AN APPLICATION OF TIME SERIES ANALYSIS OF WEATHER FORECASTING

A PROJECT REPORT

Submitted by

VENKATESH.M (412516205103)

VISHUVAPANDI.P (412516205107)

Submitted to

FACULTY OF INFORMATION TECHNOLOGY

in the partial fulfilment for the award of the degree of

BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

SRI SAIRAM ENGINEERING COLLEGE: CHENNAI 600 044

ANNA UNIVERSITY: CHENNAI 600 025

SEPTEMBER 2020

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"AN APPLICATION OF TIME SERIES ANALYSIS OF WEATHER FORECASTING"** is the bonafide work of **"VENKATESH.M (412516205103)** and **VISHUVAPANDI.P (412516205107)"** who carried out the project work under my supervision.

SIGNATURE	**SIGNATURE**
Dr. T. Sheela,	Ms. J.Julie
HEAD OF THE DEPARTMENT,	**ASSISTANT PROFESSOR (SUPERVISOR),**
Department of Information Technology, Sri Sairam Engineering College, West Tambaram, Chennai – 600 044.	Department of Information Technology Sri Sairam Engineering College, West Tambaram, Chennai – 600 044.

Submitted for the Anna University Project Examination held on 22 September 2020 at Sri Sairam Engineering College, Chennai – 600 044.

(EXAMINER 1) **(EXAMINER 2)** **(EXAMINER 3)**

ACKNOWLEDGEMENT

We express our gratitude to our respected Founder and Chairman, Late **Thiru. MJF. Ln. LEO MUTHU** who always had the vision that changes can be made only through education which fulfils all the needs. His constant blessings is the driving force which made us to complete the project work successfully..

We express our gratitude to our **CEO Mr. J. SAI PRAKASH LEO MUTHU** for his constant encouragement in completing this project.

We express our solemn thanks to our Principal **Dr. A. RAJENDRA PRASAD** for having given us his spontaneous and whole-hearted encouragement for completing this project.

We are indebted to our Head of the Department **Dr. T. SHEELA** for her constant support during the entire course of this project work.

Our sincere thanks to our supervisor **Ms. J. Julie**, Asst. Professor for her valuable suggestions and constant encouragement in successful completion of this project.

Our sincere thanks to our Project Coordinator **Ms. V. NARMADHA,** Assistant Professor for their kind support in bringing out this project successfully.

Finally, we would like to thank all the staff members of the **Department of Information Technology** and all others who contributed directly and indirectly for the successful completion of our project.

We thank our almighty for his blessings to make this project a success.

ABSTRACT

Weather forecasting has become an important field of research in the last few decades. In most of the cases the researcher had attempted to establish a linear relationship between the input weather data and the corresponding target data. But with the discovery of nonlinearity in the nature of weather data, the focus has shifted towards the nonlinear prediction of the weather data. The advent of new satellite imaging technologies has made satellite images more accessible. These images can be utilized for weather predictions. This work proposes a simple approach for weather prediction that relies on satellite images and weather data as inputs. The method is divided into two parts. The first part involves the use of image processing techniques such as image segmentation on the satellite images to extract the cloud cover. On basis of the cloud cover obtained, percentage cloud cover is calculated and this calculated percentage value is stored, which is later used in the second stage of the approach. The second part involves the use of the cloud cover percentage along with other inputs such as temperature, humidity and wind speed to train an artificial neural network. The weather prediction is done by artificial neural networks. Most of the current cloud extraction algorithms are quite complicated to implement and execution time is potentially slow. In this paper, we present a novel approach which is simple to implement, fast in execution and provides good results in tests.

Keywords-Satellite Images, Image Processing, Artificial Neural Networks

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 AIM & OBJECTIVES

- To Create a low cost weather prediction strategy
- Providing much more accuracy in Weather Prediction than that of previous techniques
- Implementing Image Proceesing Strategies in Weather Prediction Techniques

1.2 PROBLEM STATEMENT

One of the most interesting features of Earth, as seen from ground, is the ever-changing distribution of clouds. They are as natural as anything we encounter in our daily lives. As they float above us, we hardly give their presence a second thought. And yet, clouds have an enormous influence on Earth's energy balance, climate, and weather. Weather forecasting is the application of science and technology to predict the state of the atmosphere for a given location. Weather forecasts are made by collecting quantitative data about the current state of the atmosphere and using scientific understanding of atmospheric processes to project how the atmosphere will evolve. The chaotic nature of the atmosphere, the massive computational power required to solve the equations that describe the atmosphere, error involved in measuring the initial conditions, and an incomplete understanding of atmospheric processes mean that forecasts become less accurate as the difference in current time and the time for which the forecast is being made increases. Depending upon their height and characteristics, the clouds can be classified into various types. Clouds can be characterized based upon their shape, color, density, degree of cover, altitude at which they occur.

There are three basic types of clouds and seven other types of clouds. The basic clouds are the Stratus, Cirrus, and Cumulus. Clouds can also be classified based on their altitude i.e., High Clouds are the “Cirrus”, the Middle Clouds are the “Alto”, and the Low Clouds are the “Stratus”. It has been found that the rainfall clouds are the Nimbostratus and Cumulonimbus. Other clouds like Cumulus will produce rain at rare chances. Clouds can block the sunlight rays reaching the Earth’s surface due to which the Earth’s surface tends to be cooler. Clouds have different shapes and structure due to which prediction of weather become complex.

1.3 PURPOSE

The main objective of the thesis is to recognize the type of cloud and estimate rainfall using certain features from the digital cloud images. Use of wavelet to separate the points needed for the cluster and using k-means clustering to combine those points will provide a better performance than previous techniques used.

1.4 SCOPE

The Scope of the work is predicting rainfall using the digital images rather than using the satellite images. The Satellite images costs lot so everyone can’t get them easily. When some new techniques are used we could get more accurate prediction

CHAPTER 2

LITERATURE SURVEY

2.1 Time Series Forecasting Using Stochastic Models

2.1.1 Introduction

In the previous chapter we have discussed about the fundamentals of time series modeling and forecasting. The selection of a proper model is extremely important as it reflects the underlying structure of the series and this fitted model in turn is used for future forecasting. A time series model is said to be linear or non-linear depending on whether the current value of the series is a linear or non-linear function of past observations.

In general models for time series data can have many forms and represent different stochastic processes. There are two widely used linear time series models in literature, viz. Autoregressive (AR) [6, 12, 23] and Moving Average (MA) [6, 23] models. Combining these two, the Autoregressive Moving Average (ARMA) [6, 12, 21, 23] and Autoregressive Integrated Moving Average (ARIMA) [6, 21, 23] models have been proposed in literature. The Autoregressive Fractionally Integrated Moving Average (ARFIMA) [9, 17] model generalizes ARMA and ARIMA models. For seasonal time series forecasting, a variation of ARIMA, viz. the Seasonal Autoregressive Integrated Moving Average (SARIMA) [3, 6, 23] model is used. ARIMA model and its different variations are based on the famous Box-Jenkins principle [6, 8, 12, 23] and so these are also broadly known as the Box-Jenkins models.

Linear models have drawn much attention due to their relative simplicity in understanding and implementation. However many practical time series show non-linear patterns. For example, as mentioned by R. Parrelli in [28], non-linear models are appropriate for predicting volatility changes in economic and financial time series. Considering these facts, various non-linear models have been suggested in literature. Some of them are the famous Autoregressive Conditional Heteroskedasticity (ARCH) [9, 28] model and its variations like Generalized ARCH (GARCH) [9, 28], Exponential Generalized ARCH (EGARCH) [9] etc., the Threshold Autoregressive (TAR) [8, 10] model, the Non-linear Autoregressive (NAR) [7] model, the Non-linear Moving Average (NMA) [28] model, etc.

In this chapter we shall discuss about the important linear and non-linear stochastic time series models with their different properties. This chapter will provide a background for the upcoming chapters, in which we shall study other models used for time series forecasting.

2.1.2 The Autoregressive Moving Average (ARMA) Models

An ARMA(p, q) model is a combination of AR(p) and MA(q) models and is suitable for univariate time series modeling. In an AR(p) model the future value of a variable is assumed to be a linear combination of p past observations and a random error together with a constant term.

Here y_t and ε_t are respectively the actual value and random error (or random shock) at time period t , ϕ_i (i = 1, 2,..., p) are model parameters and c is a constant. The integer constant p is known as the order of the model. Sometimes the constant term is omitted for simplicity. Usually For estimating parameters of an AR process using the given time series, the Yule-Walker equations [23] are used.

Just as an AR(p) model regress against past values of the series, an MA(q) model uses past errors as the explanatory variables.

Here μ is the mean of the series, θ_j (j = 1, 2,..., q) are the model parameters and q is the order of the model. The random shocks are assumed to be a white noise [21, 23] process, i.e. a sequence of independent and identically distributed (i.i.d) random variables with zero mean and a constant variance σ^2 . Generally, the random shocks are assumed to follow the typical normal distribution. Thus conceptually a moving average model is a linear regression of the current observation of the time series against the random shocks of one or more prior observations. Fitting an MA model to a time series is more complicated than fitting an AR model because in the former one the random error terms are not fore-seeable.

