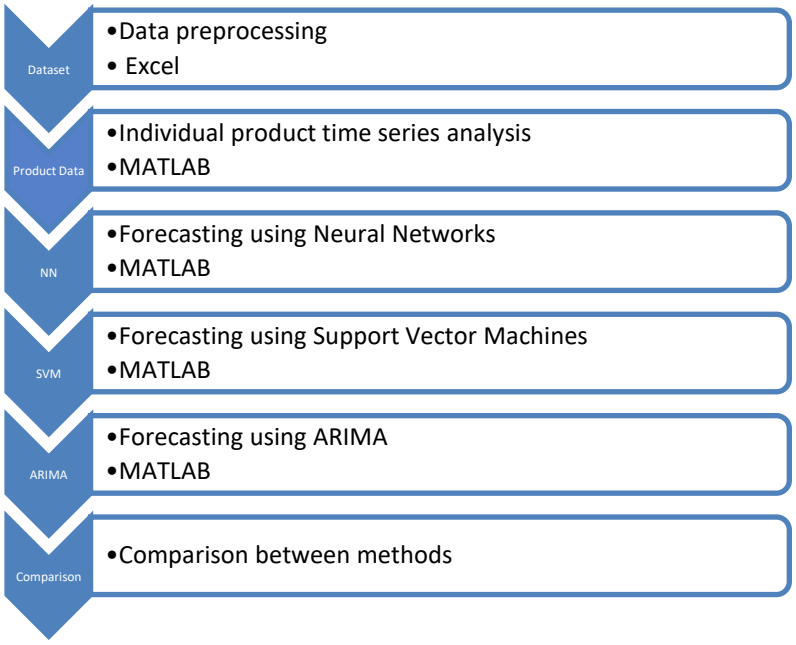


Figure 30: A flowchart of the methodology overview

7.5.1 Data pre-processing

An excerpt of the raw data extracted from Kaggle is shown below:

Product_Code	Warehouse	Product_Category	Order_Demand	Month	Year	Date
Product_0993	Whse_J	Category_028	100	7	2012	2012/07/27
Product_0979	Whse_J	Category_028	500	1	2012	2012/01/19
Product_0979	Whse_J	Category_028	500	2	2012	2012/02/03
Product_0979	Whse_J	Category_028	500	2	2012	2012/02/09
Product_0979	Whse_J	Category_028	500	3	2012	2012/03/02
Product_0979	Whse_J	Category_028	500	4	2012	2012/04/19
Product_0979	Whse_J	Category_028	500	6	2012	2012/06/05
Product_0979	Whse_J	Category_028	500	6	2012	2012/06/27
Product_0979	Whse_J	Category_028	500	7	2012	2012/07/23
Product_0979	Whse_J	Category_028	500	8	2012	2012/08/29
Product_0979	Whse_J	Category_028	500	8	2012	2012/08/29

Figure 31: Raw Kaggle data

One cannot gain insight into this data without processing it further. Excel's PivotChart was used to organise the data into rows and columns which can be analysed further. The entire dataset was selected and a PivotTable was constructed from it.

The PivotTable consists of the following:

- Row Labels: The RowLabels consists of the Product Demand categories. i.e. Product 1, Product 2 etc.
- Column Labels: The ColumnLabel consists of the quantity of product that is sorted periodically from 2012 to 2016.
- Filters: The filters are used to fill in the information in the RowLabels and ColumnLabels

After the PivotTable was constructed, the data could be viewed in detail and the following steps were taken to process it further:

- Gaps in data: The gaps in the data were filled by taking the average of the preceding and succeeding data points
- Negative values: Negative values were converted to positive by simply using the positive value of the negative data point.
- Zero values: Zero values were deleted and replaced with the average of the preceding and succeeding data points.

Since the data was extracted from a public repository, there are huge chunks of missing data in some of the product data information. These product categories were deleted from the final dataset which consists of 1803 product demand categories.

7.5.2 Individual product time series analysis

The following methodology was followed for the individual product data:

1. Plot the individual product time series data in MATLAB
2. Plot the series ACF and PACF plots
3. Determine the appropriate number of lags from ACF and PACF plots
4. Use the number of lags for the ANN forecasting model, SVR forecasting model and ARIMA model.

7.5.3 Forecasting using ANNs

7.5.3.1 Scaling of data

Scaling of the raw data into a range of [0, 1] is performed before it is processed by the ANN model.

As described in Section 5.2, if the dataset consists of $x_i = (x_1, x_2, x_3 \dots \dots \dots \dots . x_n)$, then the scaled values are obtained by:

$$z_i = \frac{x_i - \min(x_i)}{\max(x_i) - \min(x_i)} \tag{60}$$

7.5.3.2 Sliding window implementation

ANN based time series forecasting is based on the assumption that the future values of the time series are dependent entirely on its history/past values and can be used to forecast (Said, et al., 2013). The past observations are used as inputs to the neural network and the future value is the output (Said, et al., 2013). A forecast at time t+1 ($\hat{y}$) is computed from the previous values $y_n, y_{n-1} \dots . . y_{t-n+1}$ at time periods $t, t-1, t-2, \dots \dots, t-n+1$ where n is the preceding points in time.

This means that $\hat{y}$ is a function of the previous n values $y_n, y_{n-1} \dots . . y_{t-n+1}$

$$\hat{y} = f(y_n, y_{n-1} \dots . . y_{t-n+1}) \tag{61}$$

The ANN model is configured as a sliding window so that multiple predictions can be made over time.

The number of lags determined in Section 7.4.2 is used for the number of input nodes. For example, for an input lag number of eight, the sliding window can be implemented as shown in Table 3 below (Said, et al., 2013).

Table 3 : Sliding window implementation example

Number	Input lags	Forecasted output values
1		
2		
3		
4		
5		
6		
7		
8		
9	$y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8$	$\hat{y}_9$
10	$y_2, y_3, y_4, y_5, y_6, y_7, y_8, y_9$	$\hat{y}_{10}$
11	……	…..
60	$y_{52}, y_{53}, y_{54}, y_{55}, y_{56}, y_{57}, y_{58}, y_{59}$	$\hat{y}_{60}$

7.5.3.3 MATLAB function

The feedforward MLP configuration is readily available in MATLAB; specially designed for time series analysis. It is available as “narnet”; which stands for non-linear autoregressive neural network.

The syntax in MATLAB is narnet (feedbackDelays, hiddenSizes, trainFcn) where feedbackDelays represents the number of inputs to the network, hiddenSizes represents the number of layers and trainFcn represents the training algorithm.

For example, for an ANN with two inputs, a 10-neuron hidden layer and a Levenberg Marquart training algorithm, the layout structure can be represented as shown in Figure 32 below.

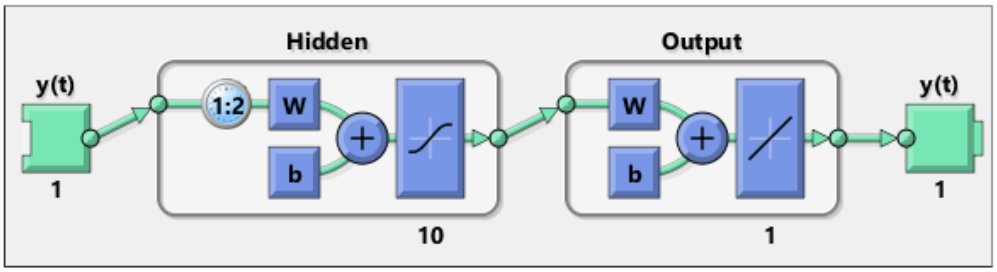


Figure 32 : narnet structure (Source: MATLAB)

7.5.3.4 Training, validation and test datasets

As mentioned in Section 7.2, the dataset consists of product demand data from January 2012 to December 2016 after pre-processing.

The individual product data was divided into training, validation and test datasets as follows: 70%, 15%, 15%.

The data division is explained as follows:

- training dataset: Consists of the first 70% of the data
- the validation set: Consists of the next 15% of the data
- test dataset: Consists of the last 15% of the data

The sliding window shown in Table 4 will be used as example to demonstrate the data division. Table 4 below demonstrates the dataset with the number of lag structures.

Table 4: Example lag structures

Number	Input lag structure	Forecasted output values
1	$y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8$	$\hat{y}_9$
2	$y_2, y_3, y_4, y_5, y_6, y_7, y_8, y_9$	$\hat{y}_{10}$
3	$y_3, y_4, y_5, y_6, y_7, y_8, y_9, y_{10}$	$\hat{y}_{11}$
4	$y_4, y_5, y_6, y_7, y_8, y_9, y_{10}, y_{11}$	$\hat{y}_{12}$
5	$y_5, y_6, y_7, y_8, y_9, y_{10}, y_{11}, y_{12}$	$\hat{y}_{13}$
	……	
	……..	
52	$y_{52}, y_{53}, y_{54}, y_{55}, y_{56}, y_{57}, y_{58}, y_{59}$	$\hat{y}_{60}$

Since the dataset consists of 52 input lag structures the data division would be as follows:

- Training: 70% of 52 = 39. This would consist of the first 39 lag structures.
- Validation: 15% of 52 = 8. This would consist of the next eight lag structures.
- Test: 15% of 52 = 8. This would consist of the last eight lag structures.

The data division is entirely dependent on the input number of lags as the data lag structure would differ according to the input number of lags.

7.5.3.5 Hyperparameter optimization

There are three major parameters that can be tuned in order to maximise the learning potential of the neural network:

1) Number of inputs
   The number of inputs was selected from the number of lags as described in Section 7.2.

2) Number of neurons in hidden layer
   The generalization capability of an ANN is provided by the number of neurons in the hidden layer. Thus, a sufficient number of neurons must be chosen through a trial and error method. If the ANN starts overfitting, then the number of neurons is sufficiently reduced until the ANN achieves a good fit.

3) The training algorithm
   There are no set guidelines on how to select a training algorithm. Variations of the training algorithms mentioned in Section 4.2.5 above were implemented for forecasting each different product demand category.

7.5.3.6 Performance measurement

The performance of the forecasting algorithm was measured using RMSE, MSE and MAD. MAPE was not utilized as at least one of the values that are scaled values is zero; hence calculation of MAPE would yield an infinite response

7.5.3.7 MATLAB Algorithm

For the forecasting performed in MATLAB, the experimental algorithm shown in Figure 33 was followed. The algorithm was followed for all of the product demand data.

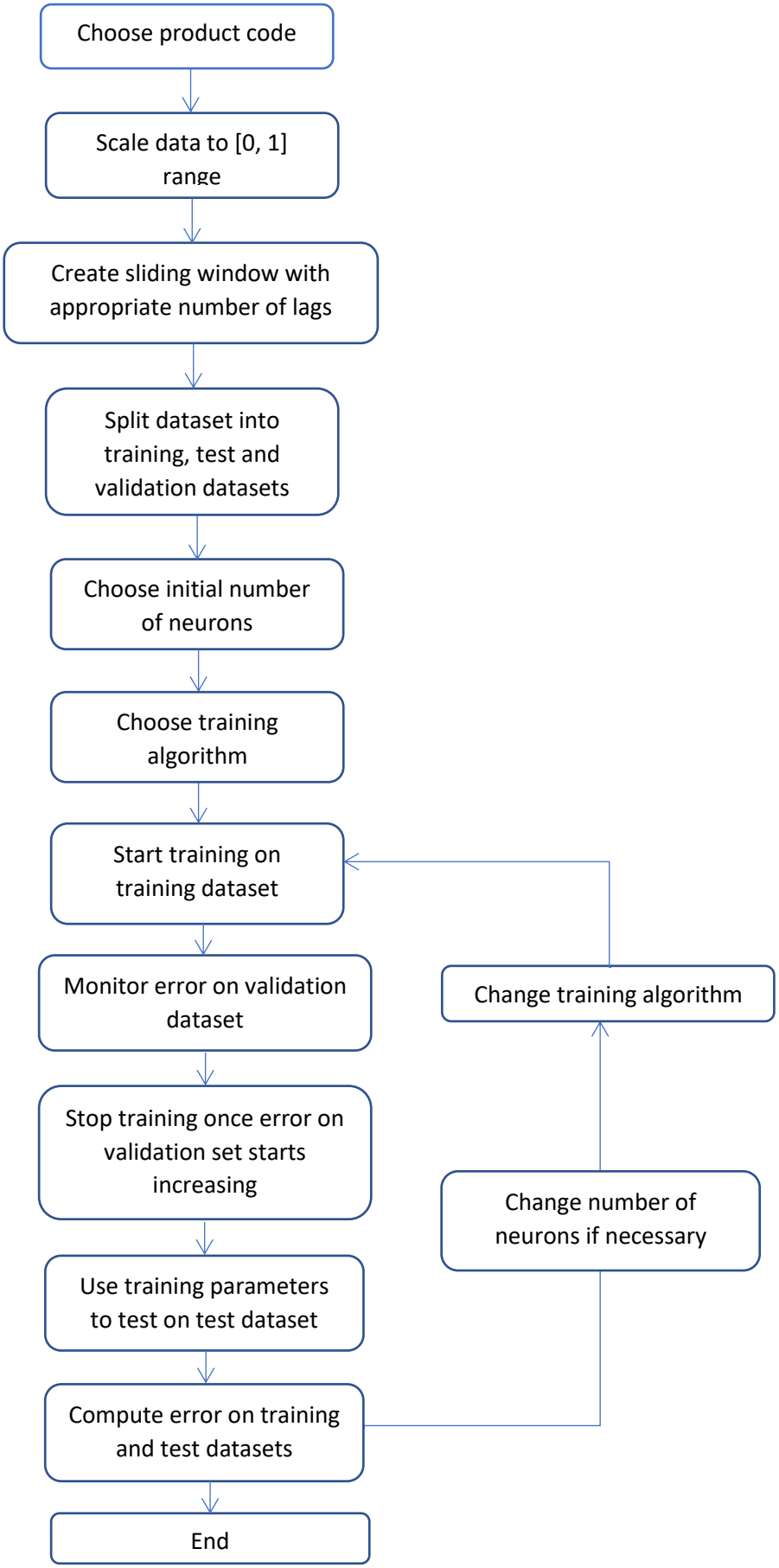


Figure 33: A flowchart of the MATLAB algorithm for ANN based forecasting

From Figure 33 above, it can be seen that the first step involves choosing the product code from the imported Excel sheet. Scaling is performed thereafter to transform the data into [0,1] range. A sliding window is created with the appropriate number of lags determined from Section 7.4.2. The dataset is split into training, validation and test datasets. An initial number of neurons and a training algorithm is chosen. Training is initialized and the error on the validation set is monitored. Training is continued until the error on the validation set begins to increase. Thereafter, training is stopped. The training parameters are used on the test dataset and performance measures are calculated. If satisfactory results are not obtained, the number of neurons is changed until the desired results are obtained. The procedure is repeated for different training algorithms listed below

- Gradient descent algorithm - traingd
- Gradient descent with adaptive learning rate- traingdx
- Gradient descent with momentum backpropagation - traingdm
- Bayesian Regularization - trainbr
- Conjugate gradient backpropagation with Fletcher Reeves restarts - traincgf
- Conjugate gradient backpropagation with Polak/Ribiere restarts - traincgp
- Conjugate gradient with Beale-Powell restarts - traincgb
- BFGS Quasi Newton backpropagation - trainbfg
- One step secant - trainoss
- Leverberg Marquardt – trainlm

Thereafter, the errors are calculated on the training and test datasets.

7.5.4 Forecasting using SVR

7.5.4.1 Scaling of data

Similar to ANN forecasting, the input data was scaled to the range [0, 1].

7.5.4.2 Sliding Window Implementation

A sliding window was applied for SVR forecasting, exactly the same as ANN forecasting. This is dependent on the input number of lags and is described in Section 7.4.3.2.

7.5.4.3 MATLAB function

The support vector regression model was implemented using *fitrsvm* in MATLAB. *Fitrsvm* is a MATLAB function that predicts the data using kernel functions via quadratic programming

(Mathworks, 2019). Appendix C contains the regression model MATLAB code.

7.5.4.4 Training, validation and test datasets

The data was subdivided into training, validation and test datasets exactly the same as in ANN forecasting, which is described in Section 7.4.3.4.

7.5.4.5 Hyperparameter optimization

The following hyperparameters are utilized by the fitrsvm MATLAB function depending on the kernel function:

- Box Constraint
  This is the MATLAB term for the parameter C which represents the penalty attached to errors. Large values of BoxConstraint/C have large penalties associated with them whereas smaller values have smaller penalties associated with them.
- Kernel Scale
  The KernelScale represents the width of the kernel function.
- Epsilon
  The Epsilon refers to the width of the support vector margin.
- Polynomial order
  The polynomial order refers to the order of the polynomial kernel.

The *fitrsvm* function fits an SVR model based on the choice of kernel function. Each kernel function requires the use of hyperparameters specific to that kernel function.

- Linear kernel: The linear kernel utilizes the Box constraint and Epsilon parameters
- Gaussian kernel: The Gaussian kernel utilizes the Box constraint, Epsilon and Kernel scale parameters
- Polynomial kernel: The Polynomial kernel utilizes the Box constraint, Epsilon, Kernel scale and Polynomial order parameters

7.5.4.6 Performance measure

The RMSE, MSE and MAD were chosen as the performance measures of the SVR model, similar to the ANN forecasting model.

7.5.4.7 MATLAB Algorithm for SVR based forecasting

Figure 34 depicts the flowchart of the MATLAB algorithm of SVR based forecasting.

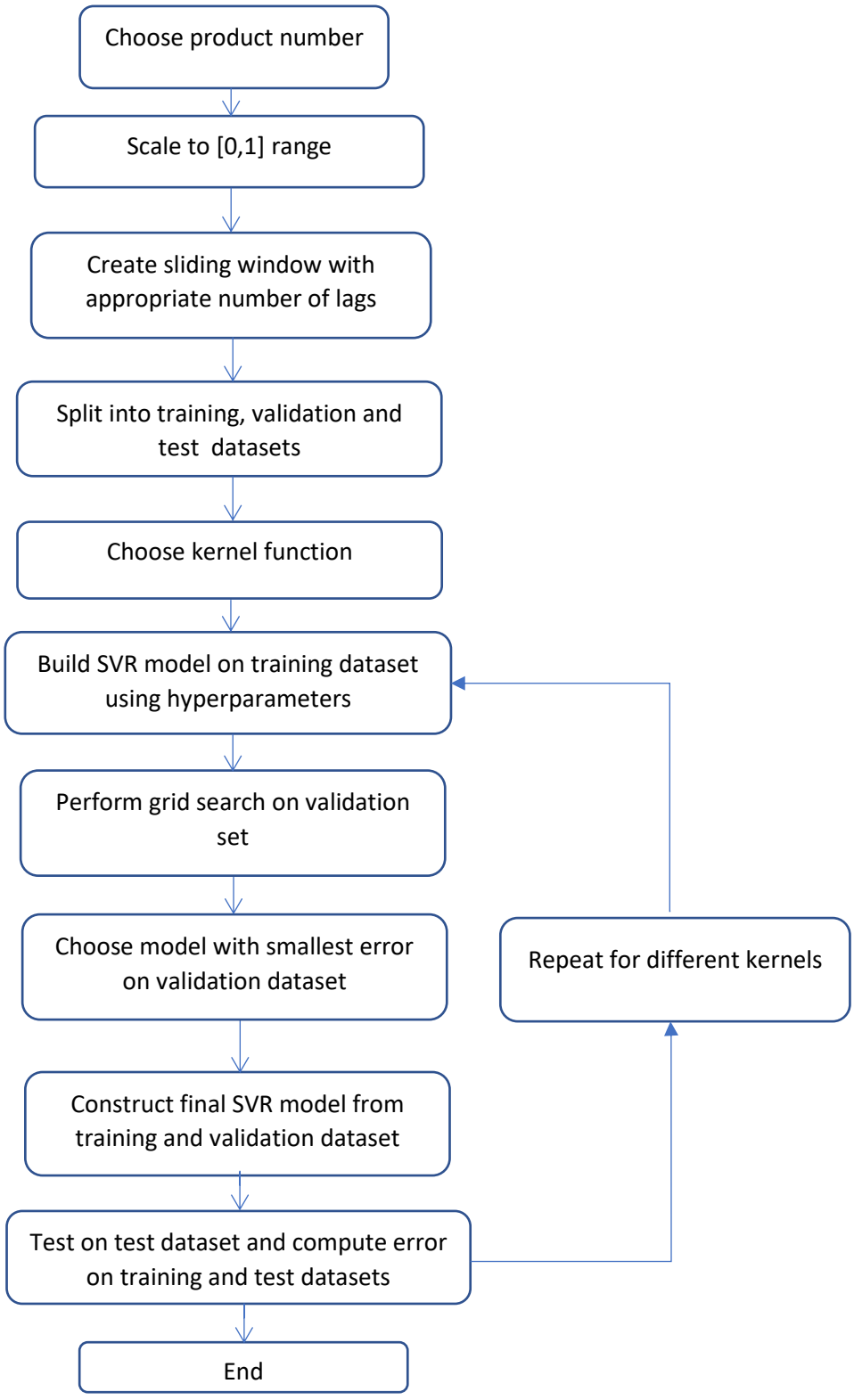


Figure 34: A flowchart of the MATLAB algorithm for SVR based forecasting

From Figure 34, the first step involves importing the product code into MATLAB and scaling the data to [0,1] range. An appropriate sliding window was created and dataset split into training, validation and test datasets. A kernel function was chosen and an SVR model built on the training set using a set of hyperparameters that provides the best fit for the training set. Different combinations of hyperparameters are tested on the validation set. The model with the smallest error on the validation set is chosen and the final model is built using the combined test and validation sets. This model is now tested on the test dataset. The error measurements are calculated. The process is repeated for different kernel functions.

7.5.5 ARIMA forecasting

7.5.5.1 Scaling

Similar to ANN and SVR based forecasting, the data was scaled to [0,1] range.

7.5.5.2 Sliding window implementation

A sliding window was implemented for the ARIMA forecasting similar to ANN and SVR forecasting using the appropriate number of lags.

7.5.5.3 MATLAB function

The *arima* function in MATLAB was used to implement the ARIMA model.

7.5.5.4 Training and test datasets

Unlike the machine learning models, there is no splitting into training, validation and test datasets. Instead, the dataset is split into training and test datasets. The training dataset is created using the first 85% of the lag structures while the test dataset is created using the last 15% of the lag structures.

7.5.5.5 Performance measure

The RMSE, MSE and MAD were chosen as the performance measures.

7.5.5.6 *MATLAB algorithm*

Figure 35 depicts the flowchart of the MATLAB algorithm for the ARIMA model.

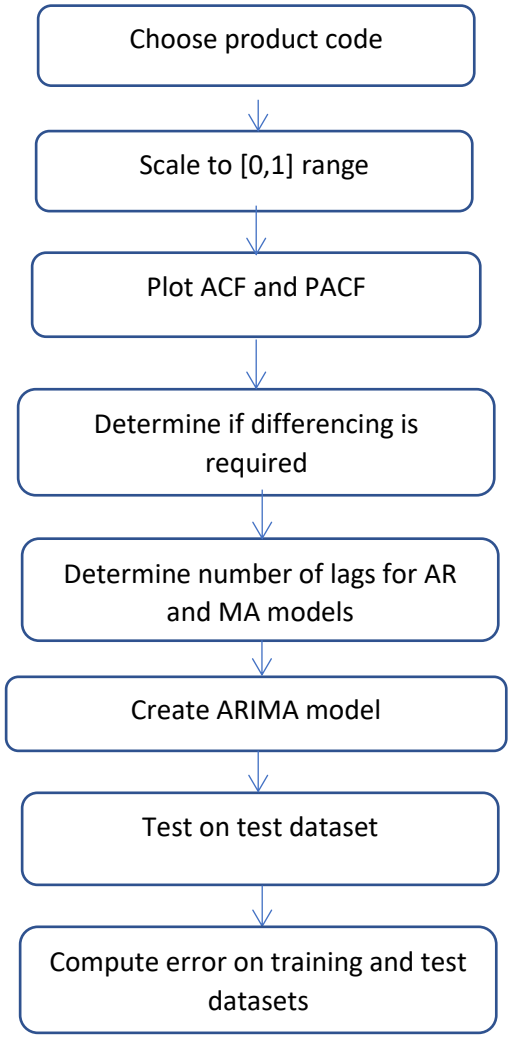


Figure 35: A flowchart of the MATLAB algorithm for ARIMA model

From Figure 35, the first step involves choosing the product code and scaling it to [0,1] range. Afterwards, the ACF and PACF of the data is plotted. It is determined if the data needs to be differenced or not. The number of lags for the AR and MA models are determined. The ARIMA model is then implemented in MATLAB and the error is calculated on the training and test sets.

8. Results and Analysis

The Kaggle dataset consists of product demand categories that vary in quantity from very small to very large. In order to standardize the products, all the products categories were scaled to the [0, 1] range. Out of the 1803 product categories in the dataset, the following characteristics were observed:

- Thirty eight percent display no trend and no seasonality with only cyclical and irregular behaviour
- Forty one percent display slight trend with no seasonality
- Twenty one percent display clear trend with no seasonality
- Zero percent of the products display seasonality

Five of the products were selected to represent these categories of products. Product 1 and 2 display no trend and seasonality, Product 3 and 4 slight trend and no seasonality and Product 5 displays clear trend with no seasonality.

8.1 Product 1

Figure 36 shows Product 1 as a function of time.

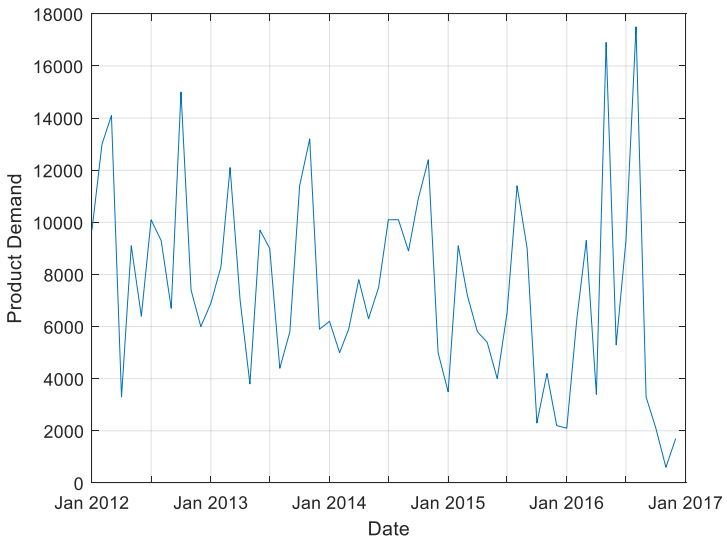


Figure 36: Time series plot of Product1

An ACF and PACF is required to determine if the series has any trend/seasonal components. If the series has trend and seasonality, it would be necessary to make it stationary using differencing.

8.1.1 Data analysis and pre-processing for Product 1

Figure 37 and 38 display the ACF and PACF of Product 1.

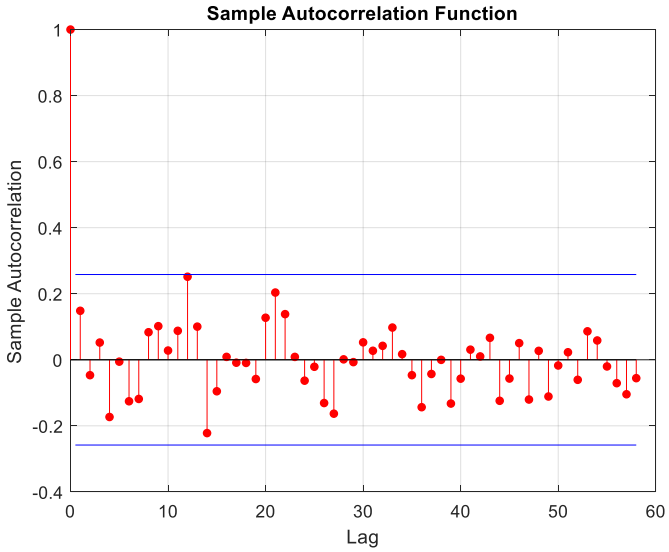


Figure 37: ACF of Product 1

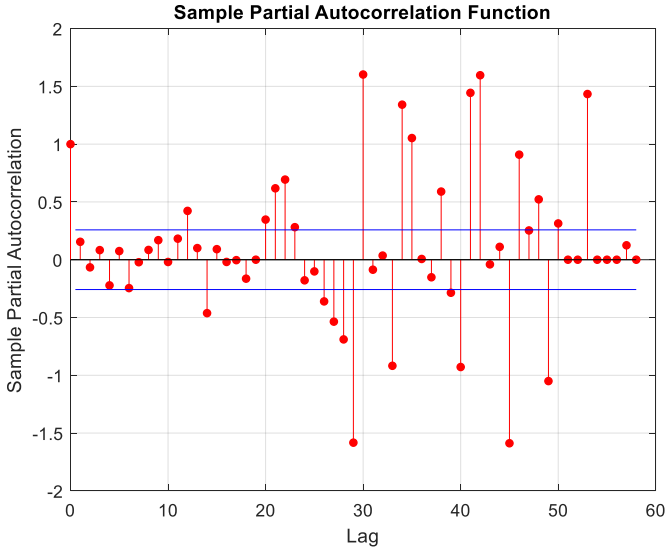


Figure 38: PACF of Product 1

An examination of the ACF does not demonstrate any trend or seasonality. There is no gradual decrease in ACF and also no repeats at fixed intervals. Thus, no further data pre-processing is required.

From an examination of the PACF, the first six lags were deemed as significant before the correlations decay to zero. So, the appropriate number of lags for the neural network/support vector regression and ARIMA AR model was selected as six. In other words, a sliding window of six lags will be utilized to forecast the next value.

For the ARIMA MA model, a lag of four was chosen from the ACF.

8.1.2 ANN based forecasting for Product 1

Figures 39 – 48 show the results of ANN based forecasting using the different training algorithms mentioned in Section 4.3.5. The performance plots for each training algorithm depict the change in MSE with the number of iterations for the training, validation and test sets. Training is performed while the validation error is monitored and is continued until the validation performance reaches a minimum.

The regression plots display the relationship between the outputs and the targets of the ANN for the training, validation and test datasets (Beale, et al., 2014). If training was performed perfectly, then the relationship between inputs and outputs would be one (R=1). Additionally, the linear regression line would be directly on top of the dashed line which indicates a perfect fit between outputs and target values (Beale, et al., 2014).

- traingd

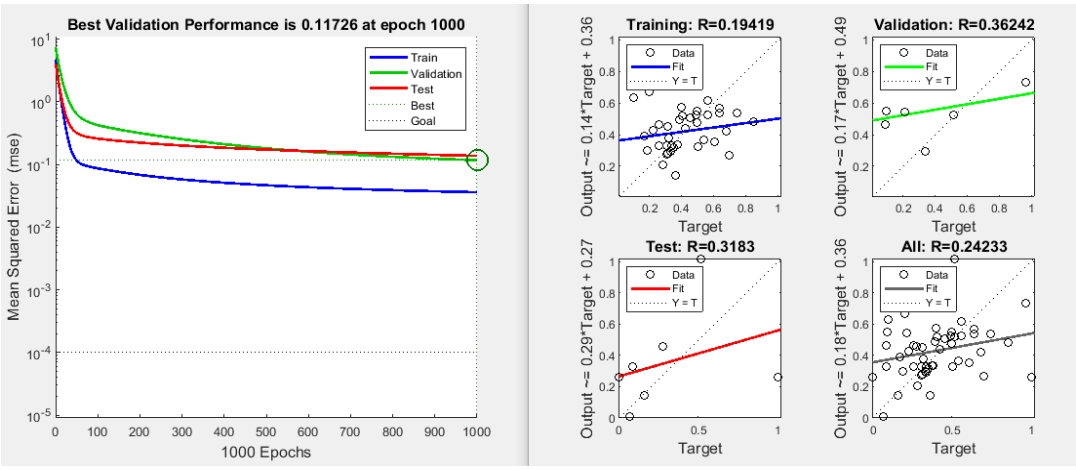


Figure 39: traingd performance and regression plots for Product 1

- traingdx

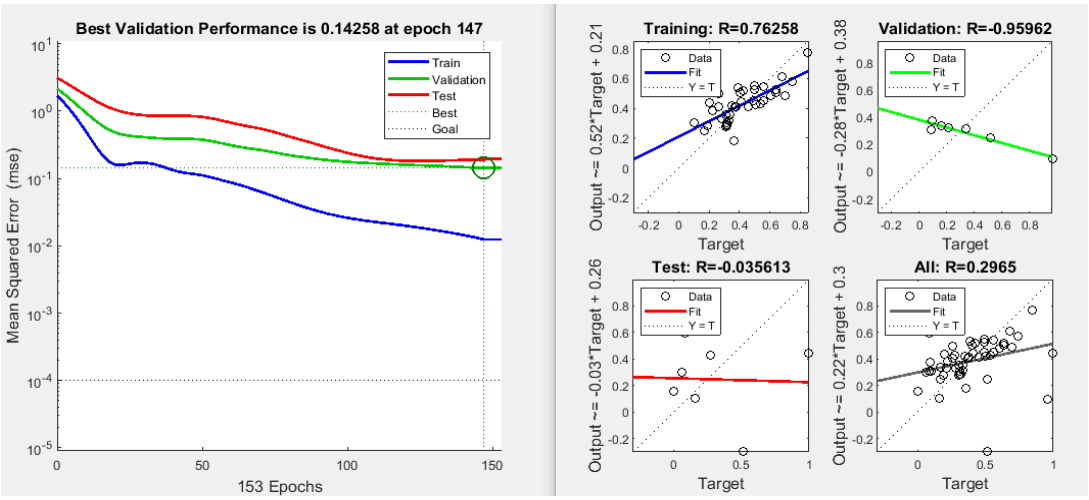


Figure 40: traingdx performance and regression plots for Product 1

- traingdm

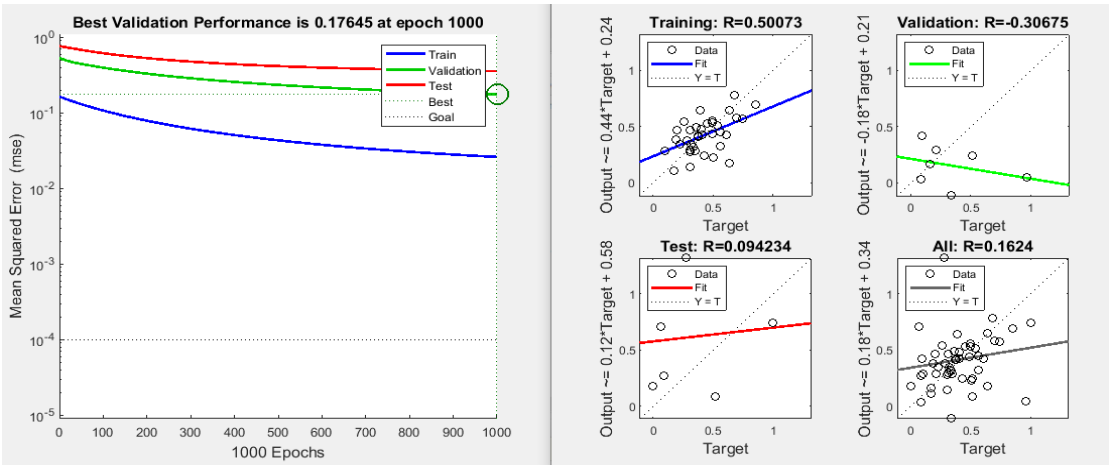


Figure 41: traindgm performance and regression plots for Product 1

- trainbr

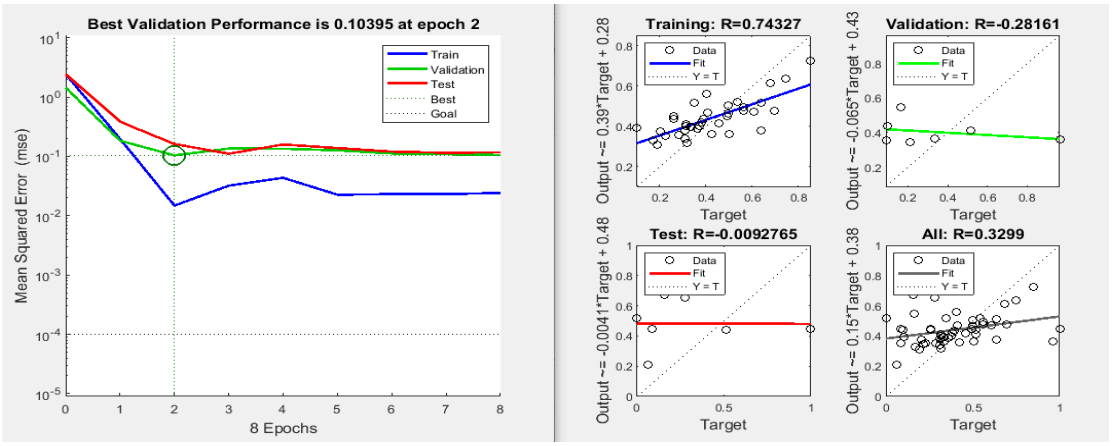


Figure 42: trainbr performance and regression plots for Product 1

- traincgf

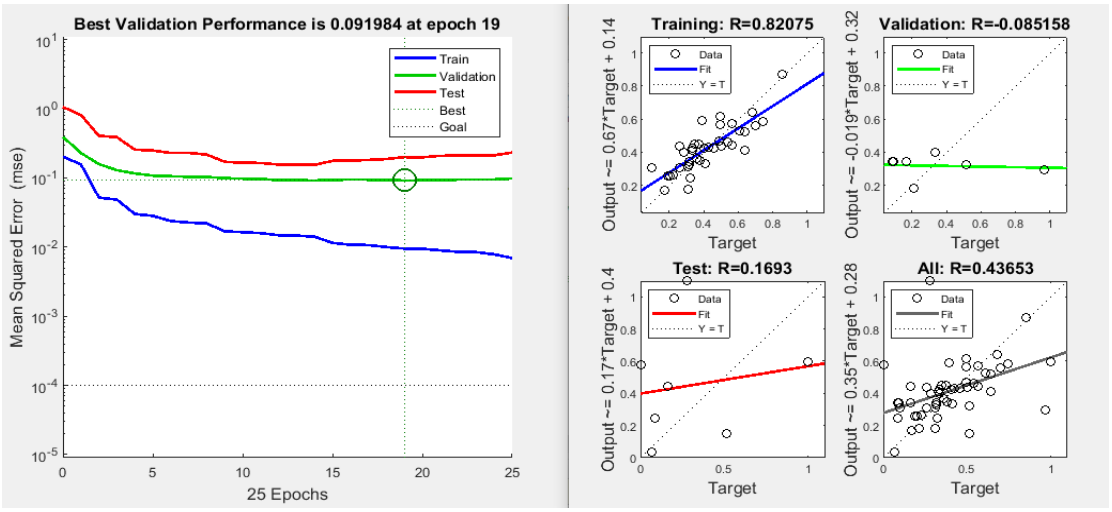


Figure 43: traincgf performance and regression plots for Product 1

- traincgp

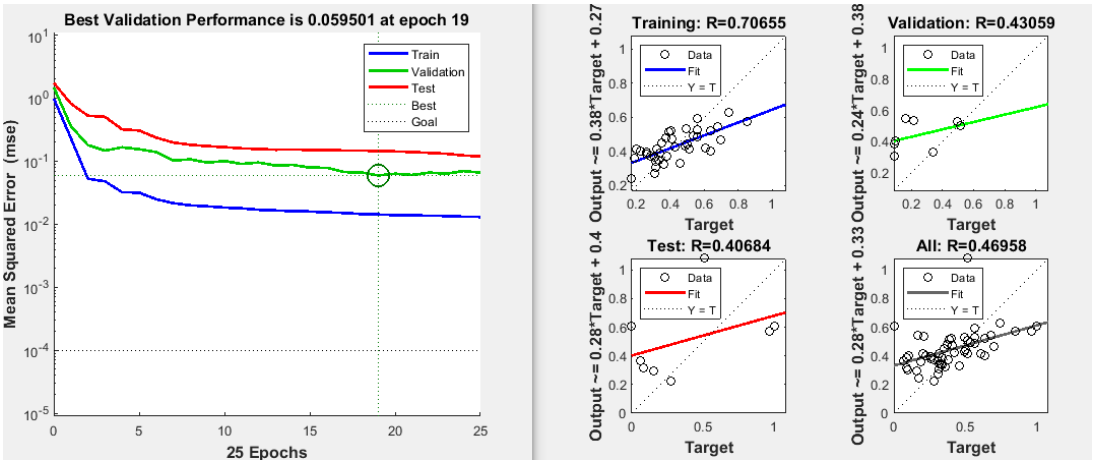


Figure 44: traincgp performance and regression plots for Product 1

- traincgb

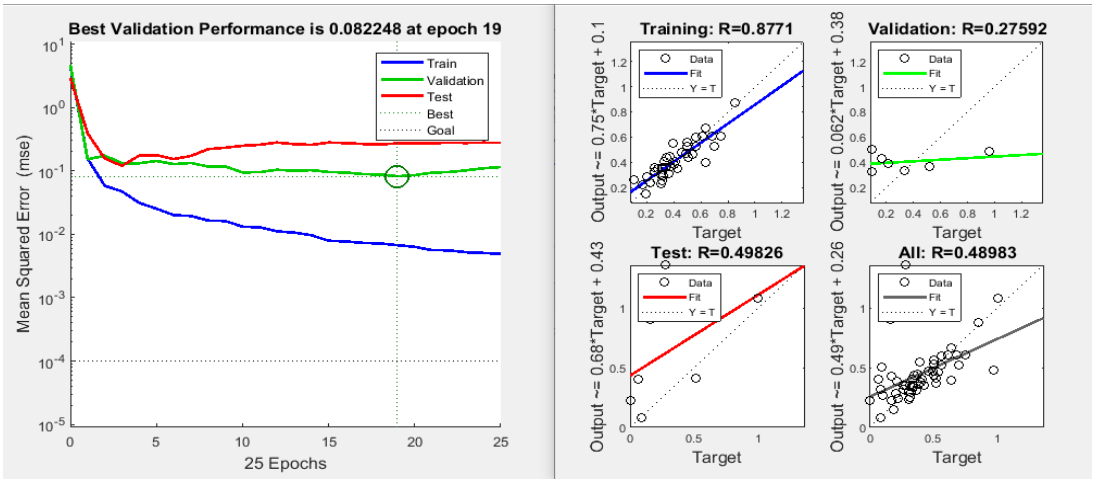


Figure 45: traincgb performance and regression plots for Product 1

- trainbfg

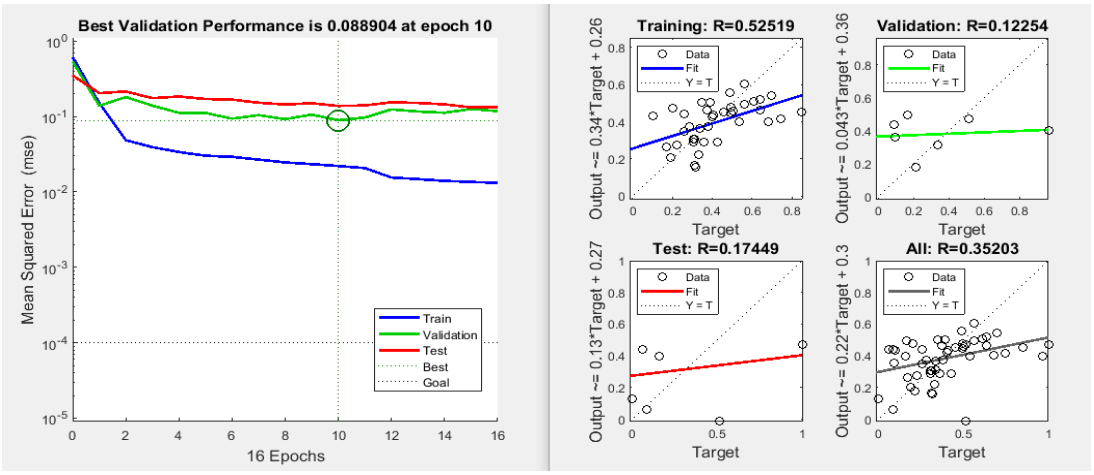


Figure 46: trainbfg performance and regression plots for Product 1

- trainoss

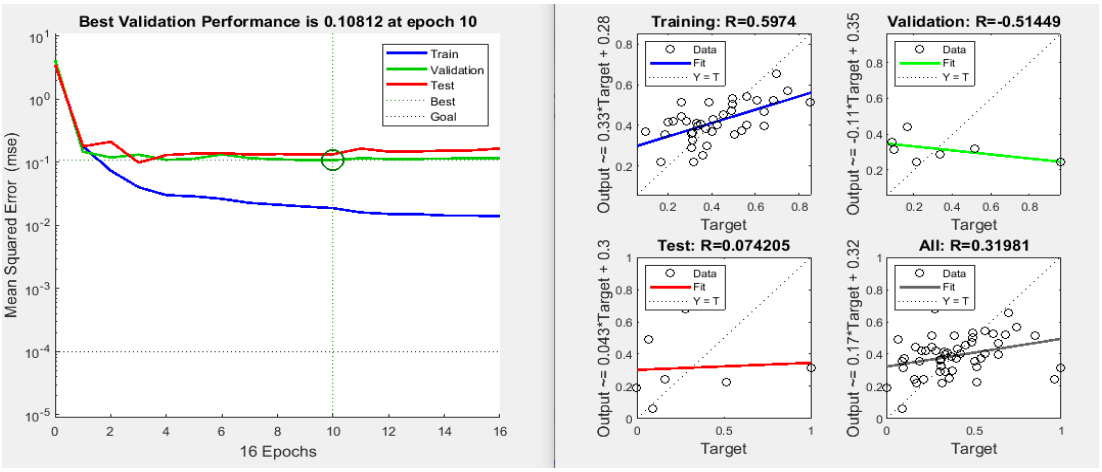


Figure 47: trainoss performance and regression plots for Product 1

- trainlm

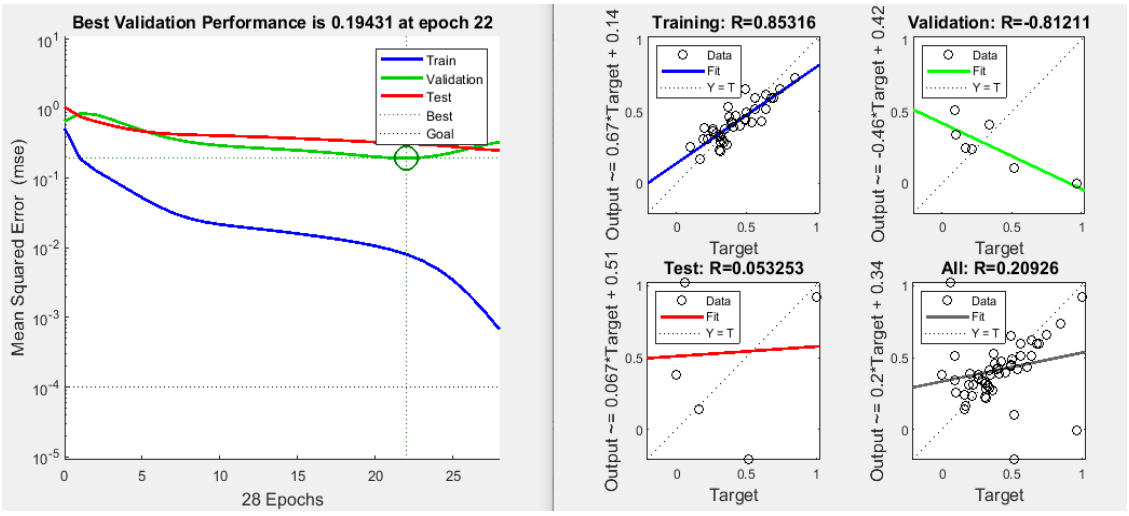


Figure 48: trainlm performance and regression plots for Product 1

Table 5 documents the training, test and total MAD, MSE and RMSE results for Product 1. The training dataset used in the table is the combined training and validation dataset. The number of neurons used can also be seen in the last column.

Table 5: ANN based forecasting results for Product 1

		Train RMSE	Train MAD	Test RMSE	Test MAD	Total MAD	Total MSE	Total RMSE	Neurons in hidden layer
1	traingd	0%	22%	3%	27%	17%	6%	24%	20
2	traingdx	7%	34%	12%	63%	15%	5,60%	23,70%	20
3	traingdm	16%	43%	26,00%	0,25%	20%	9%	30,70%	16

4	trainbr	8,00%	27.4%	15,00%	26,50%	15,77%	4,84%	22,01%	22
5	traincgf	6.37%	28.3%	12,00%	22,17%	13,80%	4,80%	21,90%	16
6	traincgp	6.4%	17.2%	14,00%	13,40%	14,20%	4,50%	21,30%	17
7	traincgb	6.2%	14.8%	16,00%	33,80%	12,90%	5,40%	23,40%	18
8	trainbfg	14.5%	21.5%	6,00%	45,00%	15,50%	4,85%	22,05%	18
9	trainoss	13.6%	34.9%	4,00%	40,70%	16,80%	5,90%	24,30%	18
10	trainlm	4.5%	36.7%	21,00%	15,30%	15,98%	7,75%	27,80%	18

8.1.3 SVR based forecasting for Product 1

- Linear kernel

Table 6 shows the optimization performed for the linear kernel.

Table 6: Linear kernel optimization for Product 1

BoxConstraint/C	Epsilon/ε	Validation error		Comment
		RMSE	MAD	
500	0.001	2.08%	24.9%	
500	0.003	2.2%	27.7%	
500	0.005	2%	27%	
500	0.007	1.8%	27.9%	
500	0.009	1.9%	27.7%	
500	0.01	2.25%	27.7%	
500	0.03	3.09%	27.3%	
500	0.05	10,00%	28.6%	
500	0.07	16.5%	32.28%	
500	0.09	12.43%	31.4%	
500	0.1	10.8%	30.8%	
500	0.3			Almost underfitting
500	0.5			Underfitting
500	0.8			Underfitting

It was observed that the smallest error on the validation set was obtained for a BoxConstraint value of 500 and an Epsilon value of 0.007. Thus, the final SVR model was built from the combined training and validation sets using these values, and tested on the test dataset. Figure 49 displays the plot of the forecast using the linear kernel.

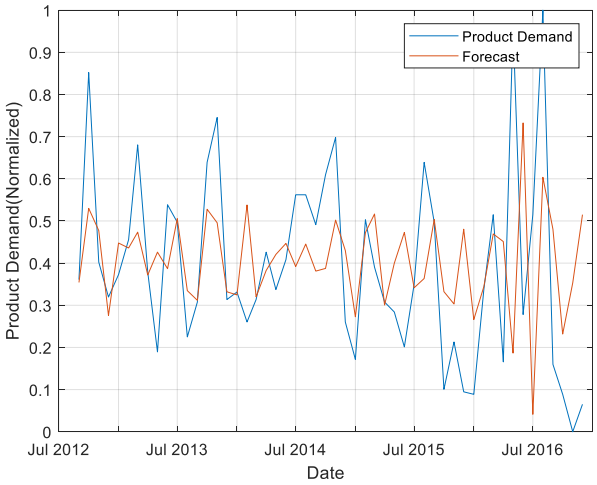


Figure 49: Linear SVR forecast for Product 1

- Gaussian kernel

Table 7 displays the optimization performed for the Gaussian kernel. Various combinations of BoxConstraint, KernelScale and Epsilon were tried out until a good fit was obtained on the training set.

Table 7: Gaussian kernel optimization for Product 1

BoxConstraint/C	**Epsilon/ε**	**KernelScale**	**Validation error**		**Comment**
			RMSE	MAD	
500	0.004	0.1			Underfitting on test and validation dataset
500	0.004	0.3			Underfitting on test and validation dataset
500	0.004	0.5	2.9%	25.8%	
500	0.004	0.6	0.8%	26.4%	
500	0.004	0.7	4.8%	27.6%	
500	0.004	0.9	12.09%	31.9%	
500	0.004	1	15.2%	35,00%	
500	0.004	1.5	25.9%	47.2%	
500	0.004	2	29.8%	53.8%	
500	0.004	2.5	30.9%	46.2%	
500	0.004	3			No longer captures the training set perfectly
500	0.004	4			No longer captures the training set perfectly

It was observed that the smallest error on the validation set was obtained for a BoxConstraint value of 500, an Epsilon value of 0.007 and a KernelScale value of 0.6. Thus, the final SVR model was built from the combined training and validation sets using these values, and tested on the test dataset. Figure 50 displays the plot of the forecast using the Gaussian kernel.