Autoregressive (AR) and moving average (MA) models can be effectively combined together to form a general and useful class of time series models, known as the ARMA models.

Usually ARMA models are manipulated using the lag operator [21, 23] notation. The lag or backshift operator is defined as $Ly_t = y_{t-1}$.

It is shown in [23] that an important property of AR(p) process is invertibility, i.e. an AR(p) process can always be written in terms of an MA(∞) process. Whereas for an MA(q) process to be invertible, all the roots of the equation θ (L) = 0 must lie outside the unit circle. This condition is known as the Invertibility Condition for an MA process.

2.1.3 Stationarity Analysis

When an AR(*p*) process is represented as $\varepsilon_t = \phi(L) y_t$, then $\phi(L) = 0$ is known as the characteristic equation for the process. It is proved by Box and Jenkins [6] that a necessary and sufficient condition for the AR(*p*) process to be stationary is that all the roots of the characteristic equation must fall outside the unit circle. Hipel and McLeod [23] mentioned another simple algorithm (by Schur and Pagano) for determining stationarity of an AR process. An MA(*q*) process is always stationary, irrespective of the values the MA parameters [23]. The conditions regarding stationarity and invertibility of AR and MA processes also hold for an ARMA process. An ARMA(*p, q*) process is stationary if all the roots of the characteristic equation $\phi(L) = 0$ lie outside the unit circle. Similarly, if all the roots of the lag equation (*L*) = 0 lie outside the unit circle, then the ARMA(*p, q*) process is invertible and can be expressed as a pure AR process.

2.1.4 Autocorrelation and Partial Autocorrelation Functions (ACF and PACF)

To determine a proper model for a given time series data, it is necessary to carry out the ACF and PACF analysis. These statistical measures reflect how the observations in a time series are related to each other. For modeling and forecasting purpose it is often useful to plot the ACF and PACF against consecutive time lags. Here μ is the mean of the time series, i.e. $\mu = E[x_t]$. The autocovariance at lag zero i.e. γ_0 is the variance of the time series. From the definition it is clear that the autocorrelation coefficient ρ_k is dimensionless and so is independent of the scale of measurement. Also, clearly $-1 \le \rho_k \le 1$. Statisticians Box and Jenkins [6] termed γ_k as the theoretical Autocovariance Function (ACVF) and ρ_k as the theoretical Autocorrelation Function (ACF).

Another measure, known as the Partial Autucorrelation Function (PACF) is used to measure the correlation between an observation k period ago and the current observation, after controlling for observations at intermediate lags (i.e. at lags < k) [12]. At lag 1, PACF(1) is same as ACF(1). The detailed formulae for calculating PACF are given in [6, 23].

Normally, the stochastic process governing a time series is unknown and so it is not possible to determine the actual or theoretical ACF and PACF values. Rather these values are to be estimated from the training data, i.e. the known time series at hand.

As explained by Box and Jenkins [6], the sample ACF plot is useful in determining the type of model to fit to a time series of length N. Since ACF is symmetrical about lag zero, it is only required to plot the sample ACF for positive lags, from lag one onwards to a maximum lag of about N/4. The sample PACF plot helps in identifying the maximum order of an AR process. The methods for calculating ACF and PACF for ARMA models are described in [23]. We shall demonstrate the use of these plots for our practical datasets in Chapter 7.

2.1.5 Autoregressive Integrated Moving Average (ARIMA) Models

The ARMA models, described above can only be used for stationary time series data. However in practice many time series such as those related to socio-economic [23] and business show non-stationary behavior. Time series, which contain trend and seasonal patterns, are also non-stationary in nature [3, 11]. Thus from application view point ARMA models are inadequate to properly describe non-stationary time series, which are frequently encountered in practice. For this reason the ARIMA model [6, 23, 27] is proposed, which is a generalization of an ARMA model to include the case of non-stationarity as well.

In ARIMA models a non-stationary time series is made stationary by applying finite differencing of the data points

- Here, p, d and q are integers greater than or equal to zero and refer to the order of the autoregressive, integrated, and moving average parts of the model respectively.

- The integer d controls the level of differencing. Generally d=1 is enough in most cases. When d=0, then it reduces to an ARMA(p,q) model.

- An ARIMA(p,0,0) is nothing but the AR(p) model and ARIMA(0,0,q) is the MA(q) model.
- ARIMA(0,1,0), i.e. $y_t = y_{t-1} + \varepsilon_t$ is a special one and known as the Random Walk model [8, 12, 21]. It is widely used for non-stationary data, like economic and stock price series.

A useful generalization of ARIMA models is the Autoregressive Fractionally Integrated Moving Average (ARFIMA) model, which allows non-integer values of the differencing parameter d. ARFIMA has useful application in modeling time series with long memory [17]. In this model the expansion of the term $(1 - L)^d$ is to be done by using the general binomial theorem. Various contributions have been made by researchers towards the estimation of the general ARFIMA parameters.

2.1.6 Seasonal Autoregressive Integrated Moving Average (SARIMA) Models

The ARIMA model (3.8) is for non-seasonal non-stationary data. Box and Jenkins [6] have generalized this model to deal with seasonality. Their proposed model is known as the Seasonal ARIMA (SARIMA) model.

In this model seasonal differencing of appropriate order is used to remove non-stationarity from the series. A first order seasonal difference is the difference between an observation and the corresponding observation from the previous year and is calculated as $z_t = y_t - y_{t-s}$. For monthly time series s = 12 and for quarterly time series s = 4. This model is generally termed as the SARIMA(p, d, q) × $(P, D,Q)^s$ model.

2.1.7 Some Nonlinear Time Series Models

So far we have discussed about linear time series models. As mentioned earlier, nonlinear models should also be considered for better time series analysis and forecasting. Campbell, Lo and McKinley (1997) made important contributions towards this direction. According to them almost all non-linear time series can be divided into two branches: one includes models non-linear in mean and other includes models non-linear in variance (heteroskedastic).

2.1.8 Box-Jenkins Methodology

After describing various time series models, the next issue to our concern is how to select an appropriate model that can produce accurate forecast based on a description of historical pattern in the data and how to determine the optimal model orders. Statisticians George Box and Gwilym Jenkins [6] developed a practical approach to build ARIMA model, which best fit to a given time series and also satisfy the parsimony principle. Their concept has fundamental importance on the area of time series analysis and forecasting [8, 27].

The Box-Jenkins methodology does not assume any particular pattern in the historical data of the series to be forecasted. Rather, it uses a three step iterative approach of model identification, parameter estimation and diagnostic checking to determine the best parsimonious model from a general class of ARIMA models [6, 8, 12, 27]. This three-step process is repeated several times until a satisfactory model is finally selected. Then this model can be used for forecasting future values of the time series.

The Box-Jenkins forecast method is schematically shown in Fig. 2.1:

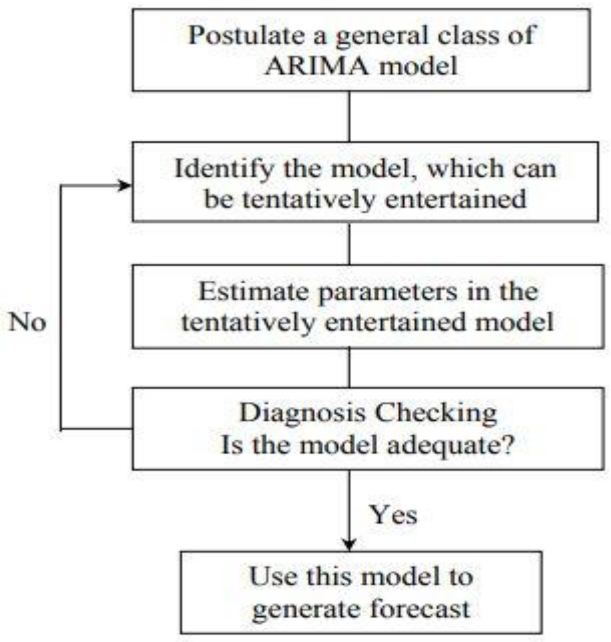

Postulate a general class of ARIMA model
Identify the model, which can be tentatively entertained
Estimate parameters in the tentatively entertained model
No
Diagnosis Checking Is the model adequate?
Yes
Use this model to generate forecast

Fig. 3.1: The Box-Jenkins methodology for optimal model selection

A crucial step in an appropriate model selection is the determination of optimal model parameters. One criterion is that the sample ACF and PACF, calculated from the training data should match with the corresponding theoretical or actual values [11, 13, 23].

Here n is the number of effective observations, used to fit the model, p is the number of parameters in the model and $\hat{\sigma}_e^2$ is the sum of sample squared residuals. The optimal model order is chosen by the number of model parameters, which minimizes either AIC or BIC. Other similar criteria have also been proposed in literature for optimal model identification.

2.2 Time Series Forecasting Using Artificial Neural Networks

2.2.1 Artificial Neural Networks (ANNs)

In the previous Chapter we have discussed the important stochastic methods for time series modeling and forecasting. Artificial neural networks (ANNs) approach has been suggested as an alternative technique to time series forecasting and it gained immense popularity in last few years. The basic objective of ANNs was to construct a model for mimicking the intelligence of human brain into machine [13, 20]. Similar to the work of a human brain, ANNs try to recognize regularities and patterns in the input data, learn from experience and then provide generalized results based on their known previous knowledge. Although the development of ANNs was mainly biologically motivated, but afterwards they have been applied in many different areas, especially for forecasting and classification purposes [13, 20]. Below we shall mention the salient features of ANNs, which make them quite favorite for time series analysis and forecasting.