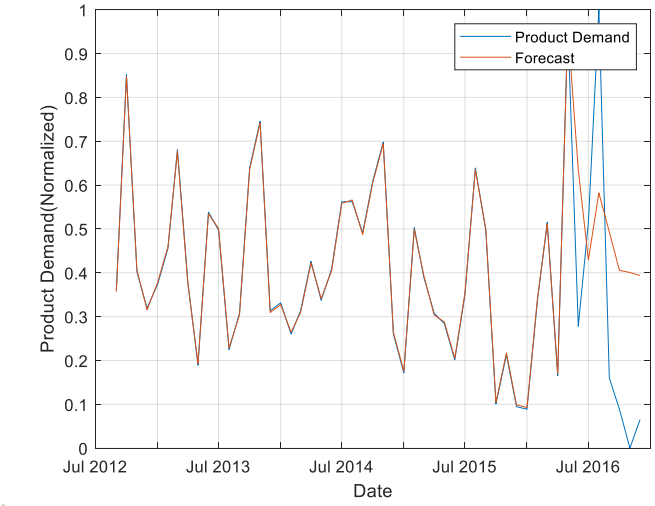


Figure 50: Gaussian SVR forecast for Product 1

- Polynomial kernel

Table 8 displays the optimization performed for the polynomial kernel. Various combinations of PolynomialOrder, BoxConstraint, KernelScale and Epsilon were tried out until a good fit was obtained on the training set.

Table 8: Polynomial order 2 optimization for Product 1

BoxConstraint/C	Epsilon/ε	KernelScale	Validation error		Comment
			RMSE	MAD	
500	0.015	0.01			Does not capture the training set perfectly
500	0.015	0.02	20%	100.6%	
500	0.015	0.04	48%	54%	
500	0.015	0.06	67.3%	73%	
500	0.015	0.08	90.7%	90.7%	
500	0.015	0.1	30.4%	57.3%	
500	0.015	0.3	2.4%	46.37%	
500	0.015	0.5	19.8%	48.7%	
500	0.015	0.7	25,00%	53.5%	
500	0.015	0.9	27,00%	56.2%	
500	0.015	1	28,00%	58%	
500	0.015	1.5			Does not capture the training set perfectly

It was observed that the smallest error on the validation set was obtained for a PolynomialOrder of 2, a BoxConstraint value of 500, an Epsilon value of 0.007 and a KernelScale value of 0.6. Thus, the final SVR model was built from the combined training and validation sets using these values, and tested on the test dataset. Figure 51 displays the plot of the forecast using the Gaussian kernel.

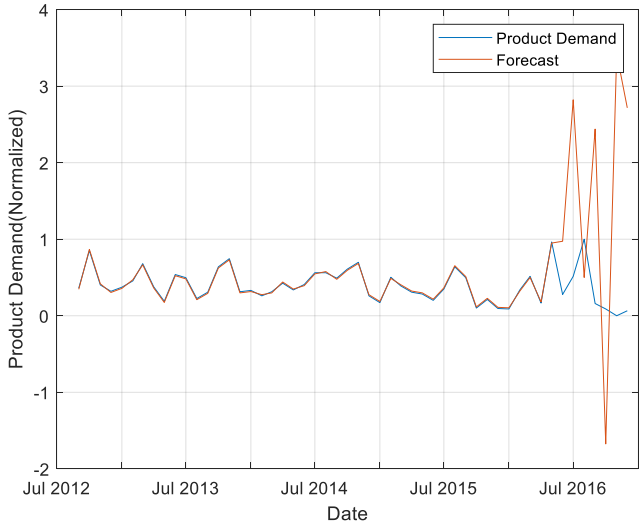


Figure 51: Polynomial SVR forecast for Product 1

Table 9 contains the MAD, MSE and RMSE on training, test and total datasets for the linear, Gaussian and polynomial kernels for Product 1.

Table 9: SVR results for Product 1

	Train RMSE	**Train MAD**	**Test RMSE**	**Test MAD**	**Total MAD**	**Total MSE**	**Total RMSE**
Linear	1.8%	25%	12.3%	37%	16.5%	5.3%	22.9%
Gaussian	0%	0.4%	17.2%	32%	1.55%	4.2%	12.3%
Polynomial order 2	0.92%	1.5%	129%	194.58%	27.5%	63.7%	79.8%

8.1.4 ARIMA based forecasting for Product 1

As described in Section 8.1.1, an ARIMA model of (6,0,4) was selected from the ACF and PACF plots of Product 1.

Figure 52 displays the plot of the forecast using this ARIMA model.

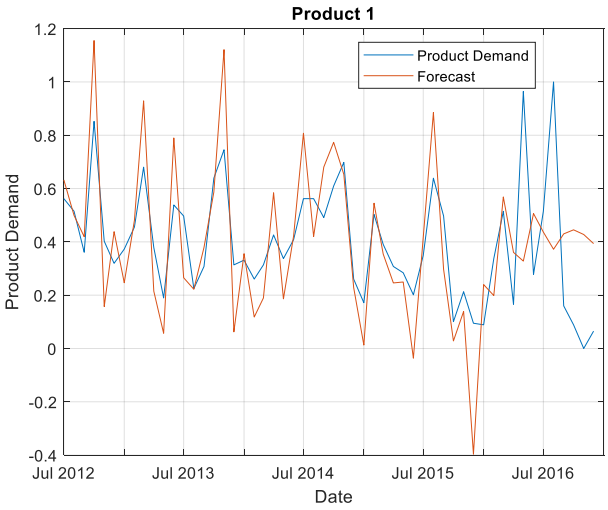


Figure 52: ARIMA forecast for Product 1

Table 10 contains the MAD, MSE and RMSE on training, test and total datasets for the ARIMA forecast for Product 1.

Table 10: ARIMA results for Product 1

	Train RMSE	**Train MAD**	**Test RMSE**	**Test MAD**	**Total MAD**	**Total MSE**	**Total RMSE**
ARIMA (6,0,4)	7.2%	13.8%	15.4%	34.9%	17.4%	0%	0.7%

8.2 Product 2

Figure 53 displays Product 2 as a function of time.

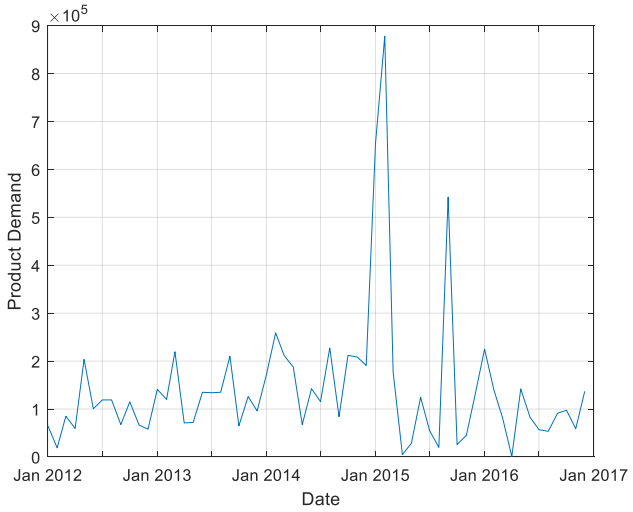


Figure 53: Time series plot of Product 2

8.2.1 Data analysis and pre-processing for Product 2

Figure 54 and 55 display the ACF and PACF plots of Product 2.

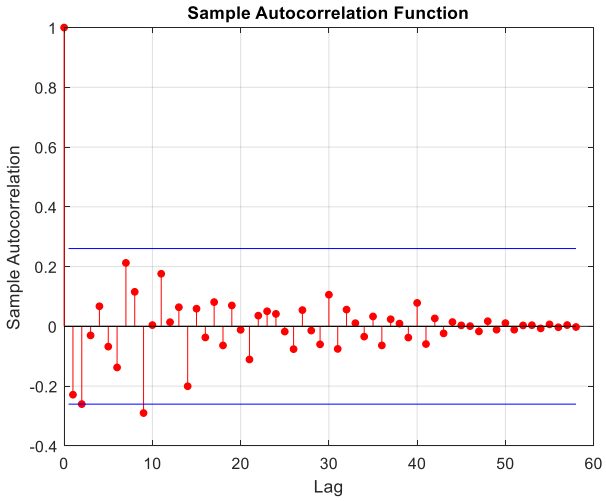


Figure 54: ACF plot of Product 2

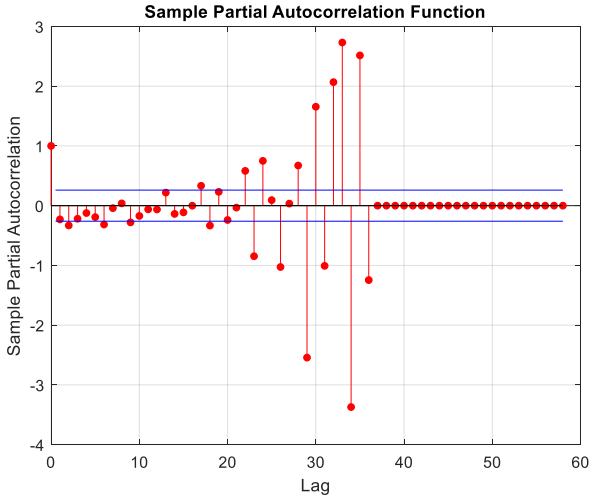


Figure 55: PACF plot of Product 2

From the plots of ACF and PACF no trend and stationarity are visible. Hence no further differencing is required. The significant number of lags was determined from the PACF as 6. This was used as the number of lags for the ANN and SVR models and the order of the AR model.

From the plot of the ACF, the first two lags were deemed significant. Thus, the order of the MA model was chosen to be 2. The final ARIMA model was chosen to be (6,0,2) model.

8.2.2 ANN based forecasting for Product 2

Figures 56 – 65 display the performance and regression plots using different training algorithms for Product 2.

- traingd

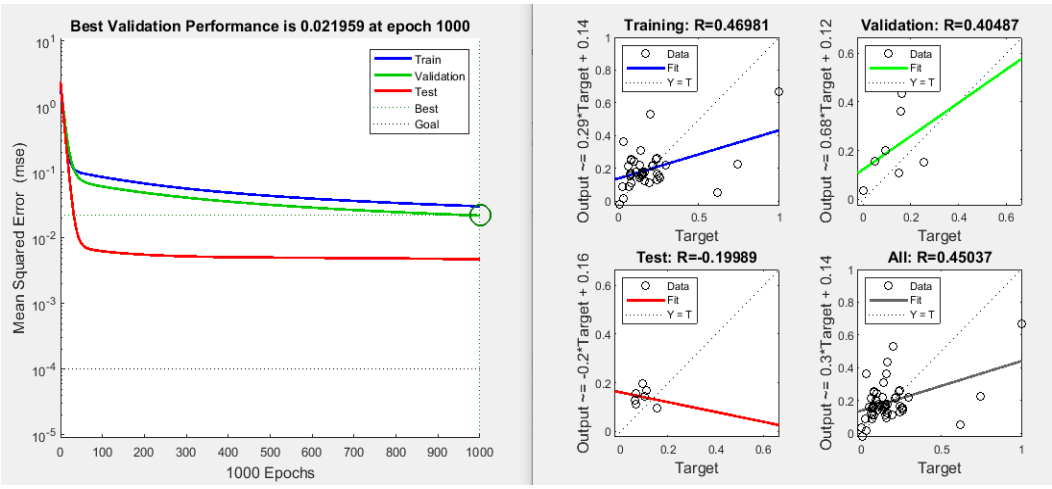


Figure 56: traingd performance and regression plots for Product 2

- traingdx

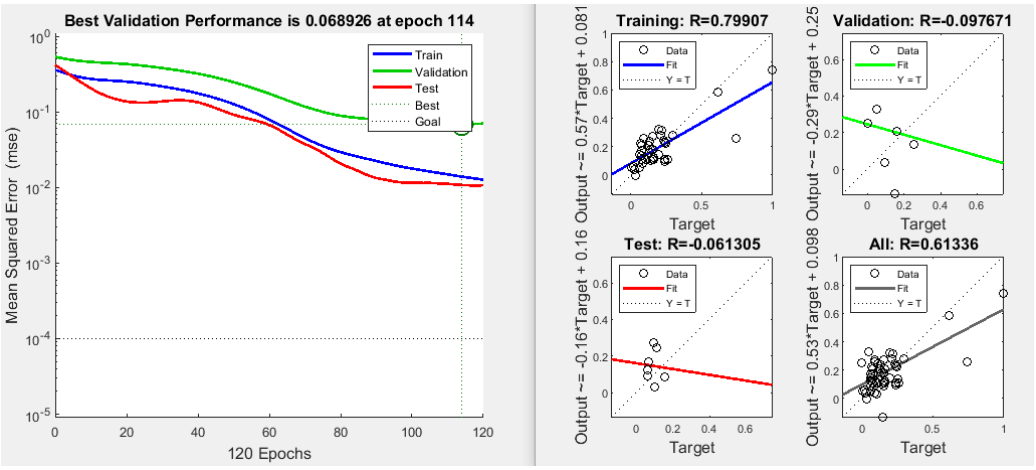


Figure 57: traingdx performance and regression plots for Product 2

- traingdm

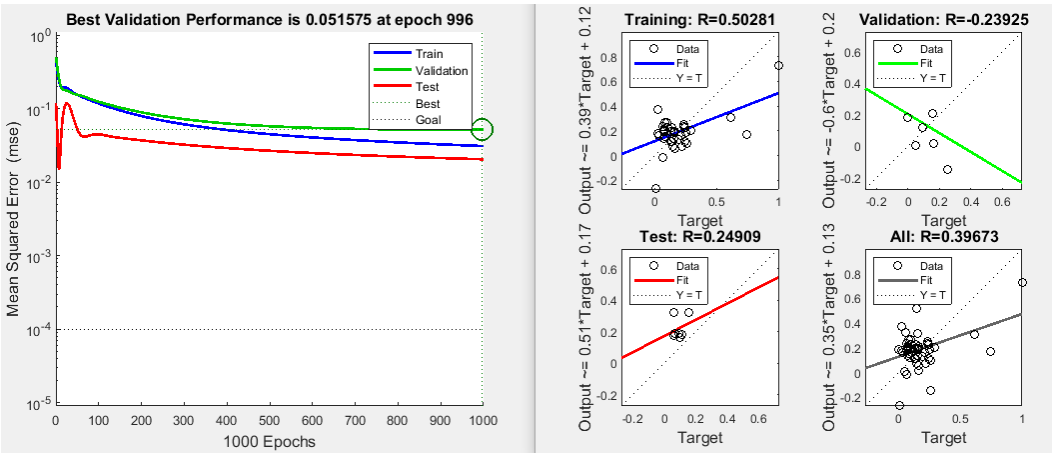


Figure 58: traingdm performance and regression plots for Product 2

- trainbr

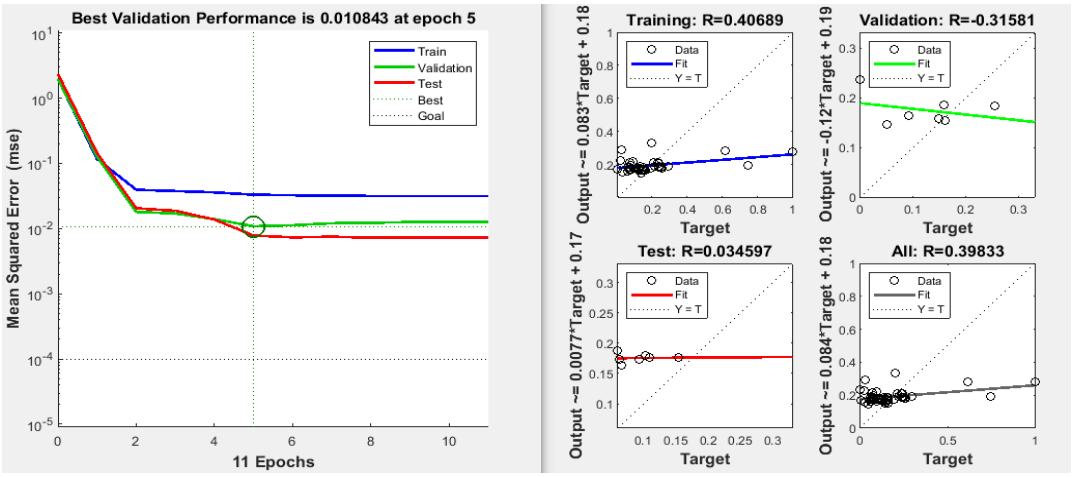


Figure 59: trainbr performance and regression plots for Product 2

- traincgf

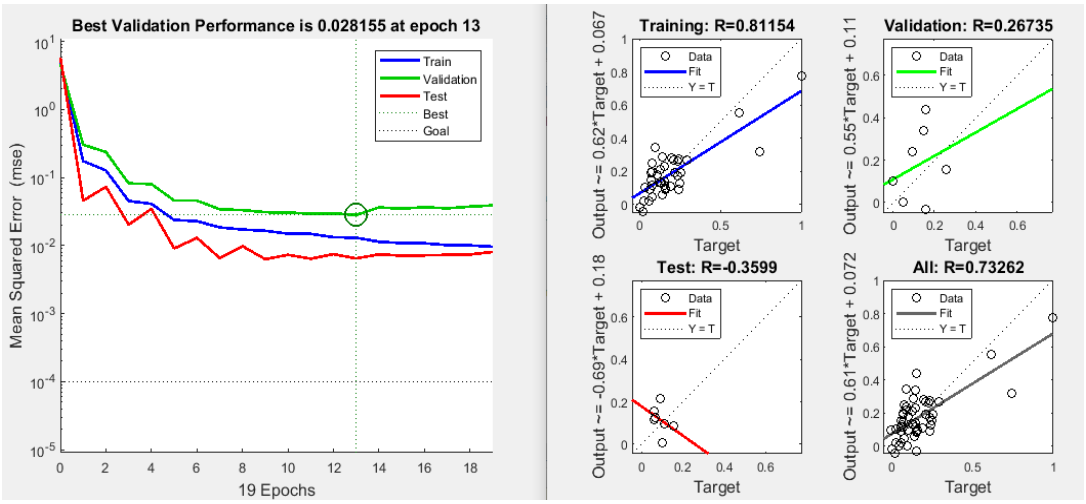


Figure 60: traincgf performance and regression plots for Product 2

- traincgp

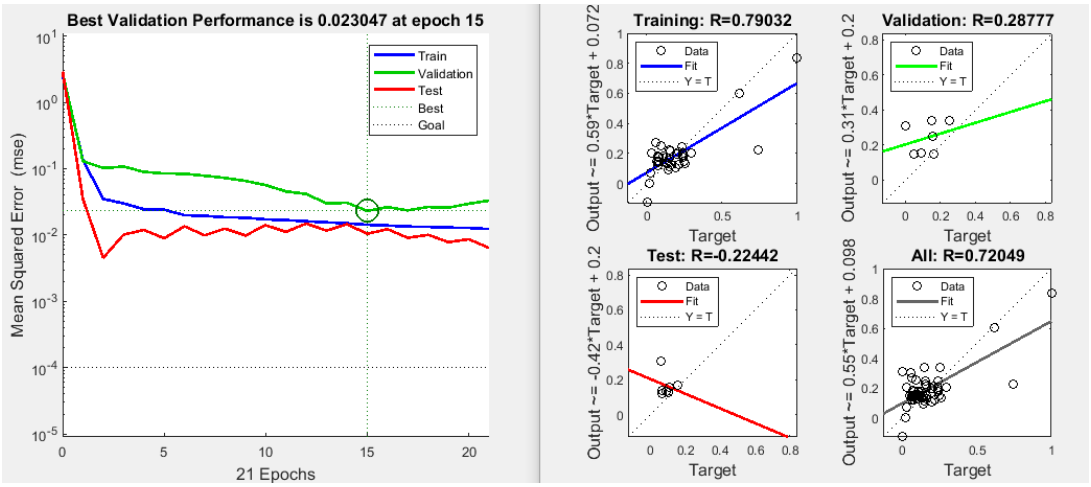


Figure 61: traincgp performance and regression plots for Product2

- traincgb

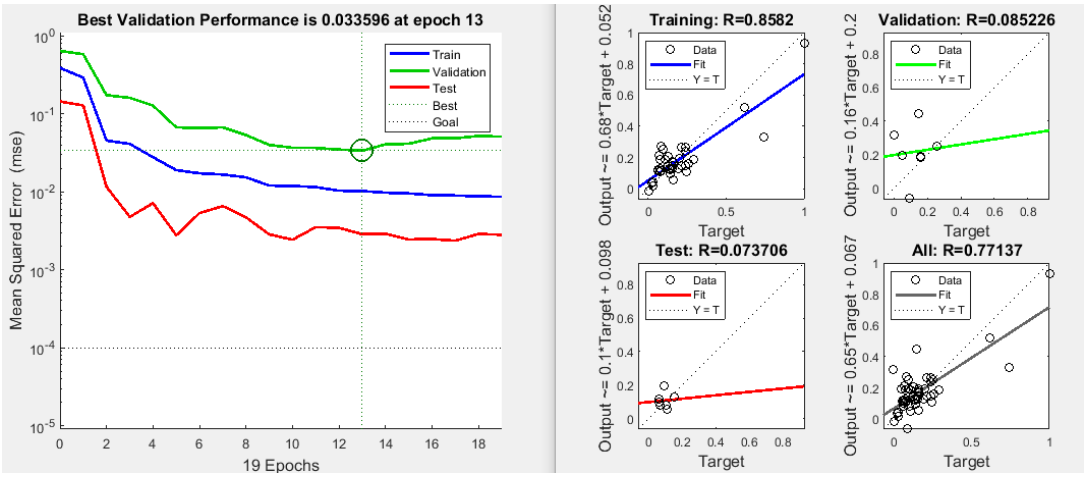


Figure 62: traincgb performance and regression plots for Product 2

- trainbfg

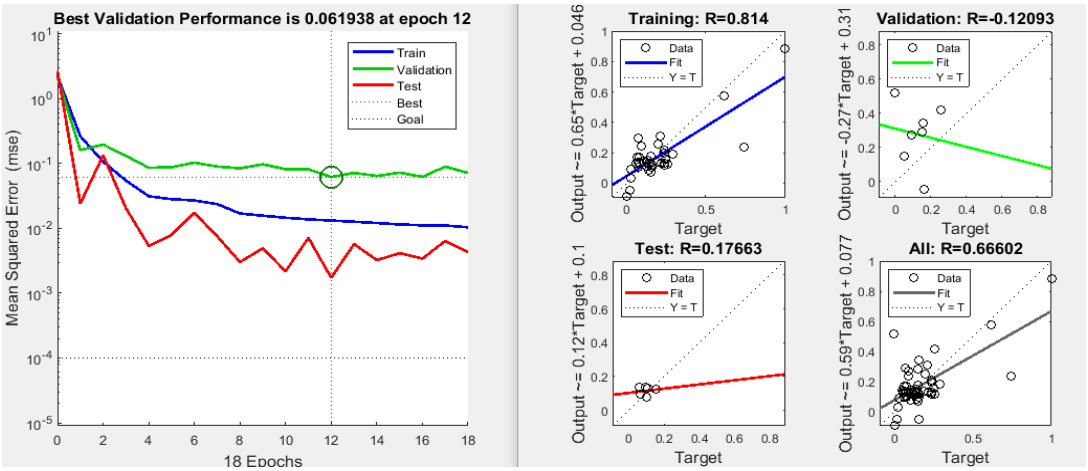


Figure 63: trainbfg performance and regression plots for Product 2

- trainoss

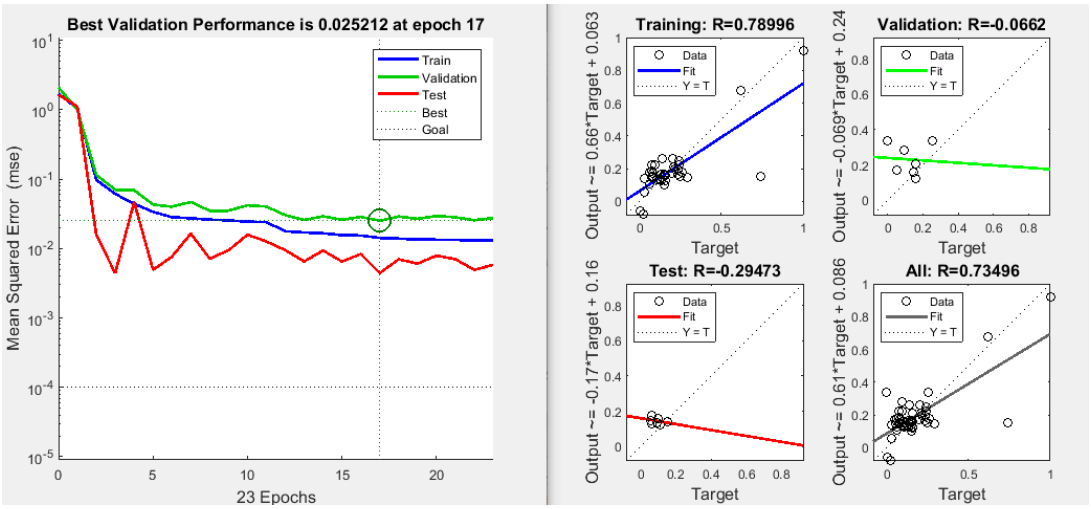


Figure 64: trainoss performance and regression plots for Product 2

- trainlm

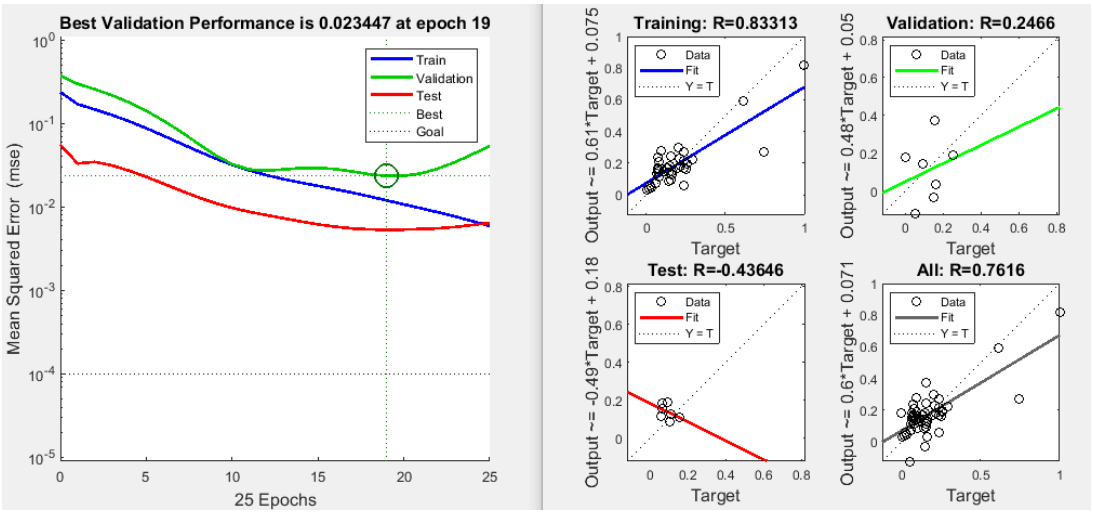


Figure 65: trainlm performance and regression plots for Product 2

Table 11 documents the training, test and total MAD, MSE and RMSE results for Product 2 using the different training algorithms.

Table 11: ANN forecasting results for Product 2

		Train RMSE	Train MAD	Test RMSE	Test MAD	Total MAD	Total MSE	Total RMSE	Number of neurons used
1	traingd	37.3%	41%	34%	48%	11%	3%	16%	20
2	traingdx	17.3%	19%	39,70%	36%	10%	2,70%	14,70%	20
3	traingdm	46.4%	40.1%	118,00%	107,00%	14%	3%	18,30%	16
4	trainbr	1.32%	9.38%	6.7%	8.2%	10.3%	2.67%	16.34%	22
5	traincgf	19.6%	24.4%	3,67%	4,00%	9,04%	1,47%	12,14%	16
6	traincgp	32.5%	41,00%	54,50%	60,00%	8,47%	1,55%	12,46%	17
7	traincgb	24.7%	28.1%	5,02%	7,00%	7,55%	1,28%	11,23%	18
8	trainbfg	35.9%	36,00%	12,55%	15,00%	9,09%	1,90%	13,70%	18
9	trainoss	25.7%	22.4%	39,08%	41,00%	7,64%	1,50%	12,24%	18
10	trainlm	21.4%	26,00%	28,90%	32,00%	8,03%	1,30%	11,40%	18

8.2.3 SVR based forecasting for Product 2

- Linear kernel

Table 12 displays the optimization performed for the linear kernel.

Table 12: Linear optimisation for Product 2

BoxConstraint/C	Epsilon/ε	Validation error		Comment
		RMSE	MAD	
400	0.005	2.15%	16.33%	
400	0.008	1.7%	20.9%	
400	0.01	2.18%	16.25%	
400	0.03	1.02%	17.9%	
400	0.05	0.4%	16,00%	
400	0.08	0.3%	15.02%	
400	0.09	0,00%	14.8%	
400	0.1	0.18%	15.1%	
400	0.2	0.9%	18.28%	
400	0.3	7.2%	29.3%	
400	0.4			Very large errors

It was observed that the smallest error on the validation set was obtained for a BoxConstraint of 400 and an Epsilon value of 0.09. Figure 66 displays the plot of the forecast using the linear kernel.

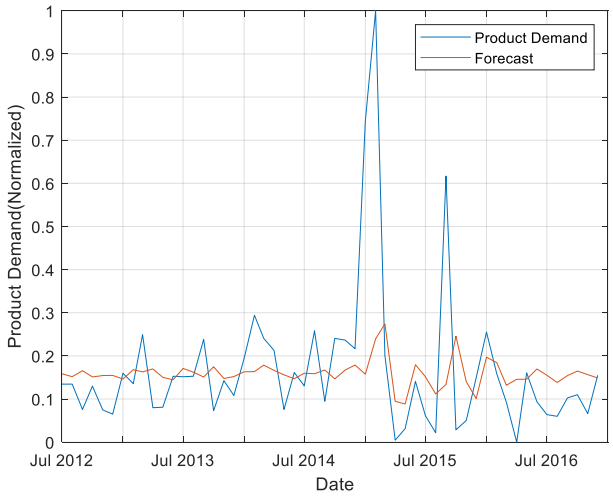


Figure 66: Linear SVR forecast for Product 2

- Gaussian kernel

Table 13 displays the optimisation performed for the Gaussian kernel for Product 2.

Table 13: Gaussian kernel optimisation for Product 2

BoxConstraint/C	**Epsilon/ε**	**KernelScale**	**Validation error**		**Comment**
			RMSE	MAD	
500	0.004	0.1			Underfitting on test and validation dataset
500	0.004	0.2	0.9%	12.26%	

500	0.004	0.3	0.3%	13.05%	
500	0.004	0.4	2.2%	20.17%	
500	0.004	0.5	6.4%	32.6%	
500	0.004	0.6	12.4%	45.7%	
501	0.004	0.7	0.5%	51.5%	
500	0.004	0.8	4.37%	49.7%	
500	0.004	0.9	12.09%	31.9%	
500	0.004	1			No longer captures the training set perfectly
500	0.004	1.5			No longer captures the training set perfectly

It was observed that the lowest error on the validation dataset was obtained for a BoxConstraint of 500, Epsilon of 0.004 and KernelScale of 0.3. Figure 67 displays the plot of the forecast using the Gaussian kernel for Product 2.

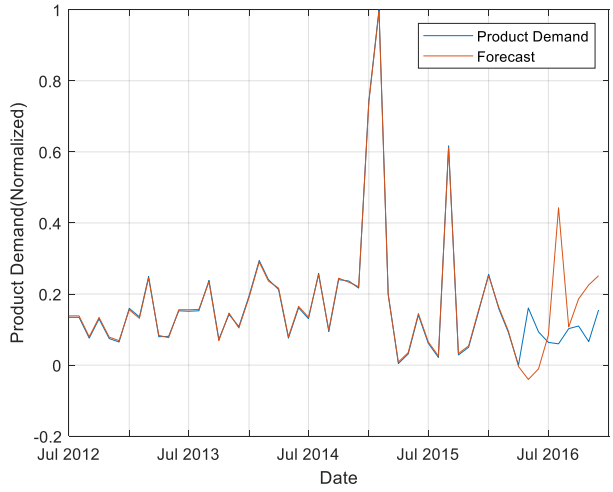


Figure 67: Gaussian SVR forecast for Product 2

- Polynomial kernel

Table 14 displays the optimisation performed for the polynomial kernel for Product 2.

Table 14: Polynomial optimisation for Product 2

BoxConstraint/C	Epsilon/ε	KernelScale	Validation error		Comment
			RMSE	MAD	
500	0.016	0.06			Does not capture the training set perfectly
500	0.016	0.08	96.5%	107.5%	
500	0.016	0.1	92.4%	107.4%	
500	0.016	0.3	64.2%	98.9%	
500	0.016	0.5	54.37%	94.2%	
500	0.016	0.6	52.6%	94.3%	

500	0.016	0.7	26.5%	88.04%	
500	0.016	0.8	20.2%	76.8%	
500	0.016	0.9	28.9%	63.74%	
500	0.016	1	31.8%	60.4%	
500	0.016	1.5			Does not capture the training set perfectly

It was observed that the smallest error on the validation set was obtained for a polynomial order of 3, BoxConstraint of 500, Epsilon of 0.016 and KernelScale of 0.8. Figure 68 displays the plot of the forecast using the polynomial kernel for Product 2.

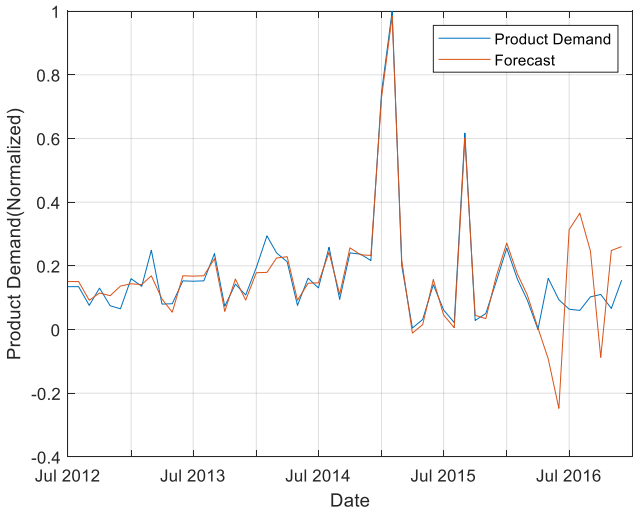


Figure 68: Polynomial SVR forecast for Product 2

Table 15 contains the MAD, RMSE and MSE on the training, test and total datasets for the linear, Gaussian and polynomial kernels for Product 2.

Table 15: SVR forecast results for Product 2

	Train RMSE	**Train MAD**	**Test RMSE**	**Test MAD**	**Total MAD**	**Total MSE**	**Total RMSE**
Linear	39%	42%	45%	41%	53%	25%	60%
Gaussian	0%	0%	5.4%	13.07%	2.27%	0.4%	6.6%
Polynomial order 3	0%	2.05%	2.4%	22.2%	5.04%	0.8%	9.4%

8.2.4 ARIMA based forecasting for Product 2

As described in Section 8.2.1, an ARIMA model of (6,0,2) was selected from the ACF and PACF plots of Product 2.

Figure 69 displays the plot of the forecast using this ARIMA model.

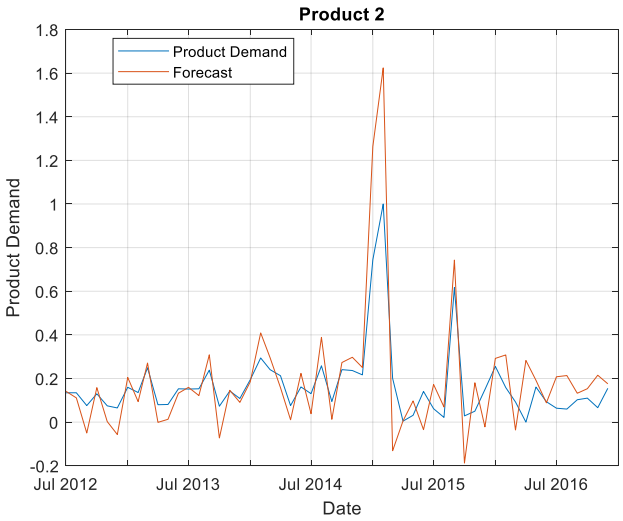


Figure 69: ARIMA forecast for Product 2

Table 16 contains the MAD, MSE and RMSE on training, test and total datasets for the ARIMA forecast for Product 2.

Table 16: ARIMA results for Product 2

	Train RMSE	Train MAD	Test RMSE	Test MAD	Total MAD	Total MSE	Total RMSE
ARIMA (6,0,2)	6.1%	13.8%	28.3%	9.5%	10.04%	17.6%	41.9%

8.3 Product 3

Figure 70 shows Product 3 plotted as a function of time.

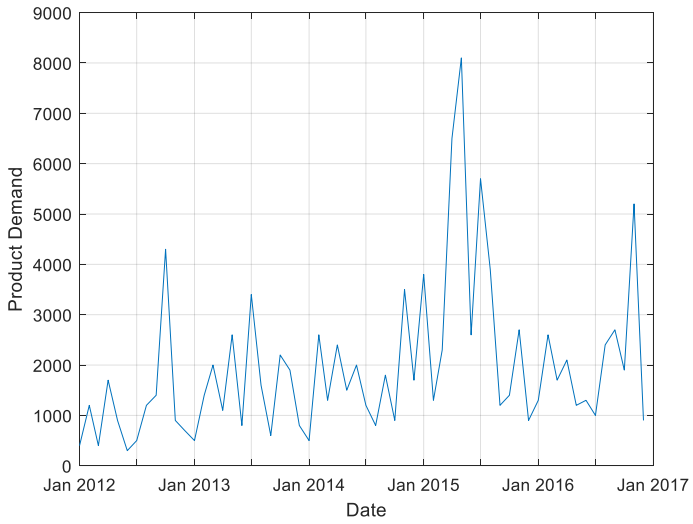


Figure 70: Time series plot of Product 3

8.3.1 Data analysis and pre-processing for Product 3

Figure 71 and 72 display the ACF and PACF plots of Product 3

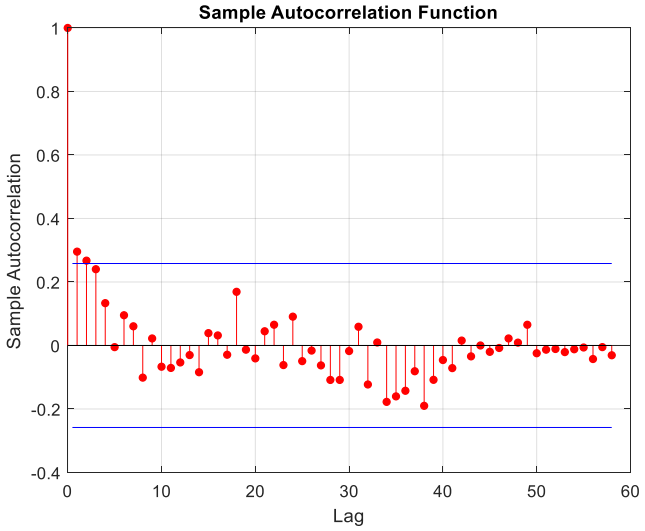


Figure 71: ACF plot of Product 3

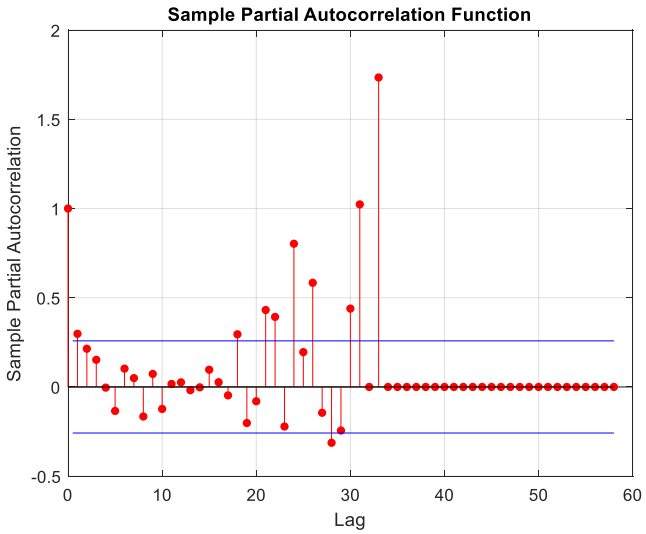


Figure 72: PACF plot of Product 3

An examination of the ACF and PACF plots demonstrates slight trend with no seasonality. Thus, a degree of differencing of at least one is required. From the PACF plot, the first three lags were deemed significant, hence the number of lags of ANN and SVR models were chosen to be three. The order of the ARIMA AR model was also chosen to be three.

From the ACF plot, the first four lags were deemed significant; thus, the order of MA model was chosen as four. The ARIMA model was chosen to be a (3,1,4) model

8.3.2 ANN based forecasting for Product 3

Figures 73 – 82 display the performance and regression plots for Product 3 using the different training algorithms.