First, ANNs are data-driven and self-adaptive in nature [5, 20]. There is no need to specify a particular model form or to make any *a priori* assumption about the statistical distribution of the data; the desired model is adaptively formed based on the features presented from the data. This approach is quite useful for many practical situations, where no theoretical guidance is available for an appropriate data generation process.

Second, ANNs are inherently non-linear, which makes them more practical and accurate in modeling complex data patterns, as opposed to various traditional linear approaches, such as ARIMA methods [5, 8, 20]. There are many instances, which suggest that ANNs made quite better analysis and forecasting than various linear models.

Finally, as suggested by Hornik and Stinchcombe [22], ANNs are universal functional approximators. They have shown that a network can approximate any continuous function to any desired accuracy [5, 22]. ANNs use parallel processing of the information from the data to approximate a large class of functions with a high degree of accuracy. Further, they can deal with situation, where the input data are erroneous, incomplete or fuzzy [20].

2.2.2 The ANN Architecture

The most widely used ANNs in forecasting problems are multi-layer perceptrons (MLPs), which use a single hidden layer feed forward network (FNN) [5,8]. The model is characterized by a network of three layers, viz. input, hidden and output layer, connected by acyclic links. There may be more than one hidden layer. The nodes in various layers are also known as processing elements. The three-layer feed forward architecture of ANN models can be diagrammatically depicted as below:

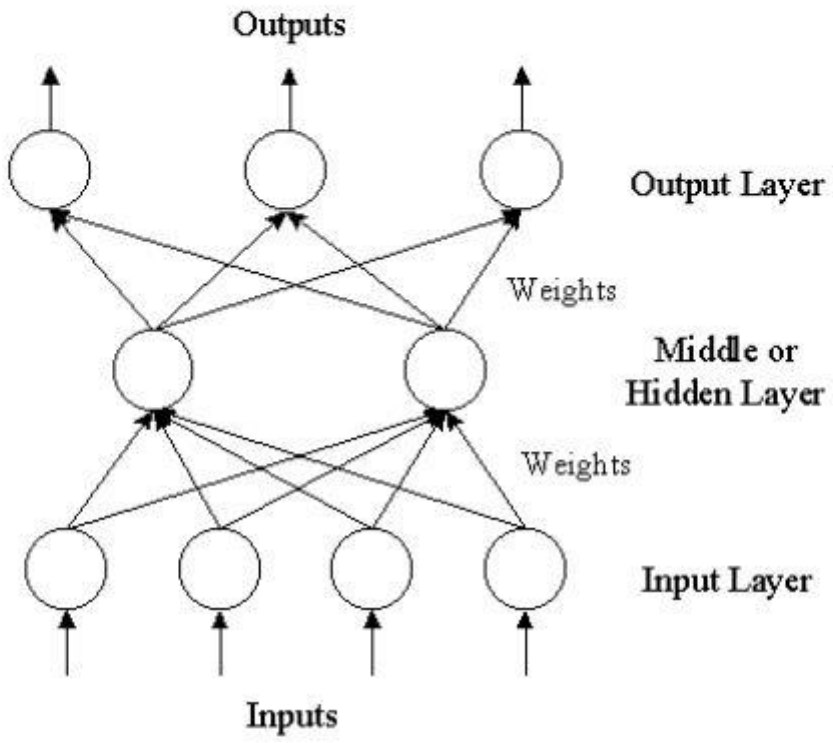


Fig. 4.1: The three-layer feed forward ANN architecture

Here Ψ is the space of all connection weights.

The optimization techniques used for minimizing the error function (4.2) are referred as *Learning Rules*. The best-known learning rule in literature is the Backpropagation or Generalized Delta *Rule* [13, 20].

2.2.3 Time Lagged Neural Networks (TLNN)

In the FNN formulation, described above, the input nodes are the successive observations of the time series, i.e. the target x_t is a function of the values x_{t-i} , (i = 1, 2,..., p) where p is the number of input nodes. Another variation of FNN, viz. the TLNN architecture [11, 13] is also widely used. In TLNN, the input nodes are the time series values at some particular lags. For example, a typical TLNN for a time series, with seasonal period s = 12 can contain the input nodes as the lagged values at time $t-1$, $t-2$ and $t-12$. The value at time t is to be forecasted using the values at lags 1, 2 and 12.

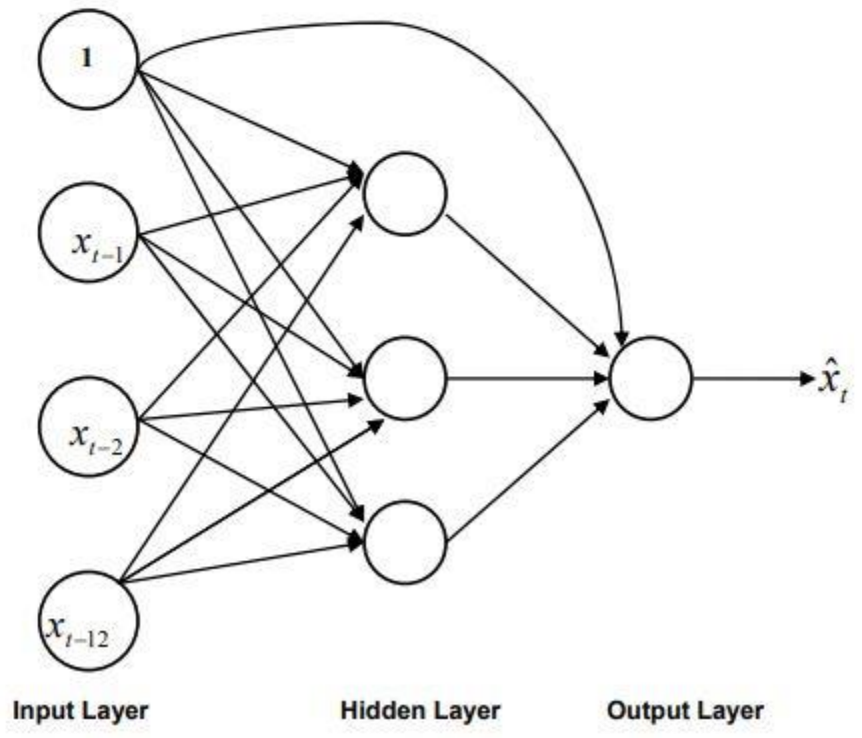


Fig. 4.2: A typical TLNN architecture for monthly data

In addition, there is a constant input term, which may be conveniently taken as 1 and this is connected to every neuron in the hidden and output layer. The introduction of this constant input unit avoids the necessity of separately introducing a bias term.

Faraway and Chatfield [11] used the notation NN ($j1$, $j2$,..., jk ; h) to denote the TLNN with inputs at lags $j1$, $j2$,..., jk and h hidden neurons. We shall also adopt this notation in our upcoming experiments. Thus Fig. 4.2 represents an NN (1, 2, 12; 3) model.

2.2.4 Seasonal Artificial Neural Networks (SANN)

The SANN structure is proposed by C. Hamzacebi [3] to improve the forecasting performance of ANNs for seasonal time series data. The proposed SANN model does not require any preprocessing of raw data. Also SANN can learn the seasonal pattern in the series, without removing them, contrary to some other traditional approaches, such as SARIMA, discussed in Chapter 3. The author has empirically verified the good forecasting ability of SANN on four practical time data sets. We have also used this model in our current work on two new seasonal time series and obtained quite satisfactory results. Here we present a brief overview of SANN model as proposed in [3].

In this model, the seasonal parameter *s* is used to determine the number of input and output neurons. This consideration makes the model surprisingly simple for understanding and implementation. The *i*th and *(i+1)*th seasonal period observations are respectively used as the values of input and output neurons in this network structure. Each seasonal period is composed of a number of observations.

Diagrammatically an SANN structure can be shown as [3]:

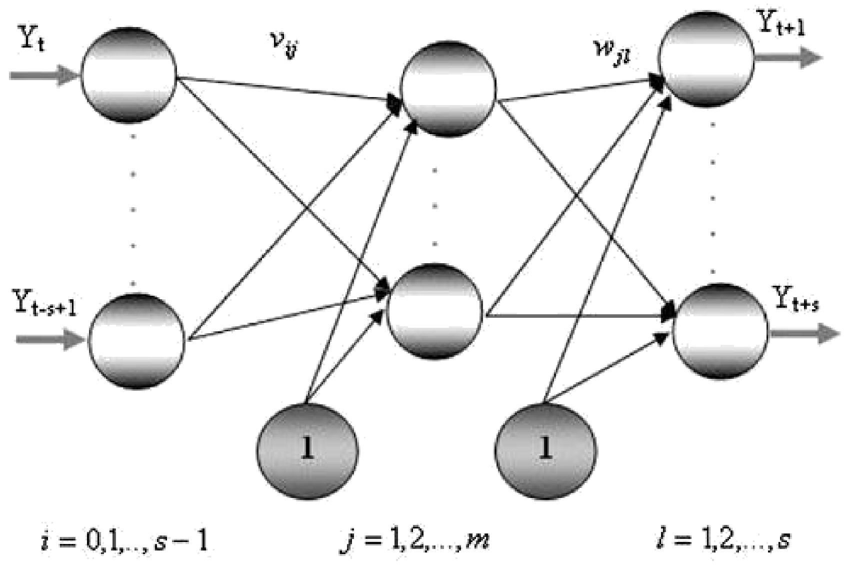


Fig. 2.4: SANN architecture for seasonal time series

Thus while forecasting with SANN, the number of input and output neurons should be taken as 12 for monthly and 4 for quarterly time series. The appropriate number of hidden nodes can be determined by performing suitable experiments on the training data.