- traingd

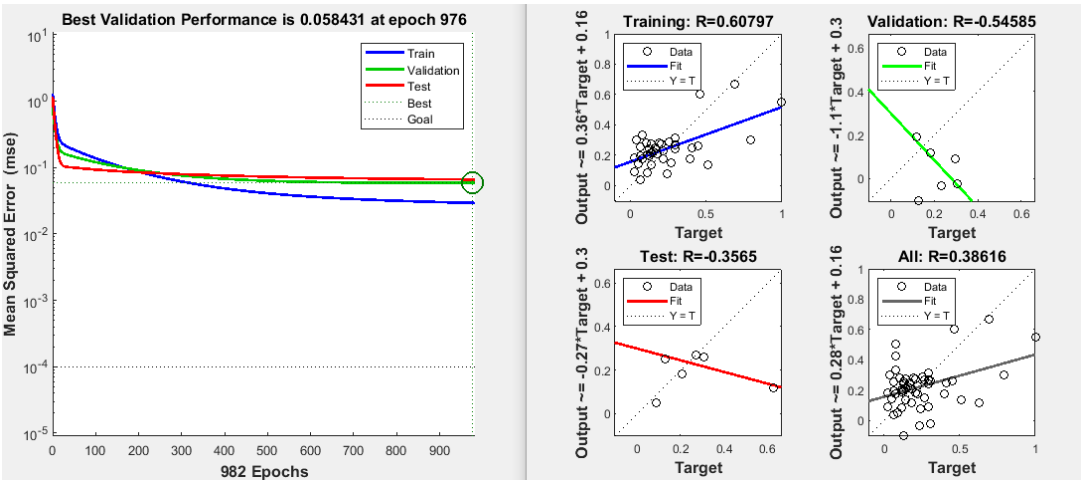


Figure 73: traingd performance and regression plots for Product 3

- traingdx

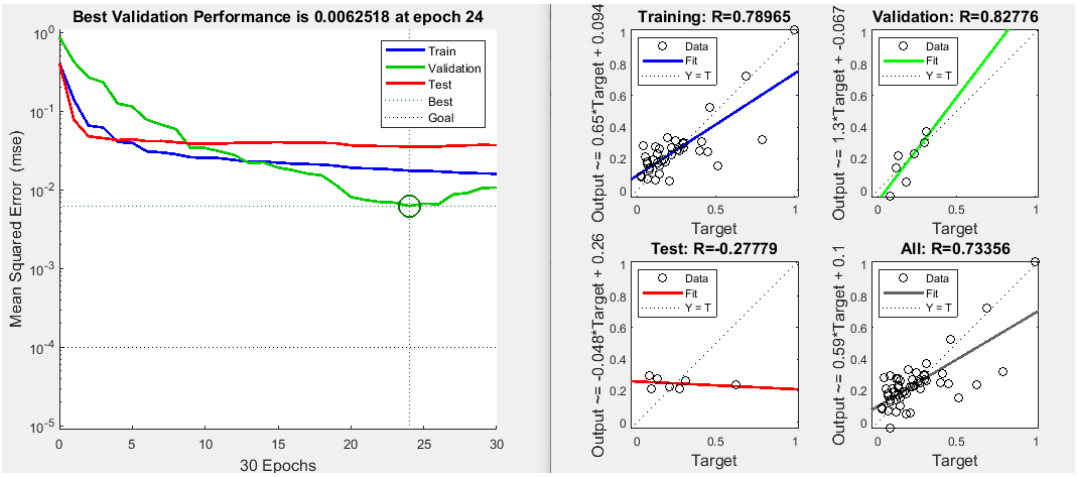


Figure 74: traingdx performance and regression plots for Product 3

- traingdm

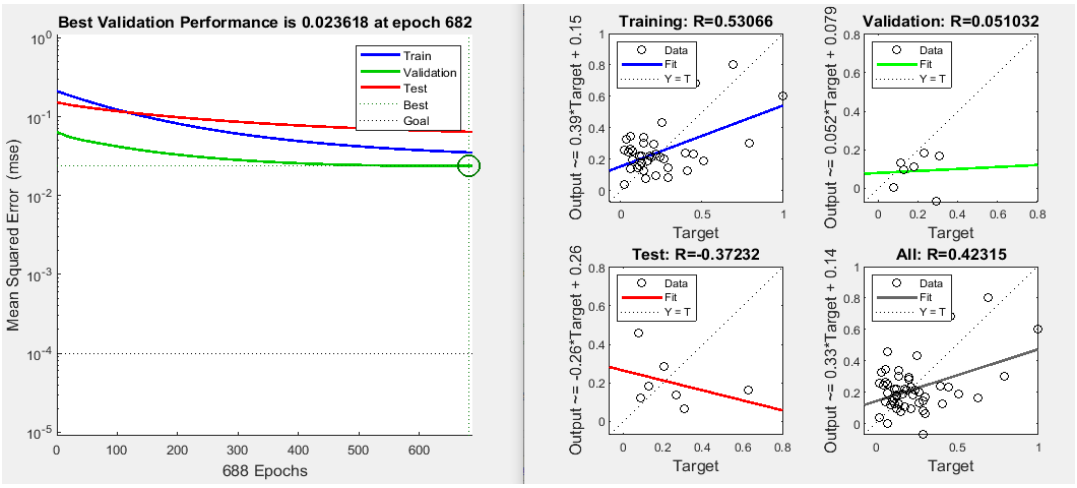


Figure 75: traingdm performance and regression plots for Product 3

- trainbr

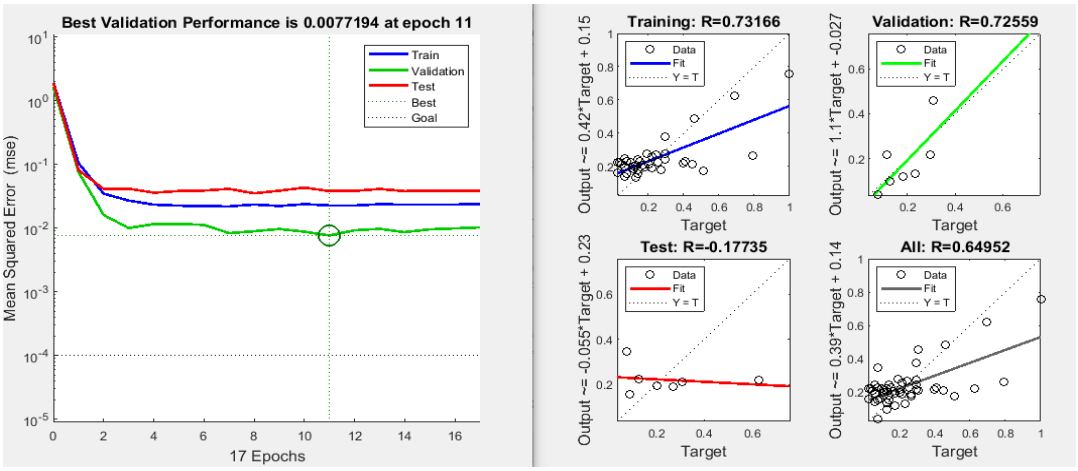


Figure 76: trainbr performance and regression plots for Product 3

- traincgf

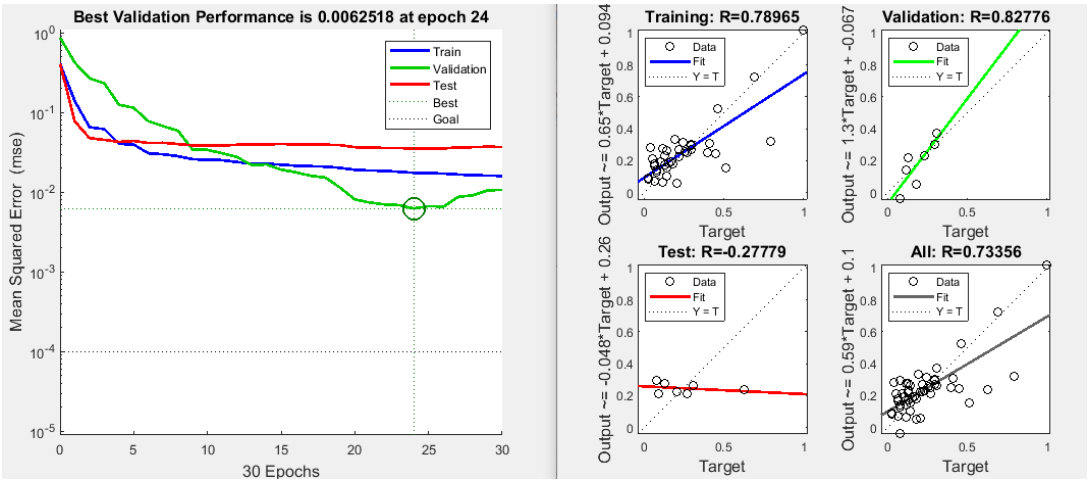


Figure 77: traincgf performance and regression plots for Product 3

- traincgp

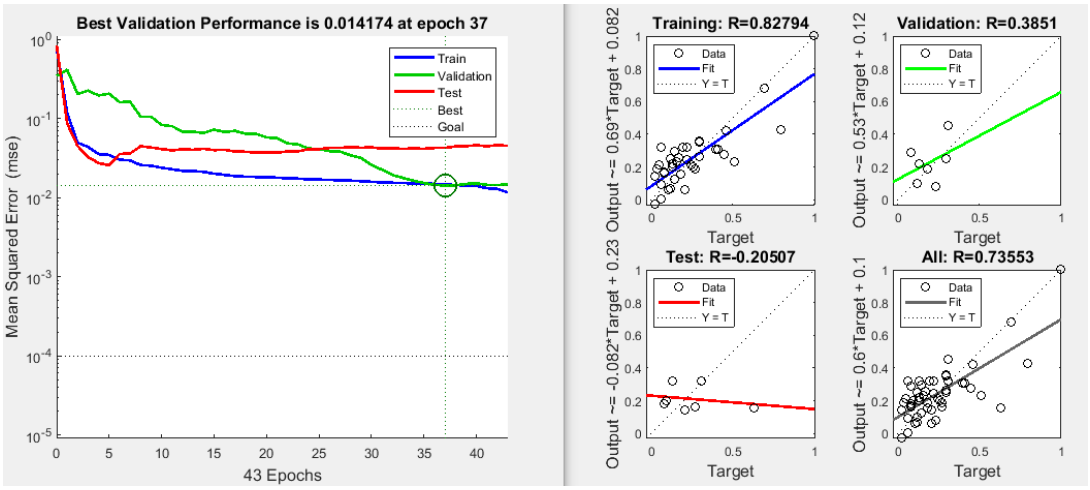


Figure 78: traincgp performance and regression plots for Product 3

- traincgb

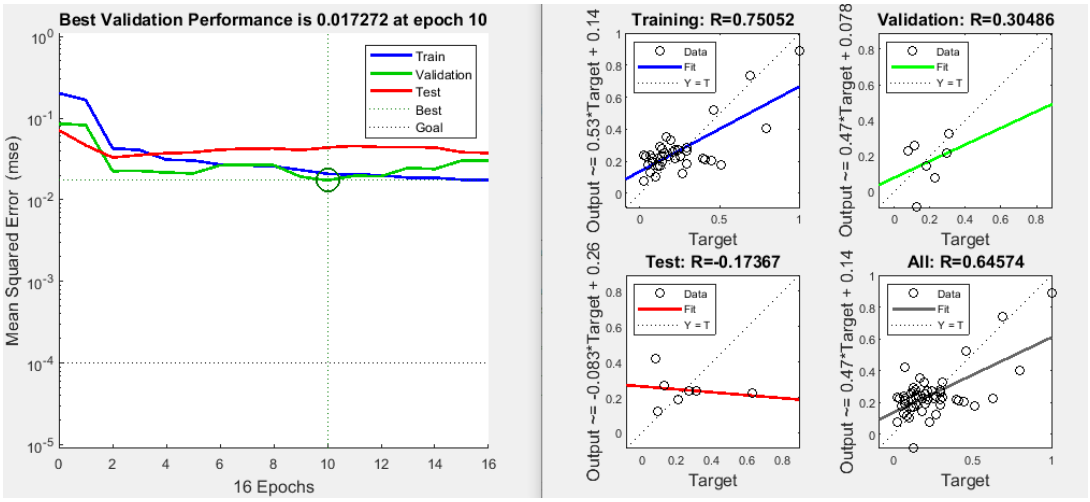


Figure 79: traincgb performance and regression plots for Product 3

- trainbfg

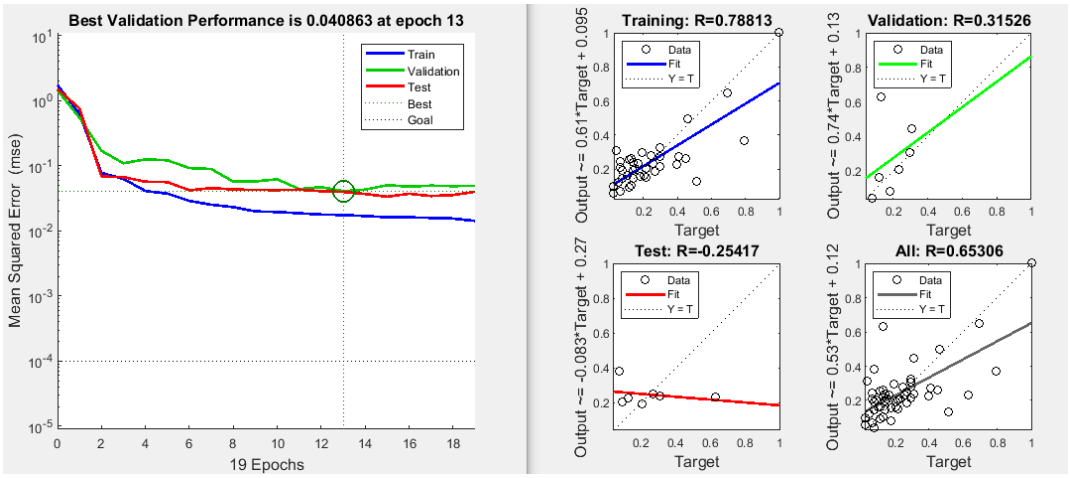


Figure 80: trainbfg performance and regression plots for Product 3

- trainoss

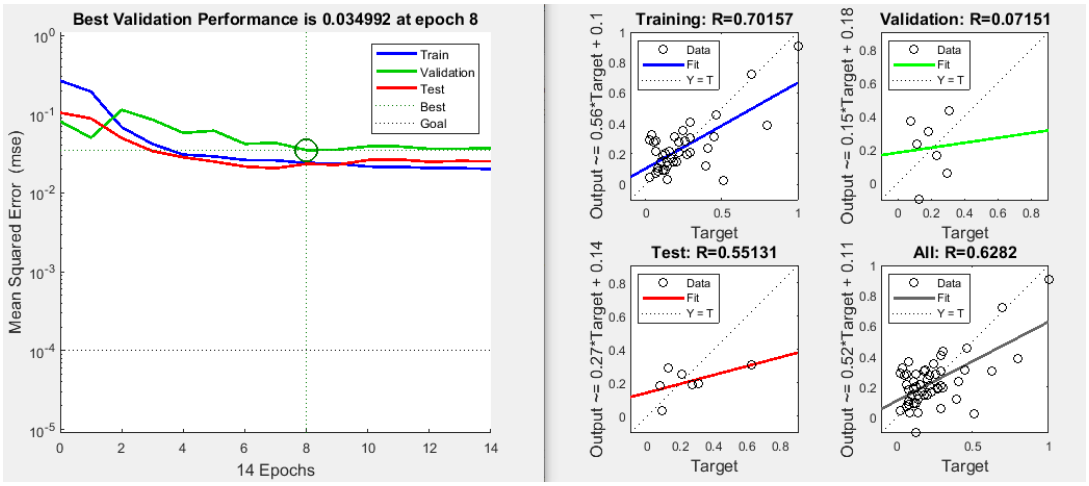


Figure 81: trainoss performance and regression plots for Product 3

- trainlm

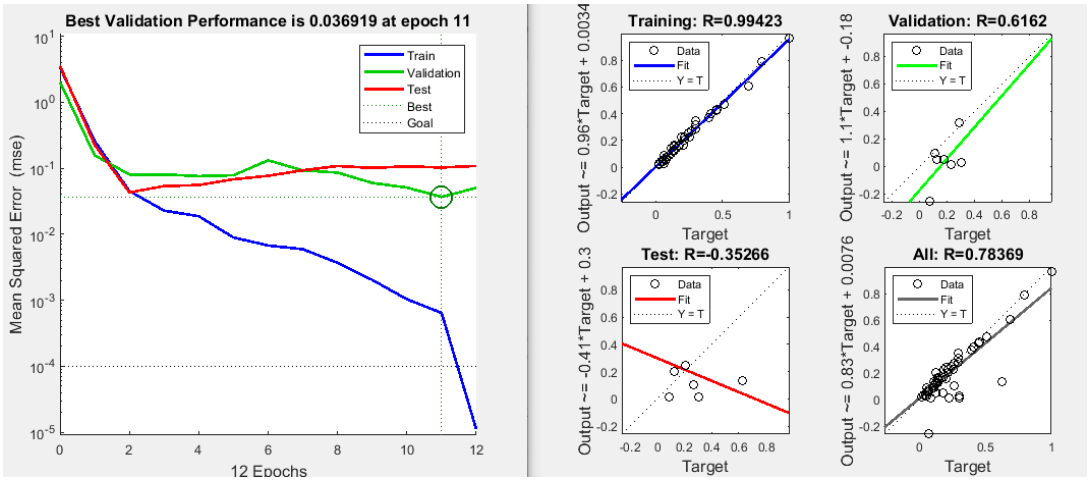


Figure 82: trainlm performance and regression plots for Product 3

Table 17 contains the MAD, RMSE and MSE on the training and test datasets for Product 3.

Table 17: ANN forecasting results for Product 3

	Train RMSE	**Train MAD**	**Test RMSE**	**Test MAD**	**Total MAD**	**Total MSE**	**Total RMSE**	**Number of neurons used**
traingd	3.5%	17.7%	0%	16.7%	14.4%	3.8%	19.5%	20
traingdx	19%	21%	28%	31%	12%	3%	18%	20
traingdm	43%	45%	54,80%	57,00%	15%	4%	19,00%	16
trainbr	16,00%	16.2%	42,40%	44,00%	11%	2,40%	15,40%	22

traincgf	6.5%	11,00%	35,30%	37,00%	9,30%	1,90%	13,80%	16
traincgp	10.17%	12.9%	46,06%	49,00%	10,00%	1,90%	13,80%	17
traincgb	17.5%	22.1%	37,88%	38,00%	11,00%	2,40%	15,60%	18
trainbfg	24.3%	25,00%	36,70%	37,00%	10,40%	2,40%	15,70%	18
trainoss	5,40%	15,00%	35,60%	36.3%	11,60%	2,60%	16,20%	18
trainlm	18.7%	36,00%	61,03%	64,00%	6,80%	2,00%	14,15%	18

8.3.3 SVR based forecasting for Product 3

- Linear kernel

Table 18 shows the optimisation performed for the linear kernel.

Table 18: Linear optimisation for Product 3

BoxConstraint/C	Epsilon/ε	Validation error		Comment
		RMSE	MAD	
400	0.001	0.1%	4.6%	
400	0.003	0.3%	4.7%	
400	0.005	0.5%	4.7%	
400	0.006	0.6%	4.7%	
400	0.008	0.83%	4.5%	
400	0.009	0.9%	4.5%	
400	0.01	1,00%	4.2%	
400	0.03	0.3%	4.4%	
400	0.05	0.6%	5.3%	
400	0.08	0.8%	5.7%	
400	0.1	0.8%	5.3%	
400	0.3	12.8%	13.8%	
400	0.5	14,00%	15,00%	

It was observed that the smallest error on the validation set was obtained for a BoxConstraint value of 400 and an Epsilon value of 0.03. Thus, the final SVR model was built from the combined training and validation sets using these values, and tested on the test dataset. Figure 83 displays the plot of the forecast using the linear kernel.

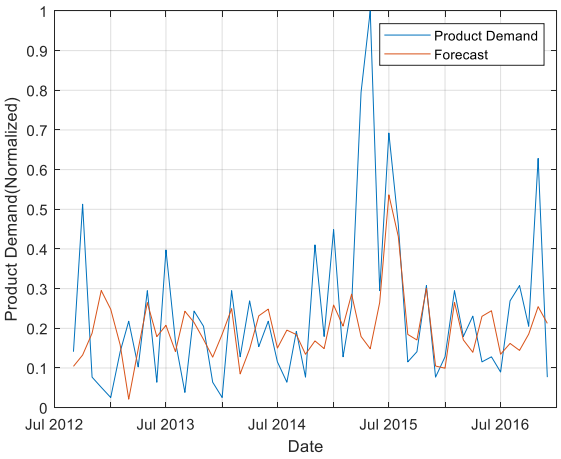


Figure 83: Linear SVR forecast for Product 3

- Gaussian kernel

Table 19 displays the optimisation performed for the Gaussian kernel.

Table 19: Gaussian kernel optimisation for Product 3

BoxConstraint/C	**Epsilon/ε**	**KernelScale**	**Validation error**		**Comment**
			RMSE	MAD	
500	0.005	0.001			Underfitting on test and validation dataset
500	0.005	0.01			Underfitting on test and validation dataset
500	0.005	0.05			Underfitting on test and validation dataset
500	0.005	0.1			Underfitting on test and validation dataset
500	0.005	0.5	3.08%	7.9%	
500	0.005	0.6	1.6%	6.1%	
500	0.005	0.7	0.6%	6.8%	
500	0.005	0.8	3.4%	10.2%	
500	0.005	0.9	7%	15.7%	
500	0.005	1	11.3%	21.6%	
500	0.005	1	53%	53%	

It was observed that the smallest error on the validation set was obtained for a BoxConstraint value of 500, an Epsilon value of 0.005 and a KernelScale value of 0.7. Figure 84 displays the plot of the forecast using the Gaussian kernel.

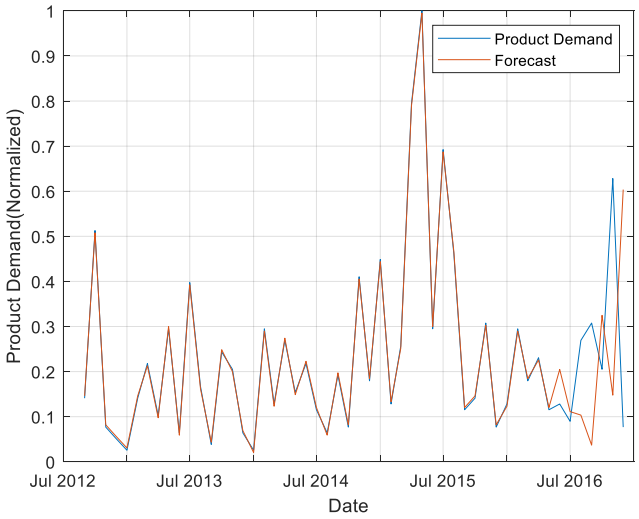


Figure 84: Gaussian SVR forecast for Product 3

- Polynomial kernel

Table 20 displays the optimisation performed for the polynomial kernel.

Table 20: Polynomial optimisation for Product 3

BoxConstraint/C	Epsilon/ε	KernelScale	Validation error		Comment
			RMSE	MAD	
500	0.053				
500	0.053	0.06	108,00%	136.4%	
500	0.053	0.07	90,00%	120.4%	
500	0.053	0.08	75%	106%	
500	0.053	0.09	62.2%	95.9%	
500	0.053	0.1	50.8%	86.9%	
500	0.053	0.15	7.9%	65.4%	
500	0.053	0.2	24.2%	64.5%	
500	0.053	0.25	49.7%	79.4%	
500	0.053	0.3	69.3%	90.9%	
500	0.053	0.4	97.6%	108%	
500	0.053	0.5	116.8%	122.17%	

It was observed that the smallest validation error was obtained for a PolynomialOrder of 2, BoxConstraint of 500, Epsilon of 0.053 and a KernelScale of 0.15. Figure 85 displays the forecast using the polynomial kernel.

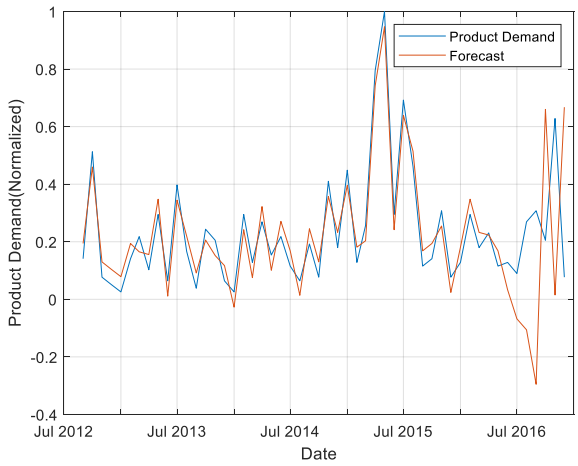


Figure 85: Polynomial SVR forecast for Product 3

Table 21 contains the MAD, MSE and RMSE on training, test and total datasets for the linear, Gaussian and polynomial kernels for Product 3.

Table 21: SVR results for Product 3

	Train RMSE	Train MAD	Test RMSE	Test MAD	Total MAD	Total MSE	Total RMSE
Linear	0.3%	9.4%	5.2%	13.7%	12%	3.7%	19.4%
Gaussian	0%	0%	2.4%	23.7%	3.6%	1.2%	11%
Polynomial	1.4%	4.6%	11.4%	41.2%	10%	3.06%	17.5%

8.3.4 ARIMA forecasting for Product 3

As described in Section 8.3.1, an ARIMA model of (6,0,2) was selected from the ACF and PACF plots of Product 3.

Figure 86 displays the plot of the forecast using this ARIMA model.

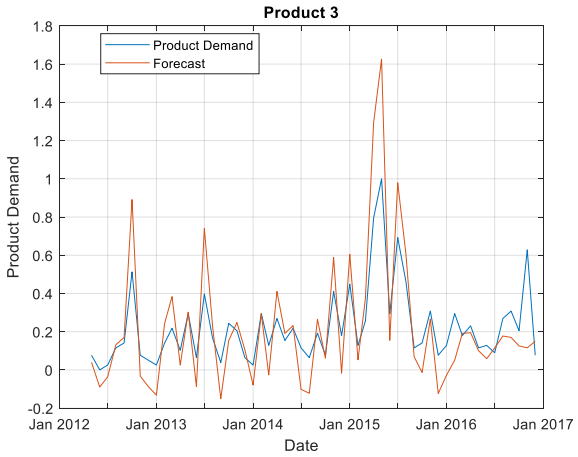


Figure 86: ARIMA forecast for Product 3

Table 22 contains the MAD, MSE and RMSE on training, test and total datasets using the ARIMA method for Product 3.

Table 22: ARIMA results for Product 3

	Train RMSE	Train MAD	Test RMSE	Test MAD	Total MAD	Total MSE	Total RMSE
ARIMA (3,1,4)	12.47%	13.86%	35.57%	12.1%	13.48%	1.3%	11.7%

8.4 Product 4

Figure 87 displays Product 4 plotted as a function of time.

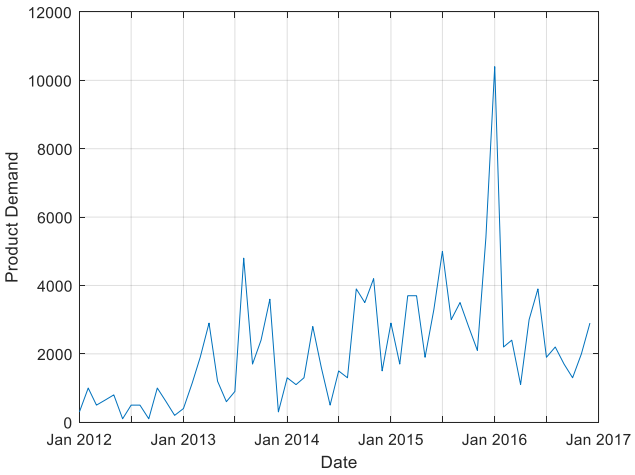


Figure 87: Time series plot of Product 4

8.4.1 Data analysis and pre-processing for Product 4

Figure 88 and 89 display the ACF and PACF of Product 4.

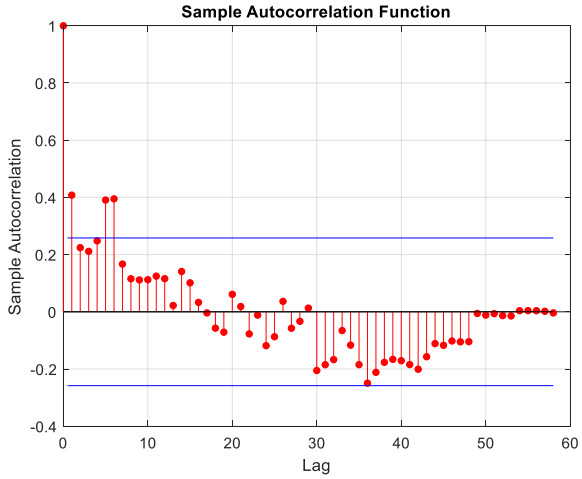


Figure 88: ACF of Product 4

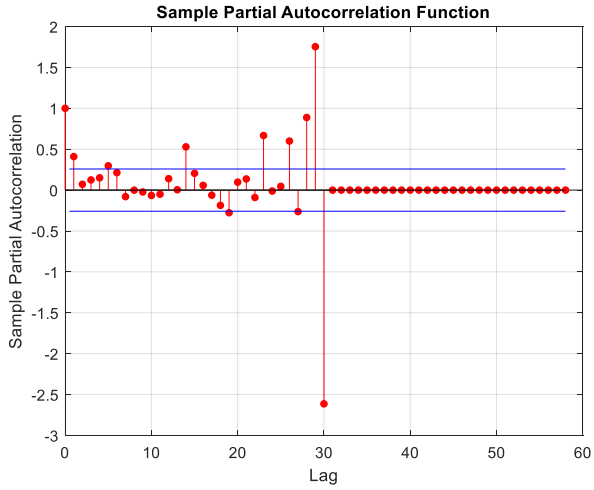


Figure 89: PACF of Product 4

An analysis of the ACF displays trend but no seasonality. Therefore, differencing of at least an order of one is required in the ARIMA model.

Upon examination of the PACF, the first six lags were deemed as significant. Thus, six lags were chosen for the ANN and SVR models. A lag order of six was also chosen for the ARIMA AR model.

From the ACF, the first six lags were deemed significant. Thus, a lag order of six was chosen for the ARIMA MA model. The final ARIMA model is thus a (6,1,6) model.

8.4.2 ANN based forecasting for Product 4

Figures 90 – 99 display the performance and regression plots using the different training algorithms for Product 4.

- traingd

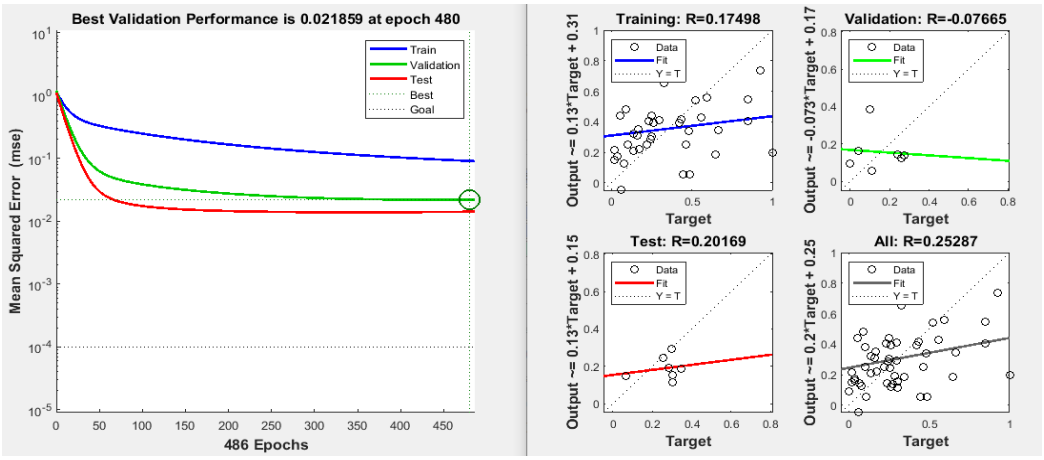


Figure 90: traingd performance and regression plots for Product 4

- traingdx

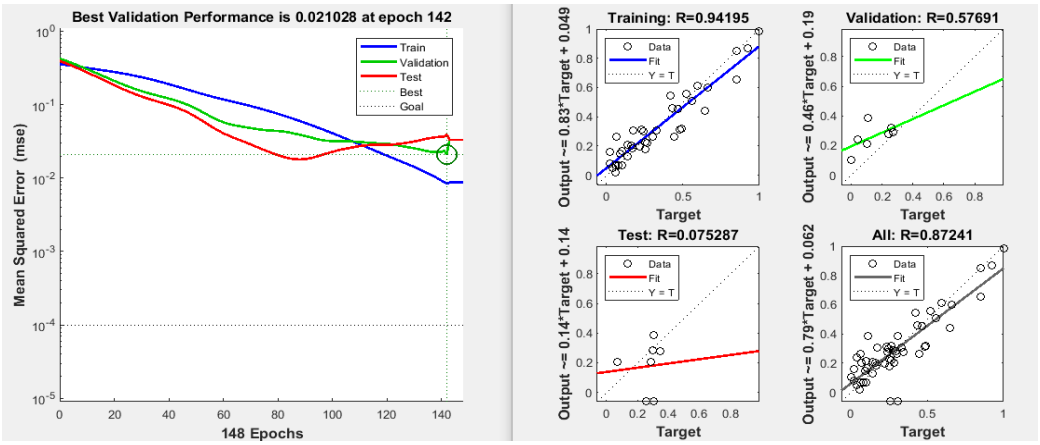


Figure 91: traingdx performance and regression plots for Product 4

- traingdm

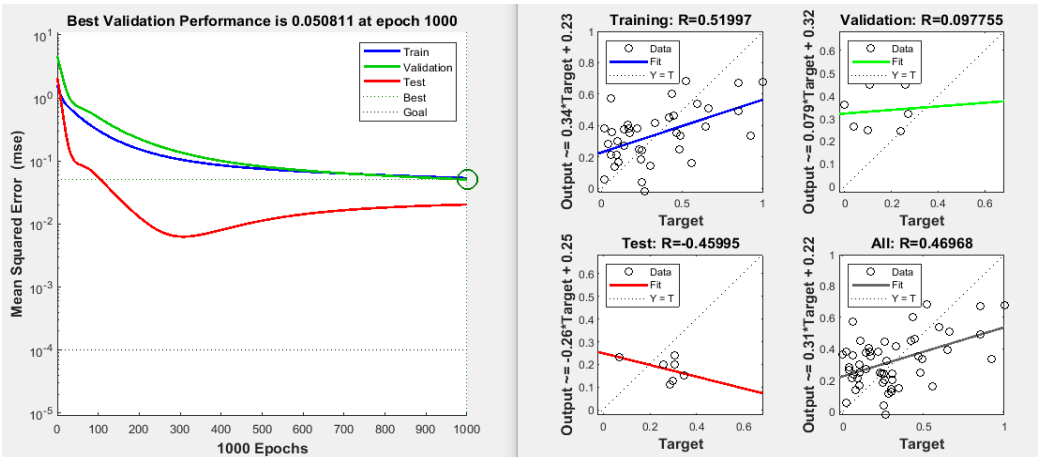


Figure 92: traingdm performance and regression plots for Product 4

- trainbr

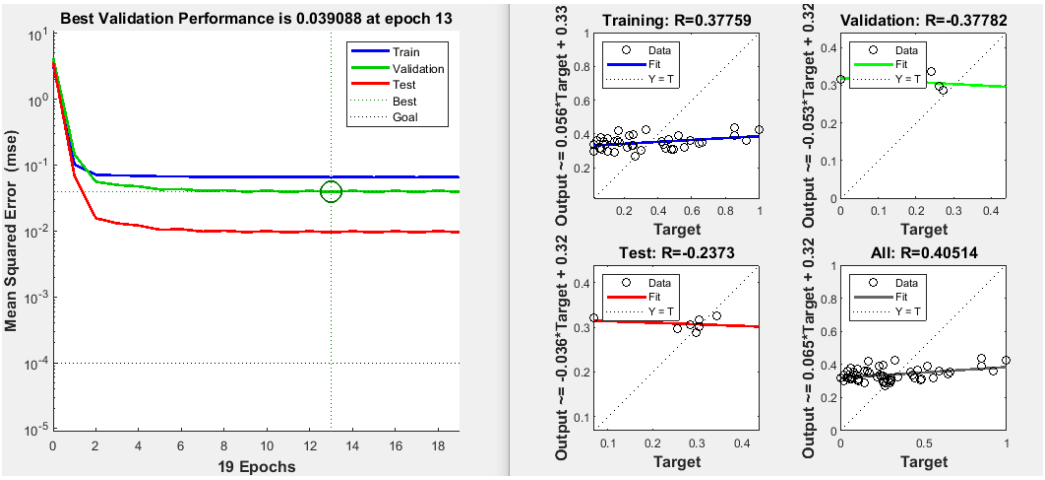


Figure 93: trainbr performance and regression plots for Product 4

- traincgf

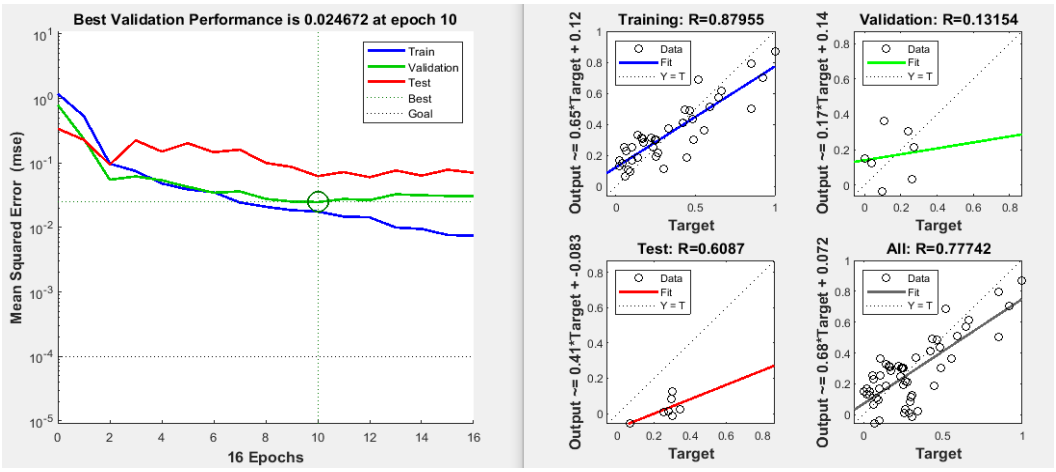


Figure 94: traincgf performance and regression plots for Product 4

- traincgp

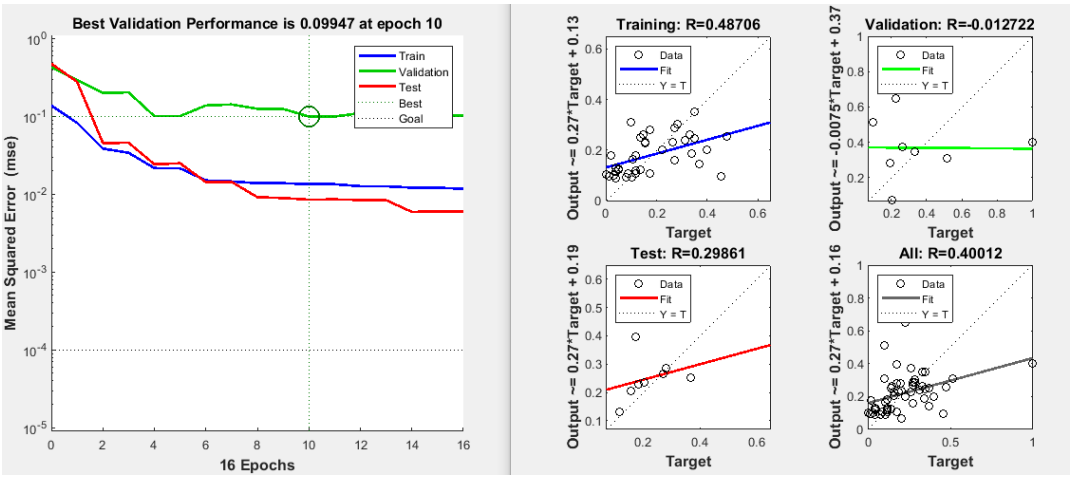


Figure 95: traincgp performance and regression plots for Product 4

- traincgb

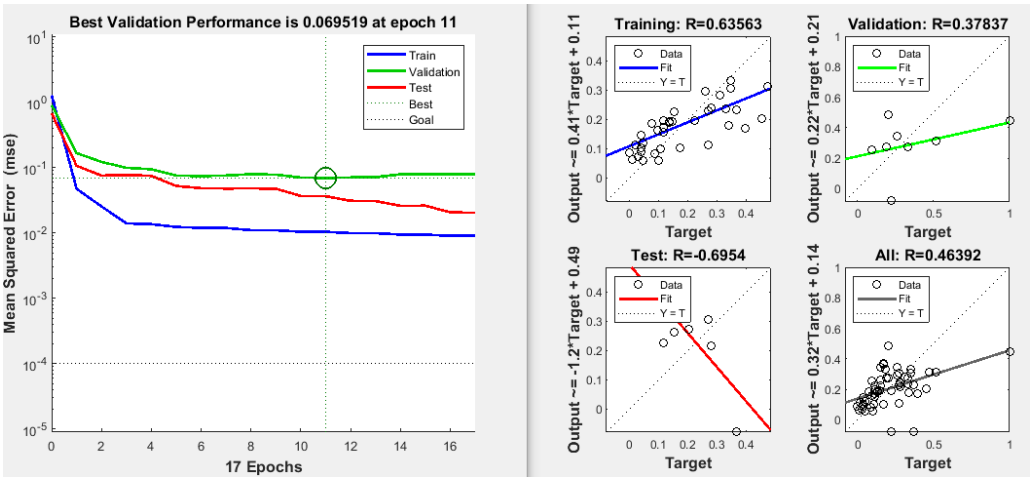


Figure 96: traincgb performance and regression plots for Product 4

- trainbfg

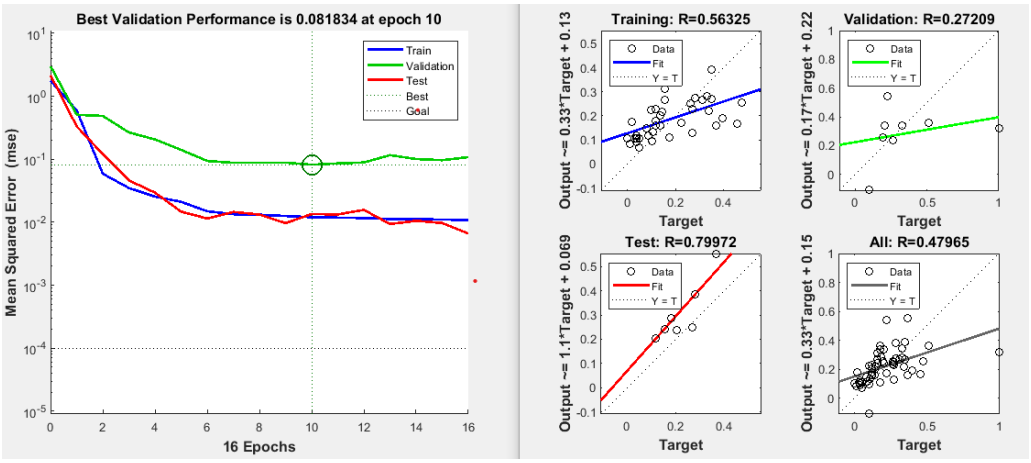


Figure 97: trainbfg performance and regression plots for Product 4

- trainoss

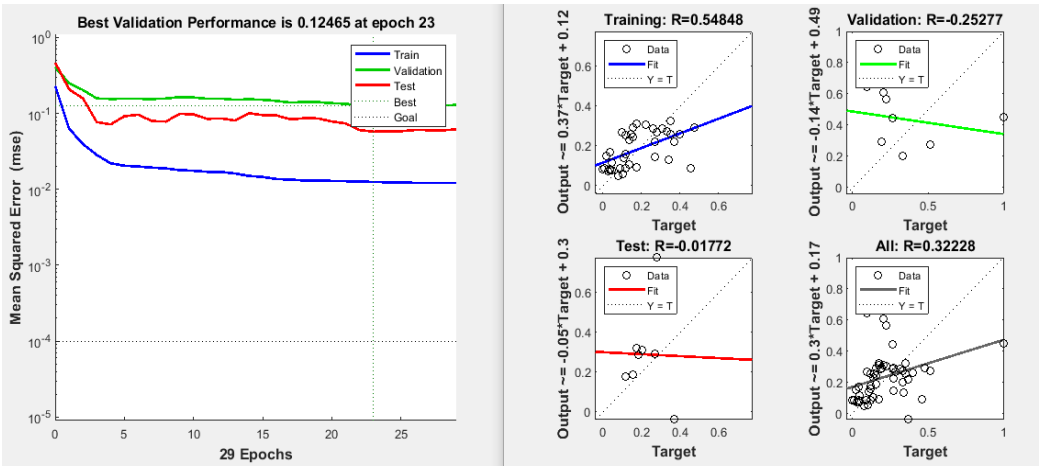


Figure 98: trainoss performance and regression plots for Product 4

- trainlm

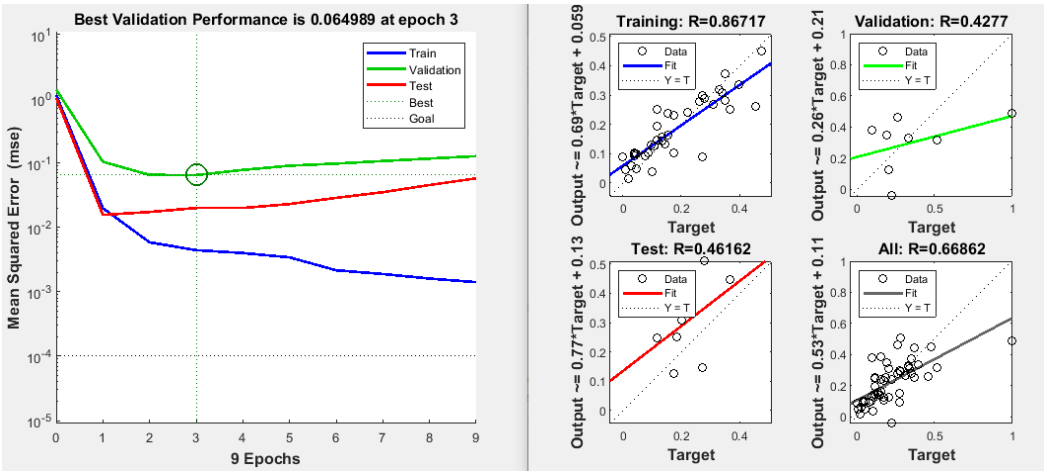


Figure 99: trainlm performance and regression plots for Product 4

Table 23 contains the MAD, RMSE and MSE on training, test and total datasets for Product 4 using the different training algorithms.

Table 23: ANN forecasting results for Product 4

	Train RMSE	**Train MAD**	**Test RMSE**	**Test MAD**	**Total MAD**	**Total MSE**	**Total RMSE**	**Number of neurons used**
traingd	11.5%	17.7%	3.13%	25.1%	18.8%	8.15%	28.5%	18
traingdx	0.4%	11.8%	7.72%	22.2%	13.4%	4.06%	20%	16
traingdm	0.5%	14.2%	0,00%	21.5%	15.3%	4.43%	21.06%	16
trainbr	0,00%	13,00%	6,00%	7.3%	10.3%	2.08%	14,00%	19
traincgf	0.1%	12.4%	5.27%	12.38%	12.4%	2.84%	16.8%	18
traincgp	0,00%	11.6%	0.7%	6,00%	10.8%	2.5%	16,00%	18
traincgb	0.4%	10.4%	0.25%	14.6%	11.08%	2.3%	15.2%	16
trainbfg	0,00%	10.7%	7.3%	10.14%	10.6%	2.26%	15.02%	16
trainoss	1.04%	12.67%	3.86%	17.15%	13.34%	3.58%	18.9%	18
trainlm	0.2%	7.7%	5.5%	12.6%	8.44%	1.57%	12.5%	16

8.4.3 SVR based forecasting for Product 4

- Linear kernel

Table 24 displays linear kernel optimisation for Product 4.

Table 24: Linear optimisation for Product 4

BoxConstraint/C	Epsilon/ε	Validation error		Comment
		RMSE	MAD	
400	0.005	3.22%	20.5%	
400	0.008	1.7%	20.9%	
400	0.01	2.18%	20.8%	
400	0.03	2.3%	20.5%	
400	0.05	2.9%	19.7%	
400	0.08	2.58%	20.8%	
400	0.09	3.4%	21.4%	
400	0.1	4.25%	19.9%	
400	0.2			Underfitting
400	0.3			Underfitting

The smallest error on the validation set was obtained using a BoxConstraint of 400 and an Epsilon of 0.008. Figure 100 shows the plot of the forecast using the linear kernel for Product 4.

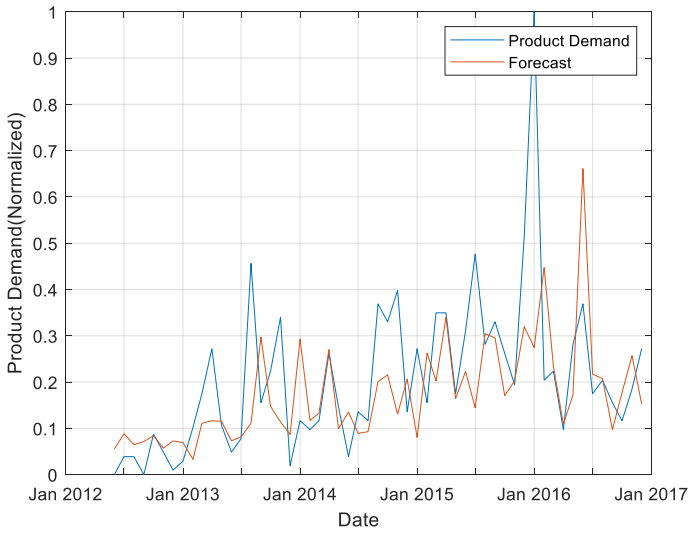


Figure 100: Linear SVR forecast for Product 4

- Gaussian kernel

Table 25 shows the Gaussian kernel optimisation for Product 4.

Table 25: Gaussian kernel optimisation for Product 4

BoxConstraint/C	**Epsilon/ε**	**KernelScale**	**Validation error**		**Comment**
			RMSE	MAD	
500	0.004	0.08			Underfitting on validation set
500	0.004	0,1	13.7%	17.4%	
500	0.004	0.2	9.07%	18.3%	
500	0.004	0.3	6.4%	21.5%	
500	0.004	0.4	6.26%	24.2%	
500	0.004	0.5	7.4%	26.5%	
500	0.004	0.6	9.2%	30.8%	
500	0.004	0.8	15%	52.0%	
500	0.004				No longer captures the training dataset perfectly

It was observed that the smallest error on the validation set was obtained for a BoxConstraint value of 500, an Epsilon of 0.004 and a KernelScale of 0.4. Figure 101 displays the Gaussian SVR forecast plot for Product 4.

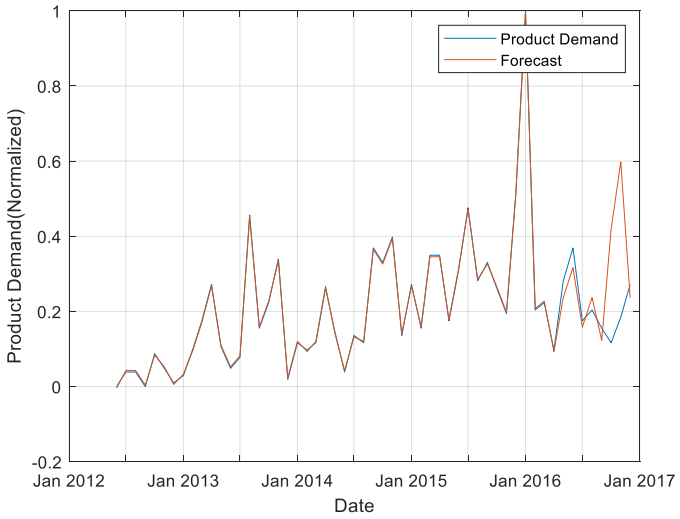


Figure 101: Gaussian SVR for Product 4

- Polynomial kernel

Table 26 shows the polynomial kernel optimisation for Product 4.

Table 26: Polynomial kernel optimisation for Product 4

BoxConstraint/C	Epsilon/ε	KernelScale	Validation error		Comment
			RMSE	MAD	
500	0.023	0.1			Underfitting on validation and test datasets
500	0.023	0.2	94.9%	100.4%	
500	0.023	0.3	82.2%	94%	
500	0.023	0.4	82.6%	94.6%	
500	0.023	0.5	69,00%	84%	
500	0.023	0.6	44.26%	72.15%	
500	0.023	0.7	46.3%	65.6%	
500	0.023	0.9	44.2%	58.9%	
500	0.023	1.5	16.3%	43.2%	
500	0.023	2	10.8%	35.4%	
500	0.023	3			No longer fits the training set perfectly

The smallest error on the validation set was obtained for a BoxConstraint of 500, Epsilon of 0.023, KernelScale of 2 and polynomial order of 2. Figure 102 shows the plot of the forecast using the polynomial kernel for Product 4.

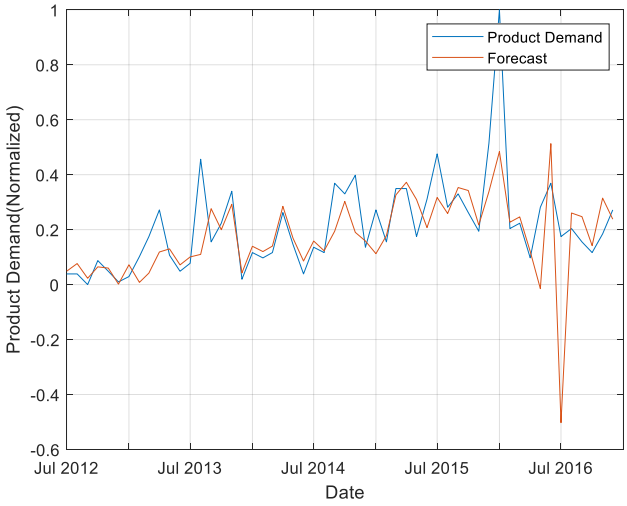


Figure 102: Polynomial SVR forecast for Product 4

Table 27 contains the MAD, MSE and RMSE on training, test and total datasets for the linear, Gaussian and polynomial kernels for Product 4.

Table 27: SVR forecasting results for Product 4

	Train RMSE	**Train MAD**	**Test RMSE**	**Test MAD**	**Total MAD**	**Total MSE**	**Total RMSE**
Linear	4.4%	10.2%	2.3%	9.47%	10.08%	2.48%	15.7%
Gaussian	0%	0.09%	7.08%	11.6%	2.03%	0.5%	7.03%
Polynomial order 2	3.2%	7.2%	6.9%	18.2%	8.86%	2.3%	15.3%

8.4.4 ARIMA forecasting for Product 4

As described in Section 8.4.1, an ARIMA (6,1,6) model was chosen for the ARIMA forecast. Figure 103 displays the plot of the ARIMA forecast.

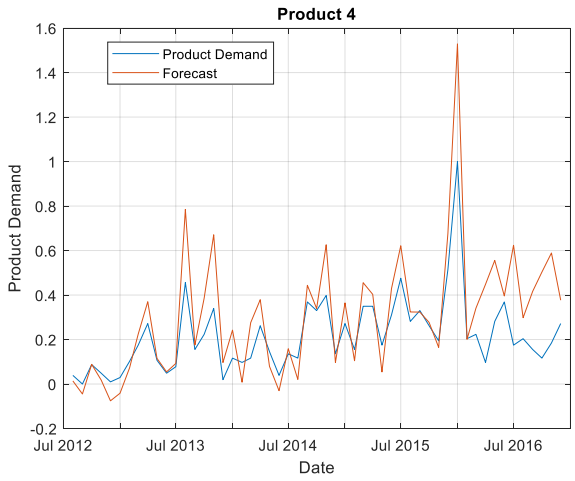


Figure 103: ARIMA forecast for Product 4

Table 28 contains the MAD, MSE and RMSE on training, test and total datasets for the ARIMA method for Product 4.

Table 28: ARIMA results for Product 4

	Train RMSE	Train MAD	Test RMSE	Test MAD	Total MAD	Total MSE	Total RMSE
ARIMA (6,1,6)	35.6%	9.26%	78.42%	26.14%	12.12%	247%	157.17%

8.5 Product 5

Figure 104 shows the time series plot of Product 5.

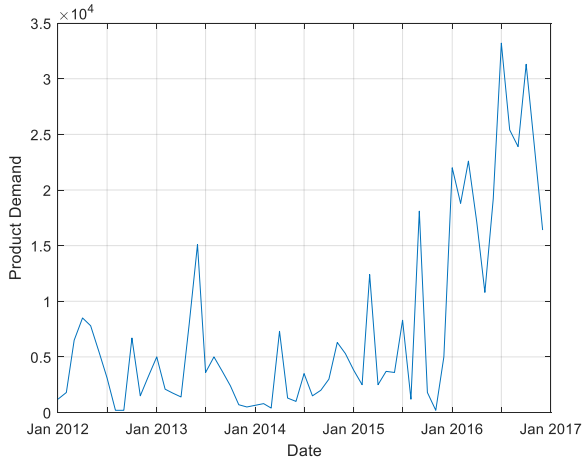


Figure 104: Time series plot of Product 5

8.5.1 Data analysis and pre-processing for Product 5

Figures 105 and 106 display the ACF and PACF of Product 5.

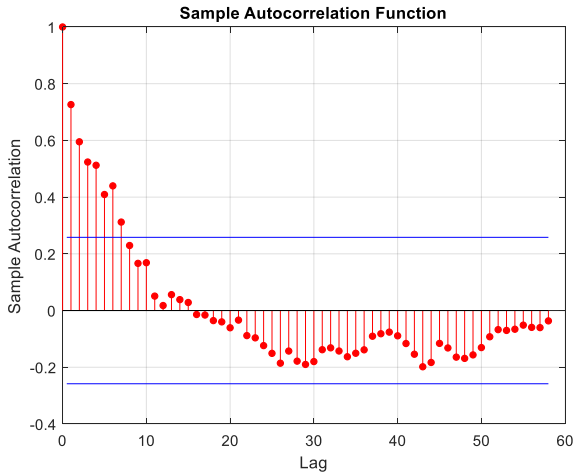


Figure 105: ACF plot of Product 5

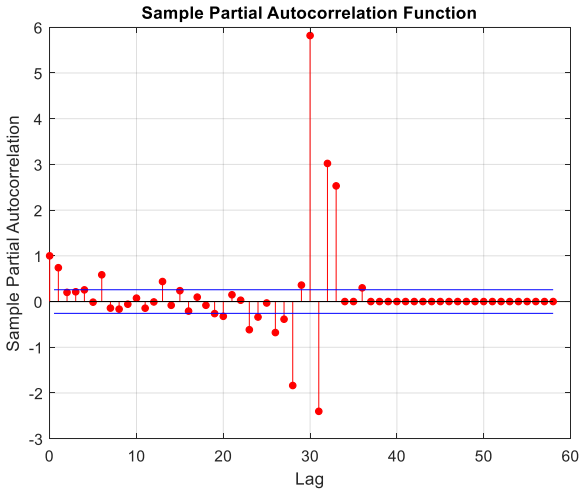


Figure 106: PACF of Product 5

From the ACF plot, very clear trend is visible. This would require at least one degree of differencing for the ARIMA model.

From the PACF plot, the first four lags are significant before the series decays. Thus, the order of the ARIMA AR model was chosen to be four. The number of lags for the ANN and SVR based forecasting models was also chosen as four. From the ACF plot, the first ten lags are significant. So, the order of the ARIMA MA model was ten. The complete ARIMA model was selected to be a (4,1,10) model.

8.5.2 ANN based forecasting for Product 5

Figures 107 – 116 display the performance and regression plots for ANN based forecasting using the different training algorithms.