2.2.5 Selection of A Proper Network Architecture

So far we have discussed about three important network architectures, viz. the FNN, TLNN and SANN, which are extensively used in forecasting problems. Some other types of neural models are also proposed in literature, such as the *Probabilistic Neural Network (PNN)* [20] for classification problem and *Generalized Regression Neural Network (GRNN)* [20] for regression problem. After specifying a particular network structure, the next most important issue is the determination of the optimal network parameters. The number of network parameters is equal to the total number of connections between the neurons and the bias terms [3, 11].

A desired network model should produce reasonably small error not only on within sample (training) data but also on out of sample (test) data [20]. Due to this reason immense care is required while choosing the number of input and hidden neurons. However, it is a difficult task as there is no theoretical guidance available for the selection of these parameters and often experiments, such as cross-validation are conducted for this purpose [3, 8].

Another major problem is that an inadequate or large number of network parameters may lead to the overtraining of data [2, 11]. Overtraining produces spuriously good within-sample fit, which does not generate better forecasts. To penalize the addition of extra parameters some model comparison criteria, such as AIC and BIC can be used [11, 13]. *Network Pruning* [13] and MacKay's *Bayesian Regularization Algorithm* [11, 20] are also quite popular in this regard.

In summary we can say that NNs are amazingly simple though powerful techniques for time series forecasting. The selection of appropriate network parameters is crucial, while using NN for forecasting purpose. Also a suitable transformation or rescaling of the training data is often necessary to obtain best results.

2.3 Time Series Forecasting Using Support Vector Machines

2.3..1 Concept of Support Vector Machines

Till now, we have studied about various stochastic and neural network methods for time series modeling and forecasting. Despite of their own strengths and weaknesses, these methods are quite successful in forecasting applications. Recently, a new statistical learning theory, viz. the *Support Vector Machine (SVM)* has been receiving increasing attention for classification and forecasting [18, 24, 30, 31]. SVM was developed by Vapnik and his co-workers at the AT&T Bell laboratories in 1995 [24, 29, 33]. Initially SVMs were designed to solve pattern classification problems, such as optimal character recognition, face identification and text classification, etc. But soon they found wide applications in other domains, such as function approximation, regression estimation and time series prediction problems [24, 31, 34].

Vapnik's SVM technique is based on the *Structural Risk Minimization (SRM)* principle [24, 29, 30]. The objective of SVM is to find a decision rule with good generalization ability through selecting some particular subset of training data, called support vectors [29, 31, 33]. In this method, an optimal separating hyperplane is constructed, after nonlinearly mapping the input space into a higher dimensional feature space. Thus, the quality and complexity of SVM solution does not depend directly on the input space [18, 19].

Another important characteristic of SVM is that here the training process is equivalent to solving a linearly constrained quadratic programming problem. So, contrary to other networks' training, the SVM solution is always unique and globally optimal. However a major disadvantage of SVM is that when the training size is large, it requires an enormous amount of computation which increases the time complexity of the solution [24].

Now we are going to present a brief mathematical discussion about SVM concept.

2.3.2 Introduction to Statistical Learning Theory

Vapnik's statistical learning theory is developed in order to derive a learning technique which will provide good generalization. According to Vapnik [33] there are three main problems in machine learning, viz. *Classification*, *Regression* and *Density Estimation*. In all these cases the main goal is to learn a function (or hypothesis) from the training data using a learning machine and then infer general results based on this knowledge.

In case of *supervised learning* the training data is composed of pairs of input and output variables. The input vectors $\mathbf{x} \in X \subseteq \Re^n$ and the output points $y \in D \subseteq \Re$. The two sets X and Y are respectively termed as the input space and output space [29, 33]. $D = \{-1,1\}$ or $\{0,1\}$ for binary classification problem and $D = \Re$ for regression problem.

In case of *unsupervised learning* the training data is composed of only the input vectors. Here the main goal is to infer the inherent structure of the data through density estimation and clustering technique.

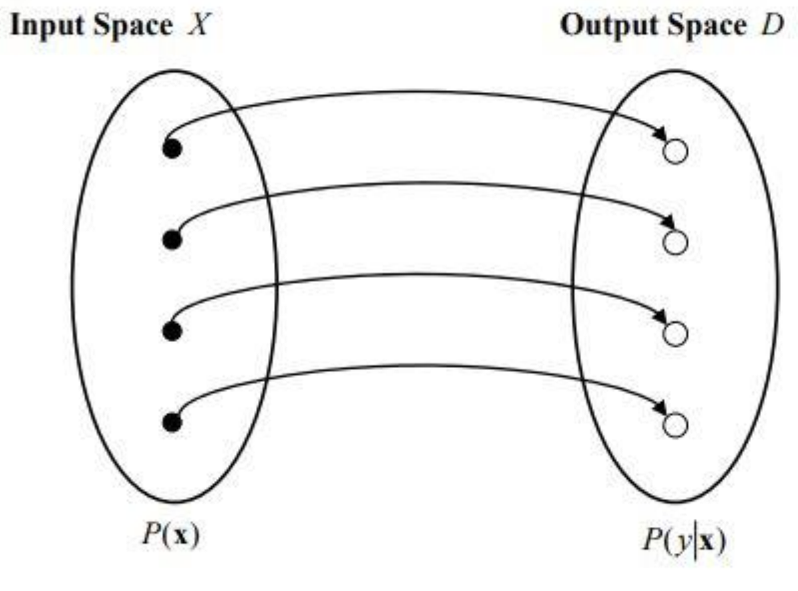


Fig. 5.1: Probabilistic mapping of input and output points

After these descriptions, the learning problem can be visualized as searching for the appropriate estimator function $f : X \to D$ which will represent the process of output generations from the input vectors [29, 33]. This function then can be used for generalization, i.e. to produce an output value in response to an unseen input vector.

2.3.3 Empirical Risk Minimization (ERM)

We have seen that the main aim of statistical learning theory is to search for the most appropriate estimator function $f : X \to D$ which maps the points of the input space X to the output space D. Following Vapnik and Chervonenkis (1971) first a *Risk Functional* is defined on $X \times D$ to measure the average error occurred among the actual and predicted (or classified) outputs due to using an estimator function f. Then the most suitable estimator function is chosen to be that function which minimizes this risk [29, 30, 33].

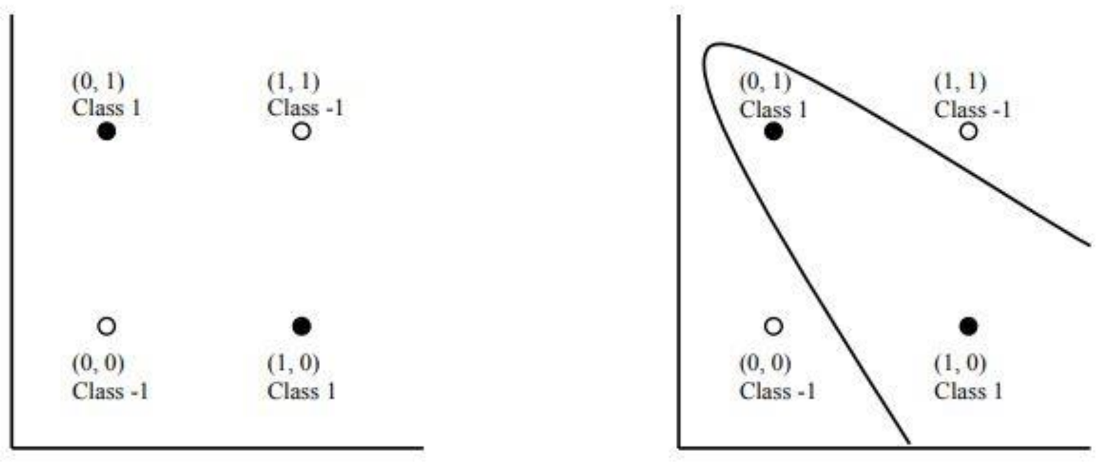
Let us consider the set of functions $F = \{f(\mathbf{x}, \mathbf{w})\}$ that map the points from the input space $X = \Re^n$ into the output space $D \subseteq \Re$ where $\mathbf{w}$ denotes the parameters defining f. Also suppose that y be the actual output point corresponding to the input vector $\mathbf{x}$. Here $P(\mathbf{x}, y)$ is the probability distribution followed by the training data. $L(y, f(\mathbf{x}, \mathbf{w}))$ is known as the *Loss Function* and it can be defined in a number of ways [24, 29, 30].