- traingd

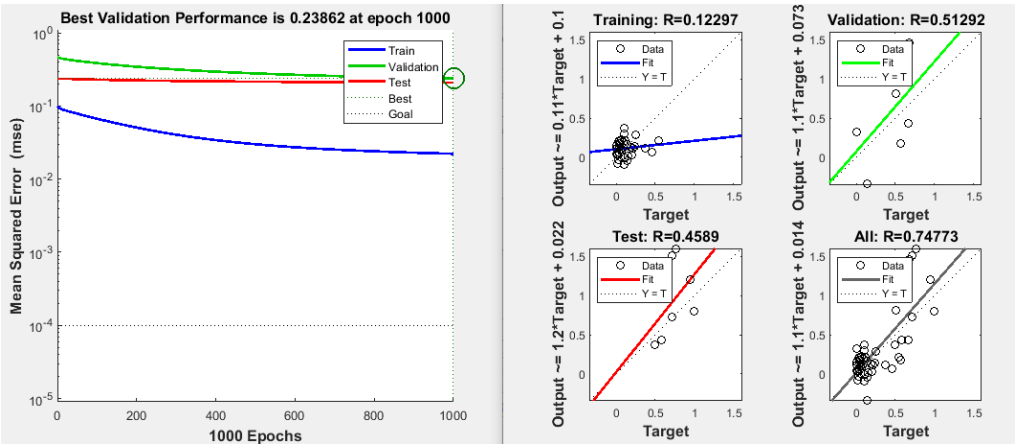


Figure 107: traingd performance and regression plots for Product 5

- traingdx

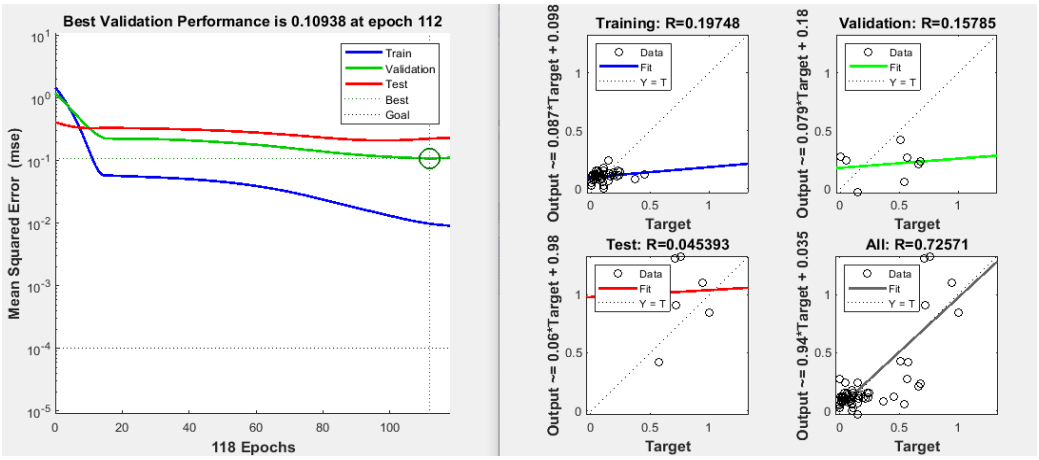


Figure 108: traingdx performance and regression plots for Product 5

- traingdm

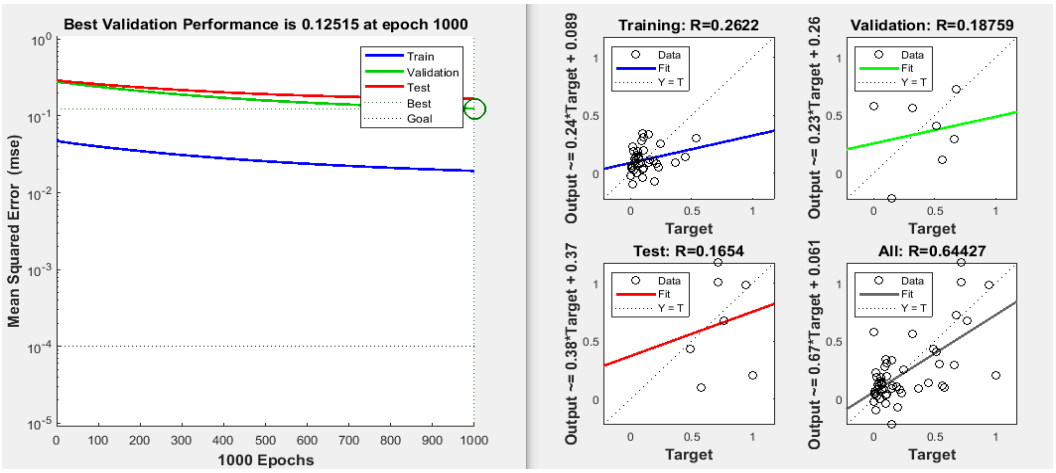


Figure 109: traingdm performance and regression plots for Product 5

- trainbr

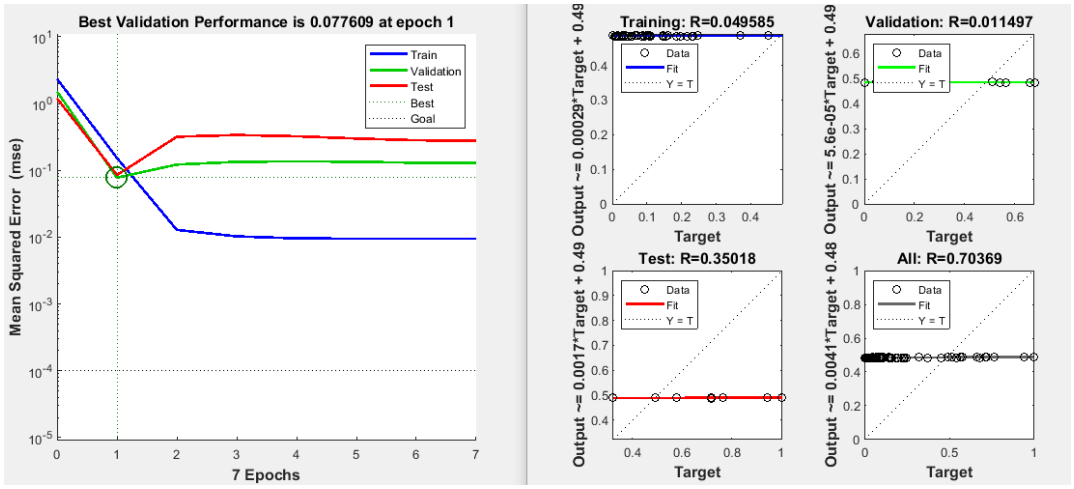


Figure 110: trainbr performance and regression plots for Product 5

- traincgf

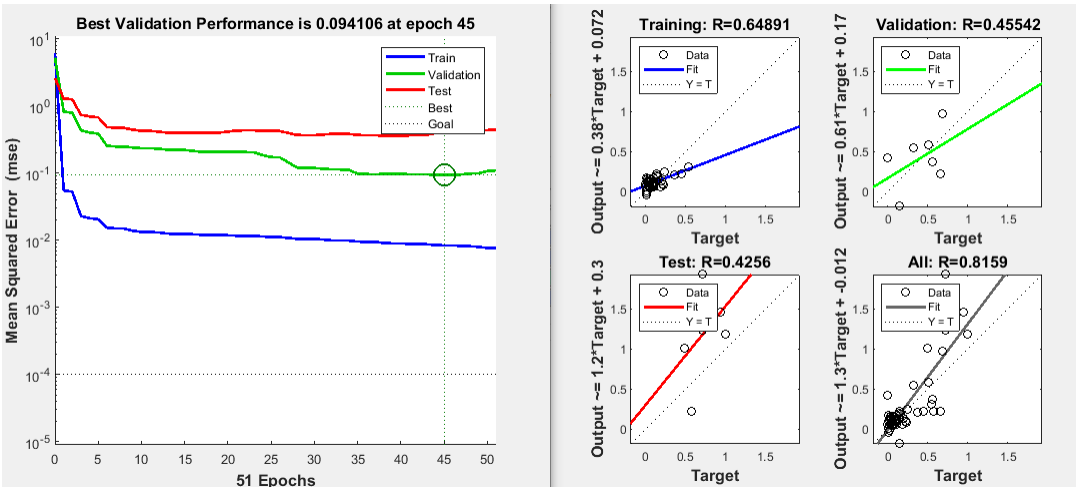


Figure 111: traincgf performance and regression plots for Product 5

- traincgp

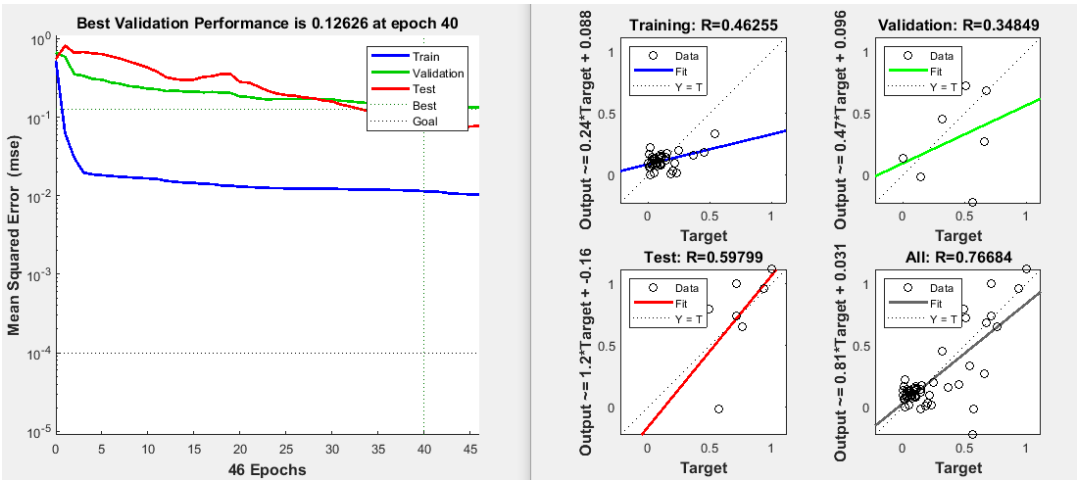


Figure 112: traincgp performance and regression plots for Product 6

- traincgb

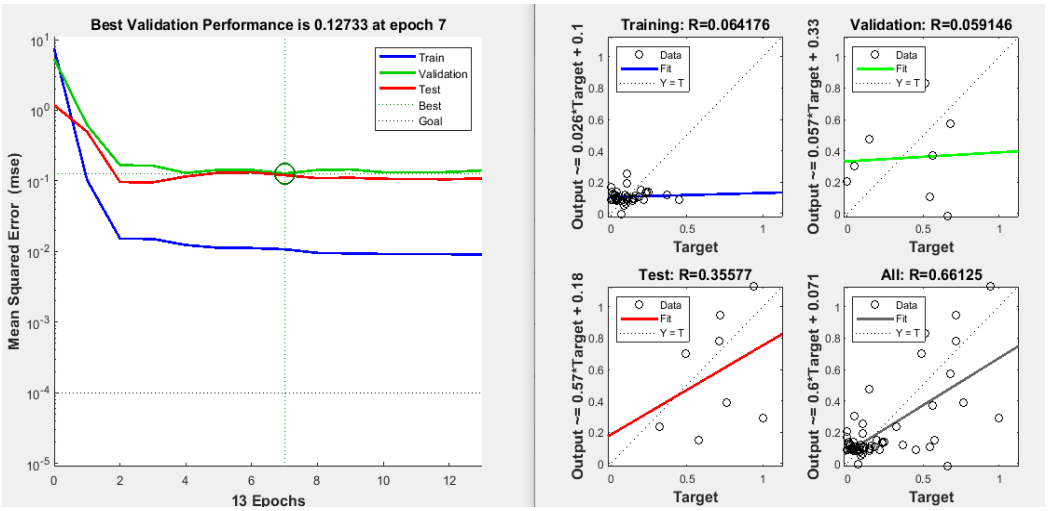


Figure 113: traincgb performance and regression plots for Product 5

- trainbfg

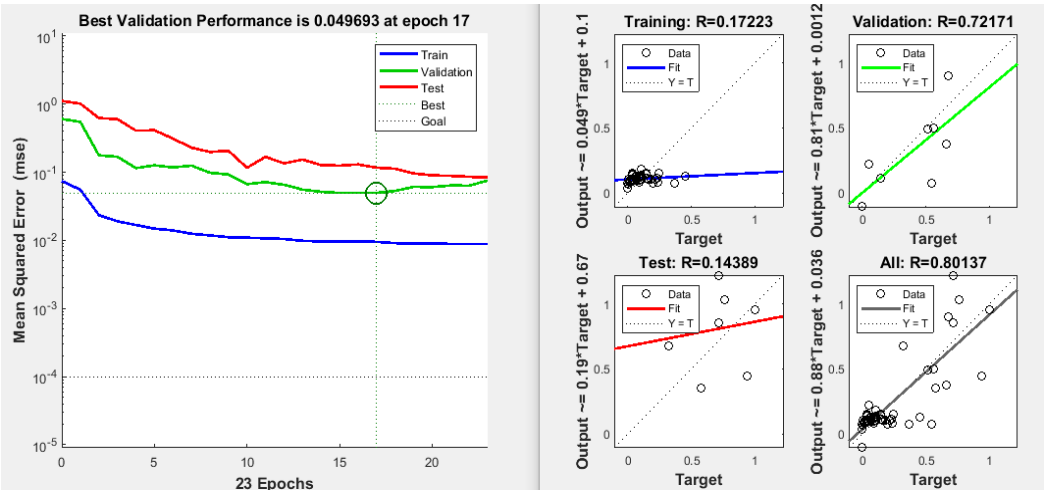


Figure 114: trainbfg performance and regression plots for Product 5

- trainoss

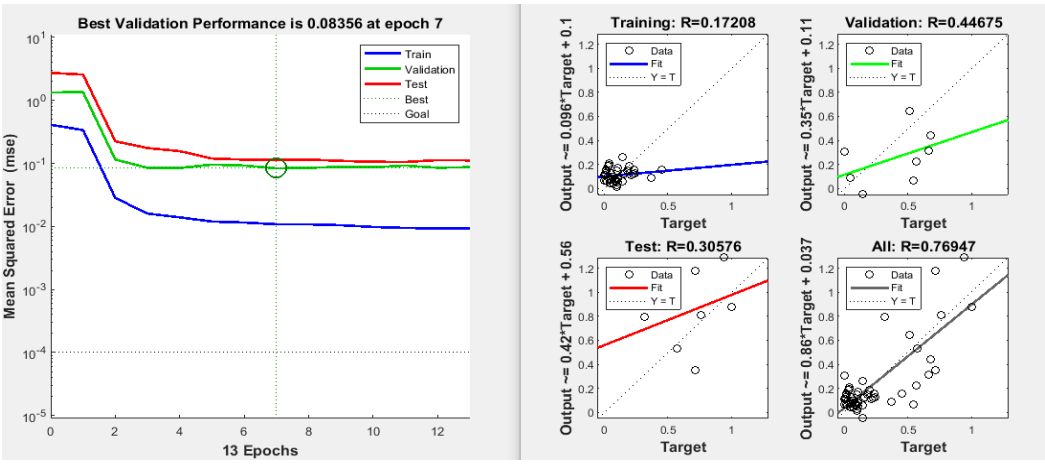


Figure 115: trainoss performance and regression plots for Product 5

- trainlm

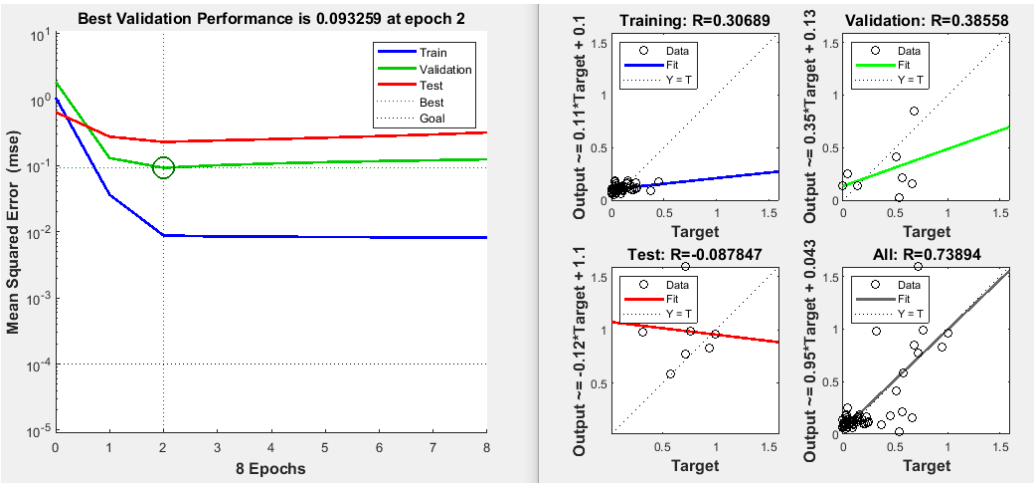


Figure 116: trainlm performance and regression plots for Product 5

Table 29 displays the MAD, RMSE and MSE on the training and test sets for the different training algorithms for Product 5.

Table 29: ANN forecasting results for Product 5

	Train RMSE	**Train MAD**	**Test RMSE**	**Test MAD**	**Total MAD**	**Total MSE**	**Total RMSE**	**Number of neurons used**
traingd	1%	13.94%	52.9%	58.5%	21.2%	13.4%	36.7%	15
traingdx	0.7%	12.4%	21.9%	27.5%	14.8%	4.9%	22.1%	15
traingdm	0%	19%	28.4%	37.9%	21.6%	7.6%	27.5%	15
trainbr	52,00%	35,00%	29.9%	23.9%	33.73%	13.2%	36.4%	17
traincgf	8.43%	12.8%	4.2%	41.3%	16.9%	6.02%	24.5%	20
traincgp	1.4%	12.3%	35.03%	44.7%	17.6%	6.7%	26,00%	20
traincgb	0.2%	11.9%	3.8%	28.8%	14.27%	4.3%	20.7%	21
trainbfg	0.4%	8.9%	8.6%	27.3%	11.9%	3.05%	17.4	20
trainoss	2.04%	11.07%	21.44%	27.2%	13.6%	3.6%	19,00%	19
trainlm	0.8%	9.9%	59.9%	31.4%	13.4%	5.24%	22.9%	18

8.5.3 SVR based forecasting for Product 5

- Linear kernel

Table 30 shows the results of the optimisation using the linear kernel for Product 5.

Table 30: Linear kernel optimisation for Product 6

BoxConstraint/C	Epsilon/ε	Validation error		Comment
		RMSE	MAD	
400	0.005	27.7%	32.5%	
400	0.008	27.4%	32.13%	
400	0.01	27.1%	31.8%	
400	0.02	27.3%	31.5%	
400	0.05	22%	28.9%	
400	0.08	20.4%	27.2%	
400	0.09	19.7%	27.4%	
400	0.1	18.5%	26.7%	
400	0.2	22,00%	33,00%	
400	0.3			Underfitting

It was observed that the smallest validation error was obtained for a BoxConstraint of 400 and an Epsilon of 0.1. Figure 117 displays the plot of the forecast using the linear kernel for Product 5.

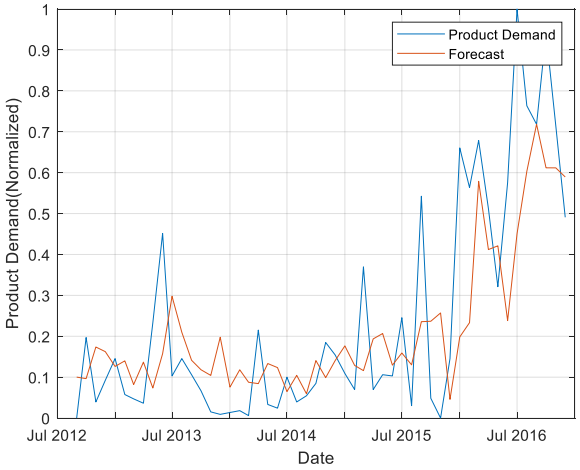


Figure 117: Linear SVR for Product 5

- Gaussian kernel

Table 31 shows the results of the optimisation using the Gaussian kernel for Product 5.

Table 31: Gaussian kernel optimisation for Product 5

BoxConstraint/C	Epsilon/ε	KernelScale	Validation error		Comment
			RMSE	MAD	
500	0.005	0.08			Underfitting
500	0.005	0,1			Underfitting
500	0.005	0.3	37%	39,00%	Underfitting
500	0.005	0.4	45.2%	47.13%	
500	0.005	0.5	52%	55,00%	
500	0.005	0.6	54%	59,00%	
500	0.005	0.7	54.8%	67.7%	
500	0.005	0.8	53.6%	79.9%	
500	0.005	0.9	50.18%	86.7%	
500	0.005	1	43.9%	87.6%	
500	0.005	1,3			No longer captures the training dataset perfectly

It was observed that the smallest validation error was obtained for a BoxConstraint of 500, Epsilon of 0.005 and a KernelScale of 1. Figure 118 below shows the plot of the forecast using the Gaussian kernel for Product 5.

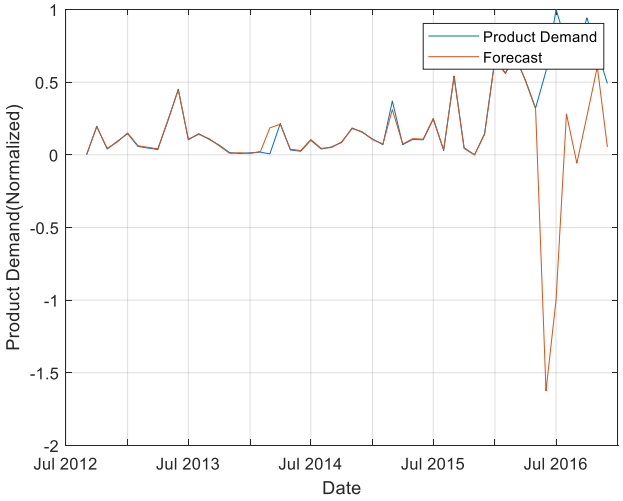


Figure 118: Gaussian SVR for Product 5

- Polynomial kernel

Table 32 shows the results of the optimisation using the polynomial kernel of order 2.

Table 32: Polynomial optimisation for Product 5

BoxConstraint/C	Epsilon/ε	KernelScale	Validation error		Comment
			RMSE	MAD	
500	0.05	0.01			
500	0.05	0.03			Underfitting on test dataset
500	0.05	0.05	20.8%	89.6%	
500	0.05	0.07	27.16%	57.03%	
500	0.05	0.09	10.6%	41.8%	
500	0.05	0.1	24.46%	46.5%	
500	0.05	0.2	29.3%	55%	
500	0.05	0.4	29.9%	116%	
500	0.05	0.6	30.4%	55.4%	
500	0.05				Underfitting

It was observed that the smallest error on the validation set was obtained for a BoxConstraint of 500, Epsilon of 0.05, KernelScale of 0.09 and Polynomial order of 2. Figure 119 displays the plot of the forecast using the polynomial kernel.

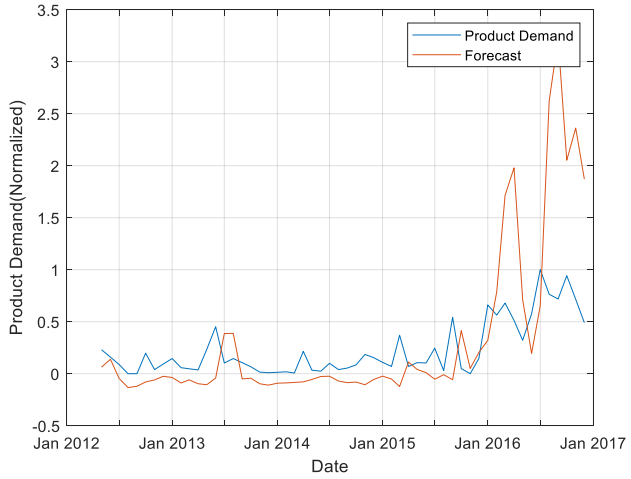


Figure 119: Polynomial forecast results for Product 5

Table 33 displays the MAD, MSE and RMSE on the training, test and total datasets for the linear, Gaussian and polynomial kernels for Product 5.

Table 33: SVR results for Product 5

	Train RMSE	**Train MAD**	**Test RMSE**	**Test MAD**	**Total MAD**	**Total MSE**	**Total RMSE**
Linear	6.8%	15%	19.7%	22.6%	13.2%	3.04%	17.44%
Gaussian	0.3%	0.4%	95.4%	95.4%	13.7%	19%	44%
Polynomial order 2	6.6%	22.3%	102.02%	120%	38.24%	61.8%	36.2%

8.5.4 ARIMA forecasting for Product 5

As described in Section 8.5.1, an ARIMA model of (4,1,10) is chosen from the ACF and PACF plots. Figure 120 displays the plot of the forecast using the ARIMA model.

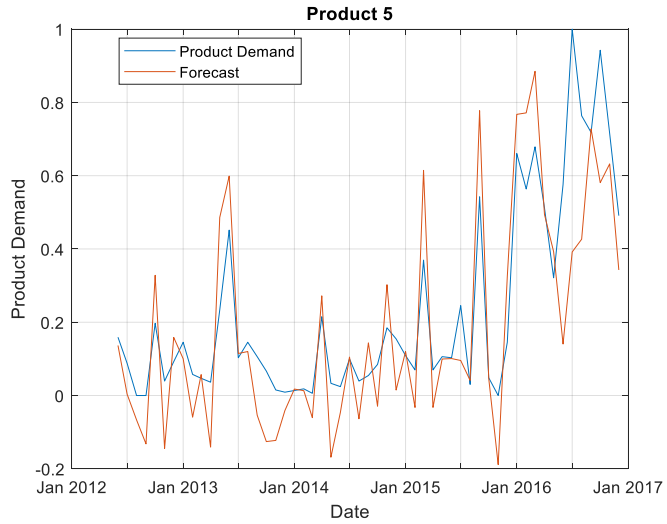


Figure 120: ARIMA forecast for Product 5

Table 34 displays the MAD, RMSE and MSE on the training and test datasets for Product 5 using the ARIMA method.

Table 34: ARIMA forecast results for Product 5

	Train RMSE	**Train MAD**	**Test RMSE**	**Test MAD**	**Total MAD**	**Total MSE**	**Total RMSE**
ARIMA (6,1,10)	14.5%	9.8%	45.1%	22.6%	12.4%	55.1%	74.23%

9. Discussions

It becomes difficult to evaluate results using several error measures as a good performance in one error measure might reflect as a bad performance in another one. The RMSE is especially considered to be a biased error measure. Hence, the MAD is used to evaluate results in this report.

Product 1

Table 35 contains the MAD on training, test and total datasets using all the different ANN algorithms, the SVR algorithms and ARIMA for Product 1. The MAD performance on the total dataset is ranked in the last column.

Table 35: MAD results for Product 1

		Train MAD	**Test MAD**	**Total MAD**	**Ranking in terms of Total MAD**
1	traingd	22%	27%	17%	
2	traingdx	34%	63%	15%	5
3	traingdm	43%	0,25%	20%	
4	trainbr	27.4%	26,50%	15,77%	5
5	traincgf	28.3%	22,17%	13,80%	3
6	traincgp	17.2%	13,40%	14,20%	4
7	traincgb	14.8%	33,80%	12,90%	2
8	trainbfg	21.5%	45,00%	15,50%	6
9	trainoss	34.9%	40,70%	16,80%	9
10	trainlm	36.7%	15,30%	15,98%	7
11	Linear	25%	37%	16.5%	8
12	Gaussian	0.4%	32%	1.55%	1
13	Polynomial order 2	1.5%	194.58%	27.5%	
14	ARIMA (6,0,4)	13.8%	34.9%	17.4%	10

From Table 35, the SVR Gaussian kernel has the best performance in terms of MAD followed by the traincgb algorithm. The ARIMA has a ranking of 10 in terms of MAD.

Product 2

Table 36 contains the MAD on training, test and total datasets using all the different ANN algorithms, the SVR algorithms and ARIMA for Product 2.

Table 36: MAD results for Product 2

		Train MAD	Test MAD	Total MAD	Ranking in terms of Total MAD
1	traingd	41%	48%	11%	
2	traingdx	19%	36%	10%	
3	traingdm	40.1%	107,00%	14%	
4	trainbr	9.38%	8.2%	10.3%	
5	traincgf	24.4%	4,00%	9,04%	6
6	traincgp	41,00%	60,00%	8,47%	
7	traincgb	28.1%	7,00%	7,55%	3
8	trainbfg	36,00%	15,00%	9,09%	7
9	trainoss	22.4%	41,00%	7,64%	4
10	trainlm	26,00%	32,00%	8,03%	5
11	Linear	42%	41%	53%	
12	Gaussian	0.9%	13.07%	2.27%	1
13	Polynomial order 3	2.05%	22.2%	5.04%	2
14	ARIMA (6,0,2)	13.8%	9.5%	10.04%	8

From Table 36, the SVR Gaussian kernel outperforms all the other ML and ARIMA techniques. This is followed by the SVR polynomial kernel and the traincgb algorithm. The ARIMA has a ranking of 8 amongst all the other methods.

Product 3

Table 37 contains the training, test and total MAD using all the different ANN algorithms, the SVR algorithms and ARIMA for Product 3.

Table 37: MAD results for Product 3

		Train MAD	Test MAD	Total MAD	Ranking in terms of total MAD
1	traingd	17.7%	16.7%	14.4%	
2	traingdx	21%	31%	12%	10
3	traingdm	45%	57,00%	15%	
4	trainbr	16.2%	44,00%	11%	7
5	traincgf	11,00%	37,00%	9,30%	3
6	traincgp	12.9%	49,00%	10,00%	4
7	traincgb	22.1%	38,00%	11,4%	8
8	trainbfg	25,00%	37,00%	10,40%	6
9	trainoss	15,00%	36.3%	11,60%	9
10	trainlm	36,00%	64,00%	6,80%	2

11	Linear	9.4%	13.7%	12%	11
12	Gaussian	0.8%	23.7%	3.6%	1
13	Polynomial	4.6%	41.2%	10.2%	5
14	ARIMA (3,1,4)	13.86%	12.1%	13.48%	12

For Product 3, the SVR Gaussian kernel performs the best on the overall dataset followed by the trainlm and the traincgf algorithms. The ARIMA has a ranking of 12 amongst all the other methods.

Product 4

Table 38 contains the training, test and total MAD using all the different ANN algorithms, the SVR algorithms and ARIMA for Product 4.

Table 38: MAD results for Product 4

		Train MAD	**Test MAD**	**Total MAD**	**Ranking in terms of Total MAD**
1	traingd	17.7%	25.1%	18.8%	
2	traingdx	11.8%	22.2%	13.4%	
3	traingdm	14.2%	21.5%	15.3%	
4	trainbr	13,00%	7.3%	10.3%	5
5	traincgf	12.4%	12.38%	12.4%	
6	traincgp	11.6%	6,00%	10.8%	7
7	traincgb	10.4%	14.6%	11.08%	8
8	trainbfg	10.7%	10.14%	10.6%	6
9	trainoss	12.67%	17.15%	13.34%	
10	trainlm	7.7%	12.6%	8.44%	2
11	Linear	10.2%	9.47%	10.08%	4
12	Gaussian	0.09%	11.6%	2.03%	1
13	Polynomial order 2	7.2%	18.2%	8.86%	3
14	ARIMA (6,1,6)	9.26%	26.14%	12.12%	9

For Product 4, the SVR Gaussian kernel performs the best on the whole dataset followed by the trainlm and the SVR polynomial kernels. The ARIMA achieves a ranking of 9 amongst all the other algorithms.

Product 5

Table 39 contains the training, test and total MAD using all the different ANN algorithms, the SVR algorithms and ARIMA for Product 5.