The most suitable prediction function is the one which minimizes the expected risk $R(f)$ and is denoted by $f0$. This is known as the *Target Function*. The main task of learning problem is now to find out this target function, which is the ideal estimator. Unfortunately this is not possible because the probability distribution $P(\mathbf{x}, y)$ of the given data is unknown and so the expected risk (5.1) cannot be computed. This critical problem motivates Vapnik to suggest the *Empirical Risk Minimization (ERM)* principle [33].

2.3.4 Vapnik-Chervonenkis (VC) Dimension

The VC dimension h of a class of functions F is defined as the maximum number of points that can be exactly classified (i.e. shattered) .

It is proved by Burges in 1998 [1] that the VC dimension of the set of oriented hyperplanes in $\Re^n$ is $(n + 1)$. Thus three points labeled in eight different ways can always be classified by a linear oriented decision boundary in $\Re^2$ but four points cannot. Thus VC dimension of the set of oriented straight lines in $\Re^2$ is three. For example the XOR problem [29] cannot be realized using a linear decision boundary. However a quadratic decision boundary can correctly classify the points in this problem. This is shown in the figures below:



(a) The linearly non-separable XOR problem

(b) A quadratic decision boundary classifying the XOR problem

Fig. 5.2: The two-dimensional XOR problem

2.3.5 Structural Risk Minimization (SRM)

The crucial shortcoming of ERM principle is that in practice we always have a finite set of observations and so it cannot be guaranteed that the estimator function minimizing the empirical risk over *F* will also minimize the expected risk. To deal with this issue the SRM principle was developed by Vapnik and Chervonenkis in 1982 [33]. The key result motivating this principle is that the difference between the empirical and expected risk can be bounded in terms of the VC dimension of the class *F* of estimator functions. Below we present the corresponding mathematical theorem for {0,1} binary classification problem [29]:

Equation (4.3) is the main inspiration behind the SRM principle. It suggests that to achieve a good generalization one has to minimize the combination of the empirical risk and the complexity of the hypothesis space. In other words one should try to select that hypothesis space which realizes the best trade-off between small empirical error and small model complexity. This concept is similar to the *Bias-Variance Dilemma* of machine learning [29].

2.3.6 Support Vector Machines (SVMs)

The main idea of SVM when applied to binary classification problems is to find a canonical hyperplane which maximally separates the two given classes of training samples [18, 24, 29, 31, 33]. Let us consider two sets of linearly separable training data points in $\mathfrak{R}^n$ which are to be classified into one of the two classes $C1$ and $C2$ using linear hyperplanes, (i.e. straight lines). From an infinite number of separating hyperplanes the one with maximum margin is to be selected for best classification and generalization [29, 33]. Below we present a diagrammatic view of this concept:

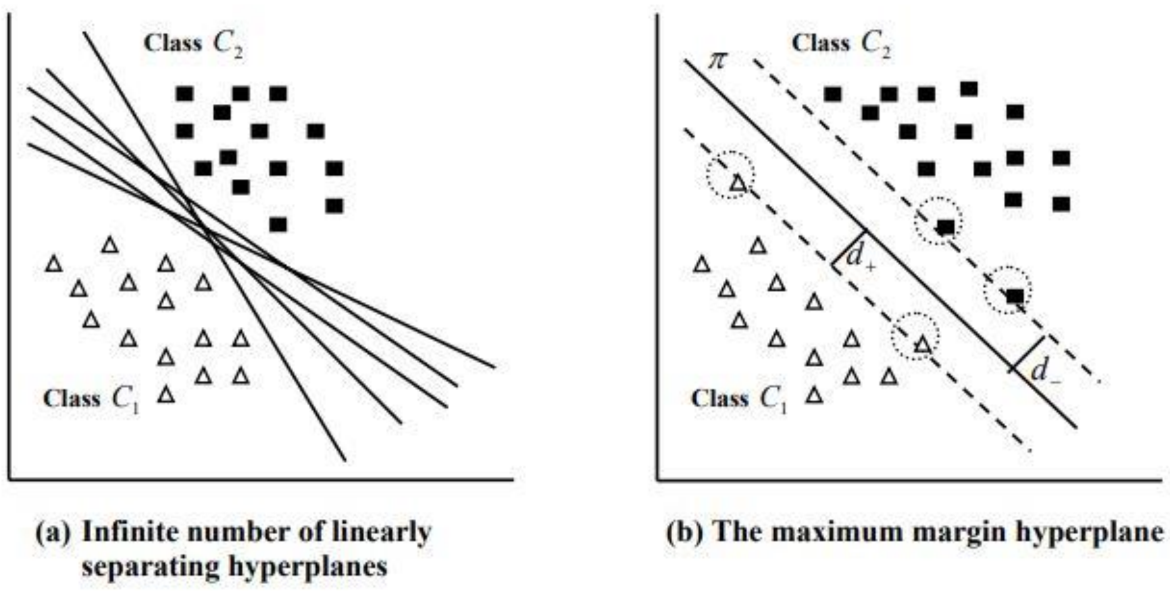

Class C_2
Class C_1
(a) Infinite number of linearly separating hyperplanes
π
Class C_2
d_+
d_-
Class C_1
(b) The maximum margin hyperplane

Fig. 5.3: Support vectors for linearly separable data points

In Fig.4.3(a) it can be seen that there are an infinite number of hyperplanes, separating the training data points. As shown in Fig.4.3(b), $d+$ and $d-$ denote the perpendicular distances from the separating hyperplane to the closest data points of $C1$ and $C2$ respectively. Then either of the distances $d+$ or $d-$ is termed as the margin and the total margin is

$M = d_+ + d_-$. For accurate classification as well as best generalization, the hyperplane which maximizes the total margin is considered as the optimal one and is known as the *Maximum Margin Hyperplane* [29, 33]. Offcourse for this optimal hyperplane $d_+ = d_-$ [29]. The data points from either of the two classes which are closest to the maximum margin hyperplane are known as *Support Vectors* [29, 33]. In Fig.4.3(b) π denotes the optimal hyperplane and circulated data points of both the classes represent the support vectors.

Let us consider that the training set is composed of the input-output pairs $\{\mathbf{x}_i , y_i\}_{i=1}^{N}$, where $\mathbf{x}_i \in \Re^n$, $y_i \in \{-1,1\}$. The goal is to classify the data points into two classes by finding a maximum margin canonical hyperplane. The hypothesis space is the set of functions $f(\mathbf{x}, \mathbf{w}, b) = \text{sgn}(\mathbf{w}^T \mathbf{x} + b)$ where $\mathbf{w}$ is the weight vector, $\mathbf{x} \in \Re^n$ and b is the bias term. The set of separating hyperplanes is given by $\{\mathbf{x} \in \Re^n : \mathbf{w}^T \mathbf{x} + b = 0$, where $\mathbf{w} \in \Re^n , b \in \Re\}$. Using SVM the problem of finding the maximum margin hyperplane reduces to solving a non-linear convex quadratic programming problem (QPP) [29, 32, 33].

Here the slack variables ξ_i are introduced to relax the hard-margin constraints and $C > 0$ is the regularization constant which assigns a penalty to misclassification. Again (4.7) represents a QPP and its solution can be found by applying Lagrange multipliers and Kühn-Tucker conditions. The optimal weight vector and decision function is similar to those in the linearly separable case. The only difference is that in this case the Lagrange multipliers have an upper bound on C , i.e. $0 \le \alpha_i \le C$ and for support vectors $0 < \alpha_i \le C$.

2.3.7 Support Vector Kernels

In SVM applications it is convenient to map the points of the input space to a high dimensional *Feature Space* through some non-linear mapping [18, 19, 29] and the optimal separating hyperplane is constructed in this new feature space. This method also resolves the problem where the training points are not separable by a linear decision boundary. Because by using an appropriate transformation the training data points can be made linearly separable in the feature space. A pictorial view of this idea can be obtained from Fig. 4.4:

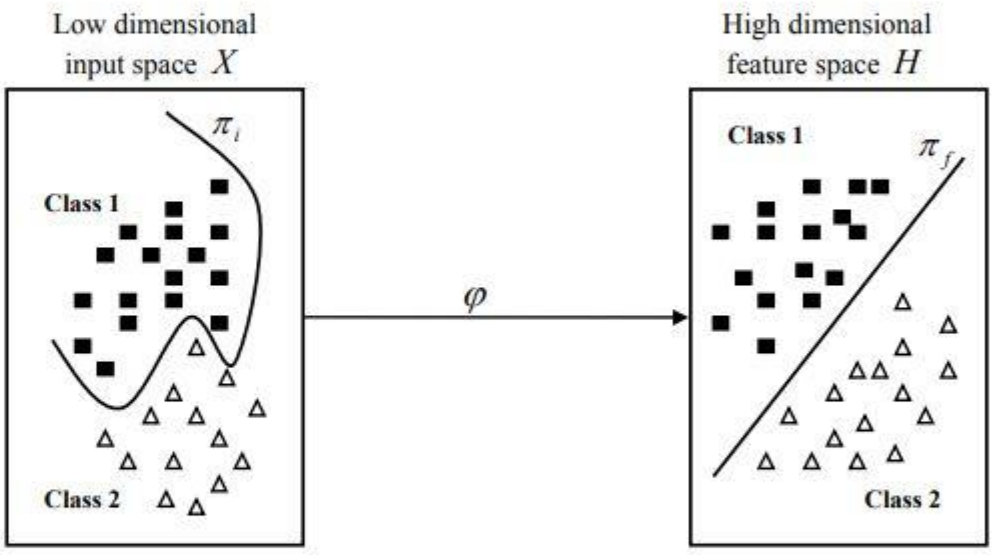

Low dimensional input space X
π_i
Class 1
Class 2
φ
High dimensional feature space H
Class 1
π_f
Class 2