Table 39: MAD results for Product 5

		Train MAD	Test MAD	Total MAD	Ranking in terms of Total MAD
1	traingd	13.94%	58.5%	21.2%	
2	traingdx	12.4%	27.5%	14.8%	8
3	traingdm	19%	37.9%	21.6%	
4	trainbr	35,00%	23.9%	33.73%	
5	traincgf	12.8%	41.3%	16.9%	
6	traincgp	12.3%	44.7%	17.6%	
7	traincgb	11.9%	28.8%	14.27%	7
8	trainbfg	8.9%	27.3%	11.9%	1
9	trainoss	11.07%	27.2%	13.6%	5
10	trainlm	9.9%	31.4%	13.4%	4
11	Linear	15%	22.6%	13.2%	3
12	Gaussian	0.4%	95.4%	13.7%	6
13	Polynomial order 2	22.3%	120%	38.24%	
14	ARIMA (6,1,10)	9.8%	22.6%	12.4%	2

For Product 5, trainbfg performs the best in terms of MAD followed by the ARIMA method and the SVR linear kernel.

From the analysis, it can be concluded that the SVR Gaussian and traincgf algorithms perform the best in terms of MAD for series with no trend and no seasonality (Products 1 and 2). The ARIMA performs eighth and tenth best amongst all the different algorithms

For series with slight trend and no seasonality (Products 3 and 4), the SVR Gaussian and the trainlm algorithms perform the best in terms of MAD. The ARIMA performs twelfth and ninth best amongst all the different algorithms.

For series with clear trend and no seasonality (Product 5), the trainbr algorithm is the best performer followed by the ARIMA.

The remaining 1798 product categories produce the following results:

- For series with no trend and seasonality and only irregular and/or cyclical behaviour, the SVR Gaussian model is the clear performer in 92% of the product series. The remaining 8% have traincgf as the model with the smallest MAD on the overall dataset.
- For series with slight trend and no seasonality, the Gaussian SVR kernel outperforms the other kernels for 85% of the products. The trainlm performs the best for 9% of the products, followed by the SVR polynomial kernel for the remaining 6% .

- For series with clear/heavy trend and no seasonality, the trainbfg algorithm outperforms the other training algorithms for about 50% of the product categories. For the remaining 50%, the ARIMA performs the best.

Out of the five products analysed in Sections 8.1 to 8.5 above, the clear and consistent performer is SVR, especially the Gaussian kernel. The SVR Gaussian kernel outperforms all the other machine learning models as well as the ARIMA model. The performance of the ANN models produces a mixed bag of results, with different algorithms producing good results with different time series. There is no clear and consistent performer.

The following points can be summarised from the analysis

- A good performance on the training dataset does not guarantee a good performance on the test dataset.
- Machine learning models are seen as “black box” models, therefore it becomes difficult to understand how they reach a solution. It is not easy to understand the internal workings of the machine learning models and how they converge to a solution.
- The best performance on the overall dataset in terms of MAD is the SVR Gaussian kernel.
- The ARIMA model tends to over forecast the time series; so, it could lead to overstocking if utilized for demand forecasting applications.

The significance of the findings lies in the applicability of the ANN, SVR and ARIMA methods to datasets of limited size. This research report utilized all three models on datasets of about 60 points, all three were effective in forecasting. However, the effectiveness of the forecast differed per model used. The SVR Gaussian kernel was the most effective out of the machine learning models.

The results also show that all the different ANN, AVR and ARIMA methods can be utilized on highly variable time series data, thereby substantiating the assumption made. The forecasting methods utilized here may be applied to bigger datasets with a larger number of data points to test their efficiency on these datasets.

10. Conclusions and recommendations

Each of the machine learning techniques had the following results:

SVR technique:

For most of the product categories in Kaggle's dataset, the Gaussian SVR kernel produces clear and consistent results in terms of forecasting. Additionally, the Gaussian kernel produces a base set of hyperparameters that can be utilized for all the different time series without having to alter them drastically. It becomes much easier to forecast using the SVR kernel with the base set of hyperparameters as very little tuning of hyperparameters is required with the different time series. It was observed that the SVR Gaussian kernel far outperforms the linear and polynomial kernels in most of the cases. Additionally, it outperforms the ANN forecast results as well as the ARIMA results on the whole dataset. For the polynomial kernel, a better fit on the test dataset could be achieved by a compromise on the fit on the training dataset.

ANN technique:

The performance of the ANN algorithms depends entirely on trial and error. One iteration produces an entirely different set of weights and biases as the weights and biases are initialized to different values with each iteration. The performance of the different algorithms depends entirely on the parameters initialized during that iteration. Therefore, forecasting using ANN produces a mixed bag of results. The ANN algorithm is slower than the SVR algorithm as the updating of weights and biases is required, and the convergence to a solution takes time. Also, it may require several iterations before a suitable solution is reached.

The study concludes that machine learning methods outperform the traditional method of forecasting using ARIMA.

Future research studies may use multi-step forecasting to forecast several steps into the future using the same techniques. For ANN based forecasting, the forecasting capability of RNNs may be investigated and compared to a feed-forward network. The dataset in this study is rather small with regards to the number of time periods being 60 data points. Therefore, only a few datapoints are available to train the machine learning models. A larger dataset could be utilized so that more data points are available for training the machine learning models. There is a likelihood that performance on the test datasets will also improve as the training dataset increases in size.

It is recommended that future research be concentrated on the use of the Gaussian kernel for demand forecasting. The ARIMA may be considered for series with clear trend and no

seasonality. The Holt – Winters method was out of scope for this research report; thus, it was not investigated here. It may be used in future research studies to benchmark the results of the SVR Gaussian kernel.

It is therefore worthwhile for practitioners to implement ML techniques to perform demand forecasting of inventory stock. Specifically, the following recommendations can be utilized by practitioners:

- For series with no trend and seasonality and only irregular/cyclical behaviour, the SVR Gaussian method is the recommended method.
- For series with slight trend and no seasonality, the SVR Gaussian model is still the recommended method.
- For series with heavy trend and no seasonality, the ANN trainbfg model is the recommended method.

11. Bibliography

Adhikari, R. & Agrawal, R., 2013. *An introductory study on time series modeling and forecasting.* 1st ed. s.l.:Lap Lambert Academic Publishing.

Athanasopoulos, G. & Hyndman, R. J., 2018. *Forecasting: principles and practice.* 2nd ed. Melbourne: OTexts.

Beale, M. H., Hagan, M. T. & Demuth, H. B., 2014. *Neural Network Toolbox,* Natick: The Mathworks, Inc..

Ben-Hur, A. & Weston, J., 2010. *A user's guide to support vector machines,* s.l.: s.n.

Benkachcha Said, J. B. H. E. H., 2013. Causal method and time series forecasting model based on artificial neural network. *International Journal of Computer Applications,* 75(7), pp. 0975-8887.

Bourne, C., 2016. *Forecasting with Machine Learning Techniques.* [Online]
Available at: https://www.cardinalpath.com/blog/forecasting-with-machine-learning-techniques/
[Accessed 27 February 2019].

Box, G. E. P., Jenkins, G. M., Reinsel, G. C. & Ljung, G. M., 2015. *Time series analysis: Forecasting and control.* 5th ed. s.l.:John Wiley & Sons, Inc.

Box, G. & Jenkins, G., 1990. *Time series analysis, forecasting and control.* 1st ed. San Francisco: Holden-Day, Inc..

Brownlee, J., 2016. *How to normalize and standardize time series data in Python.* [Online]
Available at: https://machinelearningmastery.com/normalize-standardize-time-series-data-python/
[Accessed 15 May 2020].

Brownlee, J., 2017. *How to create an ARIMA model for time series forecasting in Python.* [Online]
Available at: https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/
[Accessed 20 July 2019].

Buckley, A. & Butler, K. A., 2017. *Dealing with missing data.* [Online]
Available at: https://www.esri.com/about/newsroom/arcuser/dealing-with-missing-data/
[Accessed 9 May 2020].

Carbonneau, R., Vahidov, R. & Laframboise, K., 2007. Machine learning based demand forecasting in supply chains. *International Journal of Intelligent Information Technologies,* 3(4), pp. 40-57.

Chu, C.-W. & Zhang, G. P., 2003. A compatative study of linear and nonlinear models for aggregate retail sales forecasting. *International Journal of Production Economics,* 86(3), pp. 217-231.

Comert, Z. & Kocamaz, A. F., 2017. A study of artificial neural network training algorithms for classification of cardiotocography signals. *Journal of Science and Technology,* 7(2), pp. 93-103.

Cortez, P. & Donate, J. P., 2012. Evolutionary Support Vector Machines for Time Series Forecasting. *LNCS,* Volume II, pp. 523-530.

Crone, S. F., Guajardo, J. & Weber, R., 2006. *A study on the ability of Support Vector regression and neural networks to forecast basic time series patterns.* Boston, Springer.

Davis, M. M. & Heineke, J., 2005. *Operations Management: Integrating manufacturing and services.* 5th ed. New York: McGraw Hill.

Dodge, Y., 2003. *The Oxford dictionary of statistical terms.* 1st ed. New York: Oxford University Press.

Falk, M., 2011. *A first course on time series analysis.* 1st ed. Wurzburg: Chair of Statistics.

Frost, J., 2020. *https://statisticsbyjim.com/basics/outliers/.* [Online]
Available at: https://statisticsbyjim.com/basics/outliers/
[Accessed 2 June 2020].

Haykin, S., 2009. *Neural networks and learning machines.* 3rd ed. New Jersey: Pearson Education.

Hill, T., O'Connor, M. & Remus, W., 1996. Neural network models for time series forecasts. *Management Science,* 42(7), pp. 1082-1092.

Hsu, C.-W., Chang, C.-C. & Lin, C.-J., 2016. *A practical guide to support vector classification,* Taipei: National Taiwan University.

Jain, C. L. & Malehorn, J., 2006. *Benchmarking forecasting practises.* 3rd ed. New York: Graceway Publishing Company, Inc..

Kandananond, K., 2012. A comparison of various forecasting methods for autocorrelated time series. *International Journal of Engineering Business Management,* Volume 4.

Khandelwal, R., 2018. *Finding outliers in dataset using python.* [Online]
Available at: https://medium.com/datadriveninvestor/finding-outliers-in-dataset-using-python-efc3fce6ce32
[Accessed 1 August 2019].

Kotu, V. & Deshpande, B., 2015. *Predictive analytics and data mining.* 1st ed. Waltham: Elsevier.

Kumar, N., 2012. *Using support vector machines effectively.* [Online]
Available at: http://neerajkumar.org/writings/svm
[Accessed 17 October 2018].

Lee, H. L., Padmanabhan, V. & Whang, S., 1997. The bullwhip effect in supply chains. *MIT Sloan Management Review*, 1 April.

Lee, S. H., n.d. *IEMS.* [Online]
Available at: http://www.iems.co.kr/CPL/lecture/part3/3.%20Demand%20Forecasting.pdf
[Accessed 18 May 2020].

Lim, C. & McAleer, M., 2000. Time series forecasts of international travel demand for Australia. *Tourism Management,* Volume 23, pp. 389-396.

Mahanta, J., 2017. *Introduction to Neural Networks, Advantages and Applications.* [Online]
Available at: https://towardsdatascience.com/introduction-to-neural-networks-advantages-and-applications-96851bd1a207
[Accessed 15 August 2019].

Mathworks, 2016. *MathWorks: Machine Learning Challenges: Choosing the best model and avoiding overfitting.* [Online]
Available at: http://www.mathworks.com/tagteam/87371 92974v00 Machine Learning Whitepaper.pdf
[Accessed 3 Jan 2019].

Mathworks, 2019. *fitrsvm.* [Online]
Available at: https://www.mathworks.com/help/stats/fitrsvm.html
[Accessed 4 May 2019].

Mathworks, 2019. *MathWorks : Choose a multilayer neural network training function.* [Online]
Available at: https://www.mathworks.com/help/deeplearning/ug/choose-a-multilayer-neural-network-training-function.html
[Accessed 9 April 2019].

Mathworks, 2019. *MathWorks Documentation: Trainlm.* [Online]
Available at: http://www.mathworks.com/help/deeplearning/ref/trainlm.html
[Accessed 15 March 2019].

Ma, Y., 2012. *A Survey of time series prediction using SVM,* s.l.: s.n.

Mellit, A., Benghanem, M. & Pavan, A. M., 2012. Least squares support vector machine for short-term prediction of meteorological time series. *Theoretical and Applied Climatology*, January.

Mupparaju, K., Soni, A., Gujela, P. & Lanham, M. A., 2018. *A comparative study of machine learning frameworks for demand forecasting,* s.l.: s.n.

Nau, R., 2014. *Duke University.* [Online]
Available at: https://people.duke.edu/~rnau/Slides_on_ARIMA_models--Robert_Nau.pdf
[Accessed 6 November 2019].

Nguyen, T. et al., 2017. Big data analytics in supply chain management: A state-of-the-art literature review. *Computers and Operations Research,* Volume 000, pp. 1-11.

Quesada, A., 2019. *neuraldesigner.* [Online]
Available at: https://www.neuraldesigner.com/blog/5_algorithms_to_train_a_neural_network
[Accessed 1 August 2019].

Raicharoen, T., Lursinsap, C. & Sanguanbhoki, P., 2003. *Application of critical support vector machine to time series prediction.* Bangkok, s.n.

Said, B., Benhra, J. & El Hassani, H., 2013. Causal method and time series forecasting model based on artificial neural network. *International Journal of Computer Applications,* 75(7), pp. 0975-8887.

Samsudin, R., Shabri, A. & Saad, P., 2010. A comparison of time series forecasting using support vector machine and artificial neural network model. *Journal of Applied Sciences,* 10(11), pp. 950-958.

Sewell, M., 2008. *Structural Risk Minimization,* London: s.n.

Shahrabi, J., Mousavi, S. & Heydar, M., 2009. Supply chain demand forecasting : A comparison of machine learning methods and traditional methods. *Journal of applied sciences,* 9(3), pp. 521-527.

Shumway, R. H. & Stoffer, D. S., 2010. *Time series analysis and its applications.* 4th ed. Pittsburgh: Springer.

Singh, A., 2018. *A Gentle Introduction to Handling a Non-Stationary Time Series in Python.* [Online] Available at: https://www.analyticsvidhya.com/blog/2018/09/non-stationary-time-series-python/ [Accessed 24 May 2020].

Smola, A. J. & Scholkopf, B., 2004. A tutorial on support vector regression. *Statistics and Computing,* Issue 14, pp. 199-222.

Smola, A. & Vishwanathan, S., 2008. *Introduction to machine learning.* 1 ed. Cambridge: Cambridge University Press.

Stefanovic, N., 2015. Collaborative business intelligence model for spare parts inventory replenishment. *Computer Science and Information Systems,* III(12), pp. 911-930.

Tran, Q., 2017. *Overfitting and underfitting: Functional programming and intelligent algorithms.* [Online]
Available at: http://kerckhoffs.schaathun.net/FPIA/Slides/09OF.pdf
[Accessed 5 August 2019].

Ullah, M. I., 2014. *Basic statistics and data analysis.* [Online]
Available at: https://itfeature.com/time-series-analysis-and-forecasting/component-of-time-series-data
[Accessed 15 July 2019].

Wei, W. W. S., 2006. *Time series analysis: Univariate and multivariate methods.* 2nd ed. s.l.:Pearson.

Willemain, T. R., Smart, C. N. & Schwarz, H. F., 2004. A new approach to forecasting intermittent demand for service part inventories. *International Journal of Forecasting,* Issue 20, pp. 375-387.

Wolfram, 2020. *Smoothing Data, Filling Missing Data, and Nonparametric Fitting.* [Online]
Available at:
https://reference.wolfram.com/applications/eda/SmoothingDataFillingMissingDataAndNonparametricFitting.html
[Accessed 3 March 2020].

Yue, L. Y., 2007. *Demand forecasting by using support vector machines.* s.l., s.n.

Zaiontz, C., 2020. *Real statistics using Excel.* [Online]
Available at: https://www.real-statistics.com/time-series-analysis/stochastic-processes/handling-

missing-time-series-data/
[Accessed 15 April 2020].

Zhao, F., 2017. *Kaggle.* [Online]
Available at: https://www.kaggle.com/felixzhao/productdemandforecasting
[Accessed 23 April 2018].

Zhou, Q., Han, R. & Li, T., 2015. *A two step dynamic inventory forecasting model for large manufacturing.* s.l., s.n.

Appendix A: ACF and PACF MATLAB plot code

```
tx=datetime (2012,1,01);
ty = dateshift(tx,'start','month',0:59);

length(ty)

[h1,p1] = adftest(T);

display(h1)

plot (ty,T)

grid on
ylabel('Product Demand')
xlabel('Date')

%differencing
T_diff=diff(T)

%stationaraity test
% Hypothesis test : Augmented Dickey-filer

[h2,p2] = adftest(T_diff);

display(h2)


figure
autocorr(T_diff,'NumLags',58)

figure
parcorr(T_diff,'NumLags',58)
```

Appendix B: ANN based forecasting MATLAB code

```
%input the data
%allData= xlsread('Data_organized.xlsx', 'Sheet3','C4:BJ53')
%ProductDemand = mat2cell(allData, ones(size(allData,1),1),
size(allData,2));
cla

T=ProductDemand{3,1};
N=min(T);
M=max(T);

I=mean(T)
J=std(T)

T_normalized=(T-N)/(M-N);
T_standardized=(T-I)./J


V=num2cell(T_normalized)

%define neural network
feedbackDelays=1:3
hiddenLayersize=16
net = narnet(feedbackDelays,hiddenLayersize)

%divideblock to divide the data sequentially into train, validate and test
%datasets
net.divideFcn='divideblock'

%prepare the data for forecasting
[Xs,Xi,Ai,Ts] = preparets(net,{},{},V)

%net.trainFcn = 'traingd';%Batch Gradient Descent
%net.trainFcn = 'traingdx';%Gradient descent with adaptive learning rate
%net.trainFcn = 'traingdm';%Gradient descent with momentum backpropagation
%net.trainFcn = 'trainbr';%Bayesian regularization
%net.trainFcn = 'traincgf';%conjugate gradient backpropagation
%net.trainFcn = 'traincgp';%conjugate gradient backpropagation with
Polak/Ribiere restarts
%net.trainFcn = 'traincgb';%conjugate gradient backpropagation with
Powell/Beale restarts
net.trainFcn = 'trainbfg';%BFGS quasi newton backpropagation
%net.trainFcn = 'trainoss';%One step secant
%net.trainFcn = 'trainlm'; %levenberg marquart

%net.trainParam.epochs  1000
%Maximum number of epochs to train

net.trainParam.goal=0.0001
%Performance goal

net.trainParam.max_fail=6
%Maximum validation failures

%net.trainParam.min_grad    1e-7
%Minimum performance gradient
```

```
%net.trainParam.mu=1
%net.trainParam.mu_dec=0.8
%net.trainParam.mu_inc=1.5

%train the network
net = train(net,Xs,Ts,Xi,Ai);

% Test the Network
[ Ys Xf Af ] = net(Xs,Xi,Ai);

Ts1=cell2mat(Ts)
Ys1=cell2mat(Ys)%

size(Ts1)
size(Ys1)

errors=gsubtract(Ts1, Ys1);%
%MAPE=mean(abs(errors./Ts1))

%Trainerror=gsubtract(Ts1(1:39),Ys1(1:39));
%TrainMAPE=mean(abs(Trainerror./Ts1(1:38)))

%TrainRMSE=sum(Trainerror).^2./length(Ts1(1:39))
%TrainMAD=sum(abs(Trainerror)./length(Ts1(1:39)))

%Validationerror=gsubtract(Ts1(40:48),Ys1(40:48));
%ValidationMAPE=mean(abs(Validationerror./Ts1(39:45)))
%ValidationRMSE=sum(Validationerror).^2./length(Ts1(40:48))
%ValidationMAD=sum(abs(Validationerror)./length(Ts1(40:48)))

Trainset2error=gsubtract(Ts1(1:48),Ys1(1:48))

Trainset2RMSE=sum(Trainset2error).^2./length(Ts1(1:48))
Trainset2MAD=sum(abs(Trainset2error)./length(Ts1(1:48)))

Testerror=gsubtract(Ts1(49:56),Ys1(49:56));
%Testerr=Testerror./Ts1(46:52);
%TestMAPE=mean(abs(Test)))error./Ts1(46:52)))

TestRMSE=sum(Testerror).^2./length(Ts1(49:56))
TestMAD=sum(abs(Testerror)./length(Ts1(49:56)))

%Mean average deviation
MAD=sum(abs(Ts1 - Ys1))./length(Ts1)

%Mean squared error
MSE=sum((Ts1 - Ys1).^2)./(length(Ts1))

RMSE=sqrtm(MSE)

plot(Ts1)
hold on
plot (Ys1)

legend ('Product Demand','Forecast')
ylabel('Product Demand')
xlabel('Date')
```

Appendix C: SVR based forecasting MATLAB code

```
%input the data
cla
%allData= xlsread('Data_organized.xlsx', 'Sheet3','C4:BJ53')
%ProductDemand = mat2cell(allData, ones(size(allData,1),1),
size(allData,2));

T=ProductDemand{4,1};

%normalizing the values from between 0 to 1
N=min(T);
M=max(T);

g=mean(T)

T_normalized=(T-N)/(M-N)

%number of lags
I=8;

%number of sliding windows
K=length(T_normalized)/I

T2=table

%T2(1,:)=T3(1),T3(2),T3(3),T3(4),T3(5)

%table for standardized data
for i=1:57
T2(i,:)={T_normalized(i),T_normalized(i+1),T_normalized(i+2),T_normalized(i
+3)};
end


%divide into training and test datasets
[trainInd,valInd,testInd] = divideblock(57,0.7,0.15,0.15);

trainInd2=[trainInd,valInd];

%support vector regression model
%Mdl =
fitrsvm(T2(trainInd2,:),'Var4','KernelFunction','Gaussian','BoxConstraint',
500,'Epsilon',0.004,'KernelScale',0.1)

%Mdl =
fitrsvm(T2(trainInd2,:),'Var4','KernelFunction','linear','BoxConstraint',40
0,'Epsilon',0.1)

Mdl = fitrsvm(T2(trainInd,:),
'Var4','KernelFunction','polynomial','BoxConstraint',500,'Epsilon',0.016,'K
ernelScale',0.7,'PolynomialOrder',3)

YFit1= predict(Mdl, T2(trainInd,:))
YFit2= predict(Mdl, T2(valInd,:))
YFit3= predict(Mdl, T2(testInd,:))
```

```
YFit4= predict(Mdl, T2(trainInd2,:))



Ts1=T2.Var4(trainInd)
Ts2=T2.Var4(valInd)
Ts3=T2.Var4(testInd)

Ys1=YFit1
Ys2=YFit2
Ys3=YFit3

%Ts=T2.Var9(testInd)
%Ts=T2.Var9(valInd)

%Ys=YFit3

Ts=[Ts1;Ts2;Ts3]

Ys=[Ys1;Ys2;Ys3]

Ts_2=[Ts1;Ts2]

%Ys_2=[YFit4;Ys2]

trainRMSE=sqrtm(sum(Ts1 - Ys1).^2)./(length(Ts1))

trainMAD=sum(abs(Ts1 - Ys1)./length(Ts1))

validationRMSE=sqrtm(sum(Ts2 - Ys2).^2)./(length(Ts2))

validationMAD=sum(abs(Ts2 - Ys2)./length(Ts2))

train2RMSE=sqrtm(sum(Ts_2 - YFit4).^2)./(length(Ts_2))

train2MAD=sum(abs(Ts_2 - YFit4)./length(Ts_2))


testRMSE=sqrtm(sum(Ts3 - Ys3).^2)./(length(Ts3))

testMAD=sum(abs(Ts3 - Ys3)./length(Ts3))


error=(abs(Ts-Ys)./Ts);

MSE=sum((Ts - Ys).^2)./(length(Ts))

RMSE=sqrtm(MSE)

MAD=sum(abs(Ts - Ys))./length(Ts)

tx=datetime (2012,4,01);
ty = dateshift(tx,'start','month',0:56);

%length(ty)
%length(Ts)
```

```
%length(Ys)

%xlim(datetime([2012 2016],[9 12],[1 1]));
%xtickformat('dd-MMM-yyyy')

plot(ty,Ts)
hold on
grid on
plot(ty,Ys)
%plot(ty,[YFit4;Ys3])


legend ('Product Demand','Forecast')
ylabel('Product Demand(Normalized)')
xlabel('Date')
```

Appendix D: ARIMA code

```
T=ProductDemand{1,1};

N=min(T)
M=max(T)

T_norm=(T-N)/(M-N);

%no of lags
n=6

T_model=T_norm'

%tx=datetime (2012,n+1,01);
%ty = dateshift(tx,'start','month',0:60-n-1);

tx=datetime (2012,1,01)
ty=dateshift(tx,'start','month',0:59)

Mdl=arima(n,0,4)

%EstMdl=estimate(Mdl,500)

idp=1:Mdl.P;

ide=(Mdl.P + 1):51;

length(idp)
length(ide)
trest=T_model(ide)

EstMdl=estimate(Mdl,T_model(ide),'Y0',T_model(idp));

length(EstMdl)

residual=infer(EstMdl,T_model(ide))

length(residual)
length(T_model(ide))
prediction=T_model(ide)+residual

figure
plot (T_model(ide))
hold on
plot (prediction)
legend('Original','ARIMA','Location','best');
grid on


yf0=T_model(idxest(end - n:end));
%yf0_0=T_model(idxpre(1:Mdl.P));
%yf1=T_model(idxest((Mdl.P + 1):45));
yf=forecast(EstMdl,9,yf0)
%yf1_1=forecast(EstMdl,45,yf0_0)
```

```
%tx=datetime (2012,n+1,01);
%ty = dateshift(tx,'start','month',0:60-n-1);

length(T)
length(ty)

yf_total=[prediction;yf]
length(yf_total)
length(yf)

figure
h1=plot(ty(n+1:end),T_norm(n+1:end))
length(ty(n+1:end))
length(T_norm(n+1:end))
hold on
h2=plot (ty(n+1:end),yf_total)
%h2=plot(ty(46:54),yf,'r');

legend([h1 h2],"Product Demand","Forecast",'Location',"best")

title("Product 1")
xlabel("Date")
ylabel("Product Demand")
grid on
hold off

%error calculation
%training set

training_MAD=sum(abs(T_model(ide)-prediction))/length(prediction)
training_MSE=(sum(T_model(ide)-prediction).^2)/length(prediction)
training_RMSE=sqrtm(training_MSE)
%test set

test_MAD=sum(abs(T_model(52:end)-yf))/length(yf)
test_MSE=(sum(T_model(52:end)-yf).^2)/length(yf)
test_RMSE=sqrtm(test_MSE)
```