Fig. 5.4: Non-linear mapping of input space to the feature space

In Fig. 4.4, the data points of the input space $X \subseteq \Re^n$ are mapped to the feature space H using the nonlinear mapping $\phi : X \rightarrow H$. Due to this transformation, the linearly non-separable input points can now be separated by a linear decision boundary πf in the feature space

2.3.8 SVM for Regression (SVR)

We have briefly discussed about the SVM techniques used for classification. To use SVM for regression problem Vapnik derived a generalized method [29, 30, 33] which we shall present here in short. Except the outputs $yi \in \Re$ all other variables will have the same meaning

During the past few years extensive research works have been carried out towards the application of SVM for time series forecasting. As a result many different SVM forecasting algorithms have been derived. Some of them are the *Critical Suppot Vector Machine (CSVM)* algorithm, the *Least Square Support Vector Machine (LS-SVM)* [18] algorithm and its variants, viz. the *Recurrent Least Square Support Vector Machine* [19], the *Dynamic Least Square Support Vector Machine (DLS-SVM)* [34] etc. We are now going to present the celebrated DLS-SVM algorithm developed by Y. Fan et al., 2006 [34] for time series forecasting. Before that we shall give an overview of the LS-SVM technique.

2.3.9 The LS-SVM Method

An LS-SVM formulation employs the equality constraints and a sum-squared error (SSE) cost function, instead of quadratic program in traditional SVM. $\| \quad \| /$

Here ϕ is the non-linear mapping to a higher dimensional space and γ is the regularization parameter. The RBF kernel $K(\mathbf{x}, \mathbf{y}) = \exp(-\mathbf{x}-\mathbf{y}^2 \quad 2\sigma^2)$ with the tuning parameter σ can be employed, which satisfies the Mercer's condition [29, 30, 34].

Here $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, ..., \alpha_N]^T$, where $\alpha_i \geq 0$ ($\forall i = 1, 2, ..., N$) are the Lagrange multipliers.

Applying the conditions of optimality, one can compute the partial derivatives of L with respect to $\mathbf{w}, b, e_k, \alpha_k$, equate them to zero and finally eliminating $\mathbf{w}$ and e_k obtain the following linear system of equations [18, 19, 34]. Here $\boldsymbol{\alpha}, b$ are the solutions of the linear system (5.16).

The main benefit of the LS-SVM technique is that it transforms the traditional QPP to a simultaneous linear system problem, thus ensuring simplicity in computations, fast convergence and high precision [18, 34].

2.3.10 The DLS-SVM Method

DLS-SVM [34] is the modified version of the LS-SVM and is suitable for real time system recognition and time series forecasting. It employs the similar concept but different computation method than the recurrent LS-SVM [19]. The key feature of DLS-SVM is that it can track the dynamics of the nonlinear time-varying systems by deleting one existing data point whenever a new observation is added, thus maintaining a constant window size. Keeping in mind the LS-SVM formulation just discussed, let us consider that [34]:

are eliminated. To use the relation (5.20) we only need to know $\mathbf{Q}_N^{-1}$, which is obtained during the solution of (5.16).

When a new data point is added while forecasting then *Pruning* [34] is applied to get rid of the first point and replace it with the new input vector. To remove the first point from the training dataset it is assumed that the new data point has just been added

We shall now conclude this chapter after touching an important point. The success of SVM to produce a close forecast depends a lot on the proper selection of the hyper-parameters such as the kernel parameter σ , the regularization constant γ , the SVR constant ε , etc. An improper choice of these parameters may result in totally ridiculous forecast. As there is no structured way to choose the best hyper-parameters in advance so in practical applications, techniques like cross-validation [24, 34] or Bayesian inference [34] are used. However it should be noted that the values of the optimal hyper-parameters selected in advance could vary afterwards due to different characteristics of future observations [24].

CHAPTER 3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

3.1.1. Survey on Clouds

D K. Richards and G.D. Sullivan [1] describes the methods for using color and texture to discriminate cloud and sky in images captured using a ground based color camera. Neither method alone has proved sufficient to distinguish between different types of cloud, and between cloud and sky in general. Classification can be improved by combining the features using a Bayesian scheme. Malay K. Kundu and Priyank Bagrecha[2] proposed the feature Extraction algorithm is a very important component of any retrieval scheme. The Mband Wavelet Transform based feature extraction algorithm is explained in this paper. Kuo-Lin Hsu, X. Gao, and Soroosh Sorooshian[3] proposed some experiments. It shows that coldtopped cloud pixels with the same values of infrared brightness temperature may belong to different cloud type, thereby, indicating different rain rates at the underlying ground surfaces. It is suggested that the relationship between the satellite cloud-top brightness temperature and surface rainfall rate are non-unique for most pixel-based rainfall estimation algorithms. A scheme is developed, which first classifying cloud types based on the texture features of regional cloud images, then regressing the relationships of cloud brightness temperature and surface rain rate respective to different cloud types using the radar rainfall data. With the separation of cloud-texture types, estimated rainfall rates can be improved.

A cloud-texture classification approach is introduced to process cloud images and estimates the surface rain rate underlying a cloud pixel referencing the cloud-texture type of the pixel. Instead of determining the surface rain rate based on cloud brightness temperature at a local pixel, as many rainfall estimation algorithms do (see Hsu et al., 1997; Bellerby et al., 2000), this approach extracts the features of cloud texture in a 4° × 4° window to classify the cloud imagery into a number of cloud (texture) groups. The relationship between rainfall rate and cloud pixel brightness temperatures at each assigned cloud-texture group is identified separately using ground-based radar rainfall data. Liu Jian and Xu Jianmin[4] describes an updated operational cloud detection method of FY-2C. Compared with FY2B three channels, FY2C adds one shortwave infrared channel and split infrared channel. Research results testified that shortwave infrared and split infrared channels can be help to detect low cloud and cirrus cloud, especially at night. Anuj Srivastava and Ian H. Jermyn[5], describes the problem of identifying shape classes in point clouds. These clouds contain sampled contours and are corrupted by clutter and observation noise.

Taking an analysis-by-synthesis approach, we simulate highprobability configurations of sampled contours using models learnt from the training data to evaluate the given test data. Yanling Hao, Wei ShangGuan, Yi Zhu, and YanHong Tang[6] describes that cloud image is a kind of useful image which includes abundant information, for acquired this information, the image processing and character extraction method adapt to cloud image has to be used. Content- based cloud image processing and information retrieval (CBIPIR) is a very important problem in image processing and analysis field. The basic character, like color, texture, edge and shape was extracted from the cloud image, and then the cloud image database was provided to store the basic character information.

Since traditional image retrieval method has some limitation, for realized image retrieval accurately and quickly, the CBIR method is adaptive. Aleksey Golovinskiy, Vladimir G. Kim and Thomas Funkhouser[7] states that the design of a system for recognizing objects in 3D point clouds of urban environments. The system is decomposed into four steps: locating, segmenting, characterizing, and classifying clusters of 3D points. Specifically, we first cluster nearby points to form a set of potential object locations (with hierarchical clustering). Then, we segment points near those locations into foreground and background sets (with a graph-cut algorithm). Next, we build a feature vector for each point cluster (based on both its shape and its context).

Finally, we label the feature vectors using a classifier trained on a set of manually labeled 2010 IEEE International Conference on Computational Intelligence and Computing Research objects. Peter S. Masika [8] states that this study attempts to utilize available MSG data for developing simple cloud mask and height algorithms and thereafter compare and determine the relationship between cloud height and observed rainfall on a ground station. A multispectral threshold technique has been used: the test sequence depends on solar illumination conditions and geographical location whereas most thresholds used here were empirically determined and applied to each individual pixel to determine whether that pixel is cloud-free or cloud-contaminated. The study starts from the premise of an acceptable trade-off between calculation speed and accuracy in the output data.

Wei Shangguan; Yanling Hao; Zhizhong Lu; Peng Wu[10] states that the recent development of cloud image processing technology has become very quick; the research aspects concentrate on judge the cloud type and classify the cloud mainly.

These image processing methods relate to the subject category like image processing and pattern recognition etc; it has become one of the fields of most quickly development in the research of image processing technology. In cloud image, texture is an very important feature, since cloud image has clear texture structure, the computer texture analysis provide perfect future for study and analyze all kinds of cloud image. Variation method is a new image segmentation method development in recent years, which is adapt to modeling and extract deformable contour of random shape. In cloud image, recognize the target object has great application meaning.

3.1.2. IMAGE PROCESSING TECHNIQUES

A. Image enhancement:

It refers to sharpening or blurring, of image features like contrast or boundaries to make graphic display more visible for display & analysis. Process of enhancement does not increase the inside information in data. That includes contrast manipulation and gray level, noise reduction, filtering, interpolation, edge sharpening and magnification.

B. Image restoration:

It is mostly combined with filtering to observe image to minimize effect of noise or degradations. Effect of image restoration is depended on accuracy and extent of the knowledge of on filter design and process of degradation. The difference between Image restoration and image enhancement is concerned with more accentuation or extraction of image features [3].

C. Image compression:

It is associated with minimizing number of bits that are required to represent an image in digital form. Use of compression are in different places like broadcast TV, military communication via aircraft, remote sensing via satellite, medical images in computer tomography, radar, facsimile transmission, teleconferencing, for educational or business documents, magnetic resonance imaging and also pictures, satellite images, digital radiology, motion, geological surveys, weather maps and so on.

3.1.3. COMMON FILE FORMATS

Common file formats that use mostly are: JPEG that is a very efficient and much information per byte compressed 24 bit format that widely used in web and Internet.GIF that is 8-bit or 256 color and non-destructive compressed format that is mostly used for web.TIFF that is standard 24 bit publication format that compresses nondestructively. PSD that is a dedicated Photoshop format which keeps including all the layers all the information in the image.PS that is a standard vector format that has numerous sub-standards that could be difficult to transmit across platforms[4].

3.1.4. CLOUD AND CLOUD TYPES

CLOUD TYPE	DESCRIPTION
Cirrus	Thin, white and feathery appearance and mostly white patches or narrow bands.
Cirrocumulus	Thin white bands or ripples, sheet, or layered of clouds without shading.
Cirrostratus	High, milky white like appearance. They are transparent, whitish veil clouds with a fibrous (hair-like) or smooth appearance
Altocumulus	Bumpy rounded masses, cotton ball appearance, white and/or gray patch sheet or layered clouds

Altostratus	Transparent blue/gray clouds sheets or fibrous clouds that totally or partially cover the sky
Nimbostratus	They are continuous rain cloud also known as storm cloud
Stratocumulus	Gray or whitish layer with sheet, or layered clouds which almost always are dark
Stratus	Cover large portion of sky, thin, sheet-like, gray and thick
Cumulus	Cauliflower like appearance with bulging upper parts
Cumulonimbus	The thunderstorm cloud, heavy and dense cloud in the form of a mountain or huge tower

Table 3.1 CLOUD AND ITS TYPES

High Level Clouds (above 20, 000 feet) are the Cirrus, Cirrocumulus, Cirrostratus, and Cumulonimbus. Middle Level Clouds (between 6500–20,000 feet) are the Altocumulus, Altostratus. Low Level Clouds (below 6500) are the Nimbostratus, Stratocumulus, Stratus, and the Cumulus.

3.2 PROPOSED SYSTEM

Time series Analysis refers to obtaining up of result from a number of observations that are in chronological order. Acquired samples are processed in MATLB 2019a using image processing techniques. Images thus collected get processed and the information are collected. Using the shape, colour, morphology and other properties of the cloud, name of the type of cloud can be obtained. By using the name of the cloud, individual properties can be enlisted. Depending on density, humidity and temperature can be obtained as numerical value.

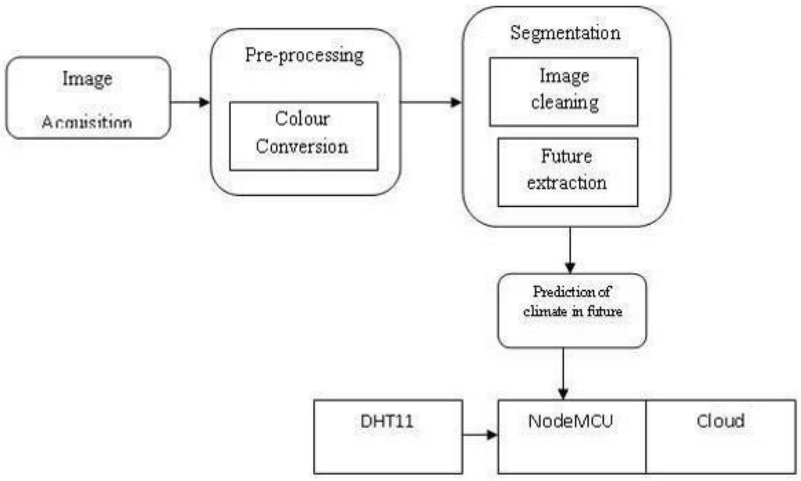
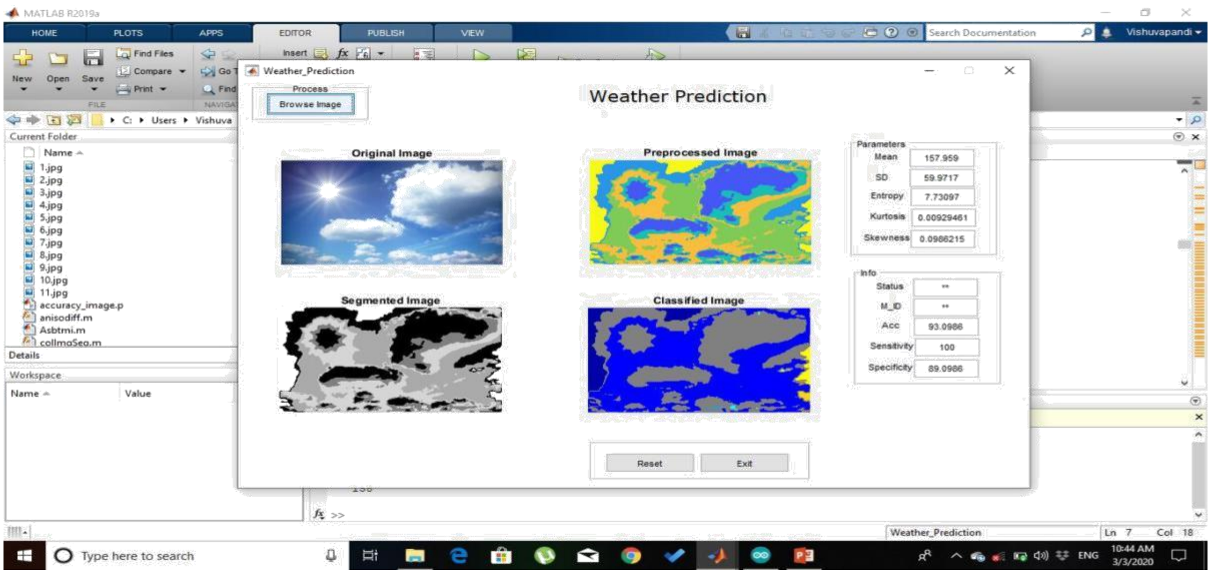


Fig.3.1. Architecture of the proposed solution

To increase the accuracy in prediction, using Arduino, Node MCU and DHT11, real time temperature and humidity can be obtained.



Parameters

Mean	157.959
SD	59.9717
Entropy	7.73097
Kurtosis	0.00929461
Skewness	0.0986215

Info

Status	**
M_ID	**
Acc	93.0986
Sensitivity	100
Specificity	89.0986

Fig.3.2. Screenshot of Image Processing MATLAB Tool

From obtained data, steady and accurate temperature & humidity can be collected. Data obtained through image processing and DHT11 are sampled together to get a average sum of the upcoming climate. As five different samples are calculated, time series average value is obtained. The calculated value is given to be the climatic condition of the next day as per done experiments and calculations. This can be considered to be one of accurate methods unlike online prediction methods since it also employs real time observations.

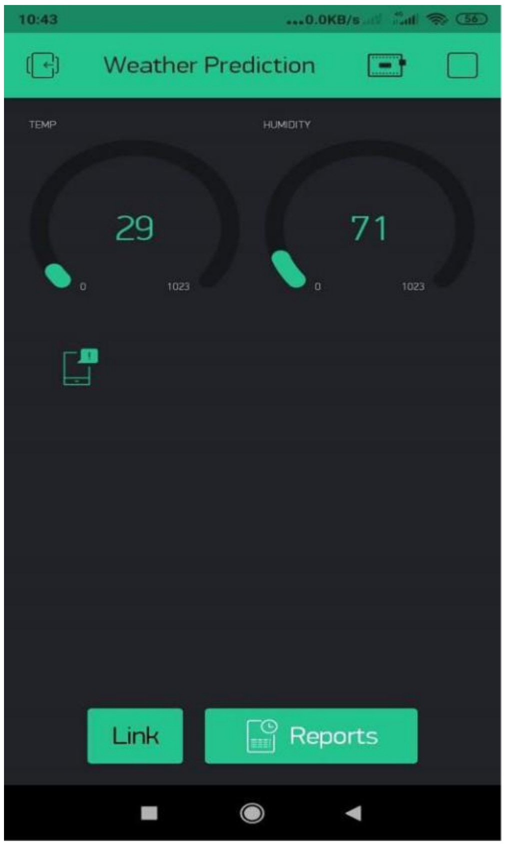


Fig.3.3. Screenshot from Android App Blynk for IOT

CHAPTER 4

TECHNICAL DESCRIPTION

4.1. HARDWARE

- **HDD:** >90GB
- **PROCESSOR:** >Pentium IV 2.4GHz
- **SYSTEM TYPE:** 32bit / 64 bit
- **RAM:** >2GB
- **OS:** WINDOWS 7/8/8.1/10
- Arduino, Node MCU, DHT11

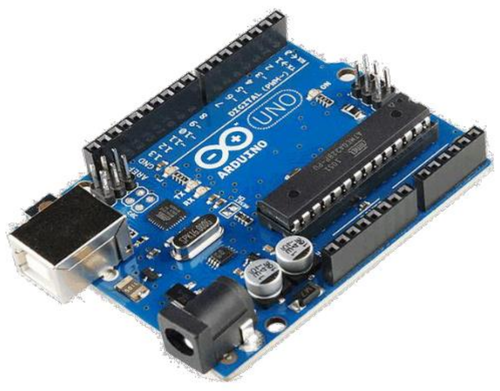


Fig 4.1 Arduino Board

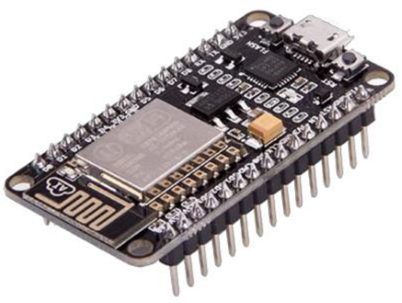


Fig 4.2 Node MCU

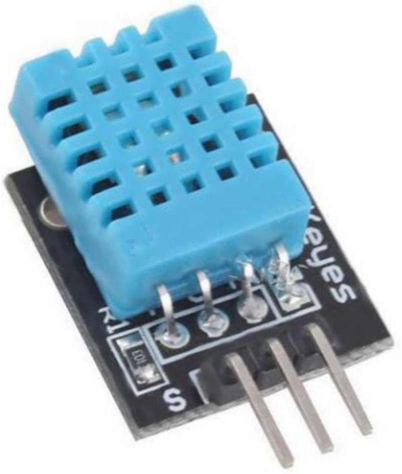


Fig 4.3 DHT 11

4.2 SOFTWARE

- **IMAGE PROCESSING TOOL** : MATLAB 2019a
- **REAL TIME WEATHER DETECTION TOOL** : Arduino Genuino
- **TOOL BOX** : Image Processing Tool Box

Fig 4.4 MATLAB R2019a Software

CHAPTER 5
SYSTEM DESIGN AND IMPLEMENTATION

5.1 SYSTEM ARCHITECTURE

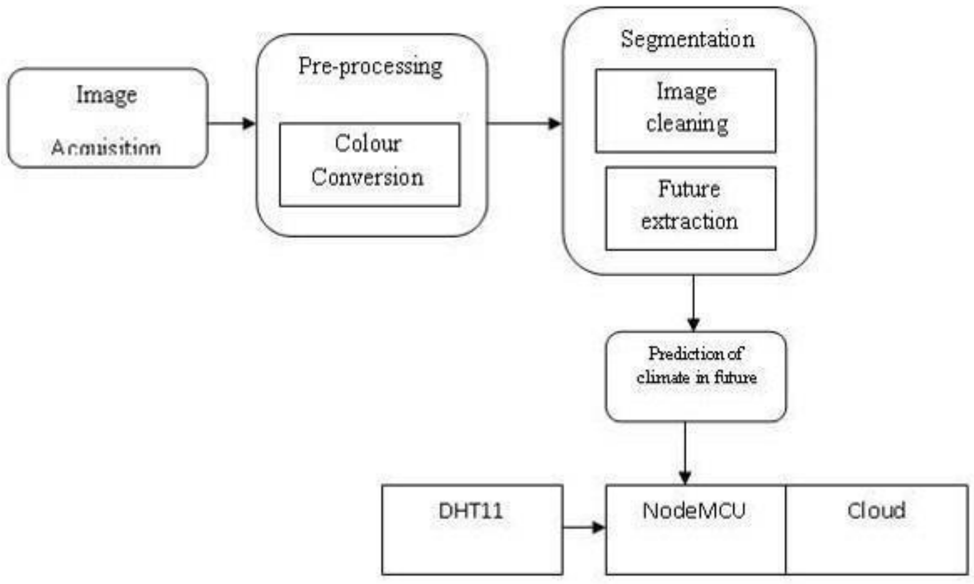


Fig 5.1 Architecture of Time Series Application

Time series Analysis refers to obtaining up of result from a number of observations that are in chronological order. Acquired samples are processed in MATLB 2019a using image processing techniques. Images thus collected get processed and the information are collected. Using the shape, colour, morphology and other properties of the cloud, name of the type of cloud can be obtained. By using the name of the cloud, individual properties can be enlisted. Depending on density, humidity and temperature can be obtained as numerical value**.**

To increase the accuracy in prediction, using Arduino, Node MCU and DHT11, real time temperature and humidity can be obtained

From obtained data, steady and accurate temperature & humidity can be collected. Data obtained through image processing and DHT11 are sampled together to get a average sum of the upcoming climate.

As five different samples are calculated, time series average value is obtained. The calculated value is given to be the climatic condition of the next day as per done experiments and calculations. This can be considered to be one of accurate methods unlike online prediction methods since it also employs real time observations.

5.2 EXPERIMENTAL RESULTS

5.2.1 A Brief Overview

Perhaps this is the chapter for which the reader is most eagerly waiting. After gaining a reasonable knowledge about time series modeling and forecasting from the previo—us chapters, we are now going to implement them on practical datasets.

In this current book, till now we have considered six time series, taken from various sources and research works. All the associated programs are written in MATLAB. To judge forecast performances of different methods, the measures MAD, MSE, RMSE, MAPE and Theil's U-statistics are considered. For each dataset, we have presented our obtained results in tabular form. Also in this chapter we have used the term *Forecast Diagram* to mean the graph showing the test (actual) and forecasted data points. In each forecast diagram, the solid and dotted line respectively represents the actual and forecasted observations.

In all the experiments we have used the RBF kernel for SVM training. Crossvalidation is applied to determine the optimum values of σ , C, n and N, where n and N are respectively the dimension and number of input vectors with $(n + N)$ being the size of the training set. The experiments involving all the techniques except SVM are iterated a large number of times. Also rescaling and data transformations are used to some of the datasets.

5.2.2 The Canadian Lynx Dataset

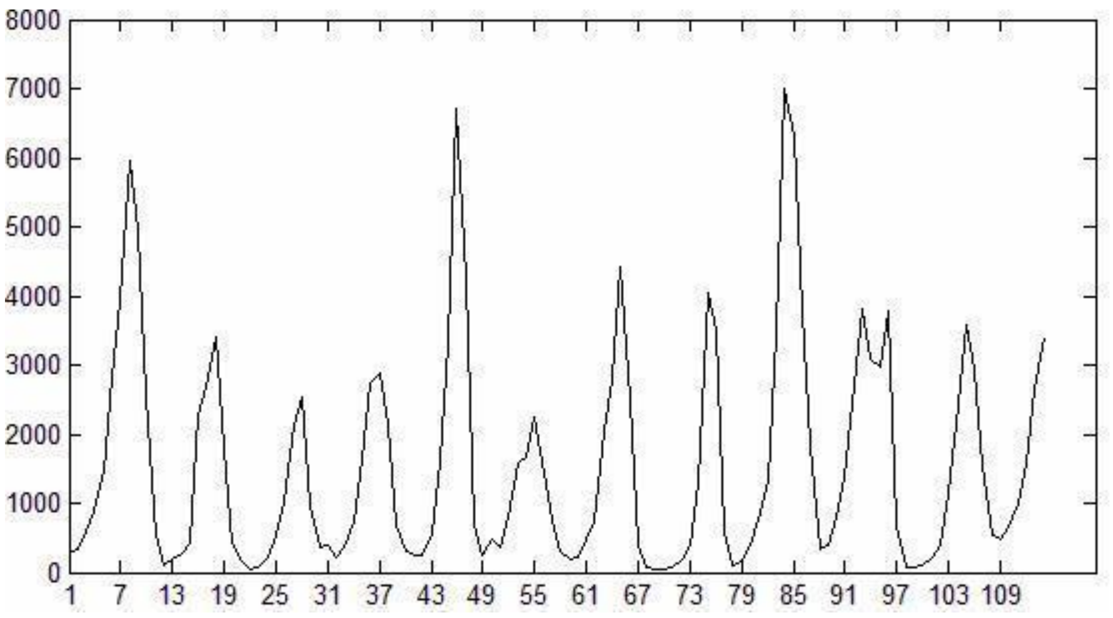


Fig. 5.2.1: Canadian lynx data series (1821-1934)

The lynx series shown in Fig. 7.2.1 contains the number of lynx trapped per year in the MacKenzie River district of Northern Canada from 1821 to 1934. It has been extensively studied by time series analysts like G. P. Zhang [8] and M. J. Campbell et al. [26].

The lynx dataset has total 114 observations. Out of these the first 100 are considered for training and the remaining 14 for testing. An AR model of order 12 has been found to be the most parsimonious ARIMA model for this series [8]. Below we have shown the sample ACF and PACF plots for the lynx data series.

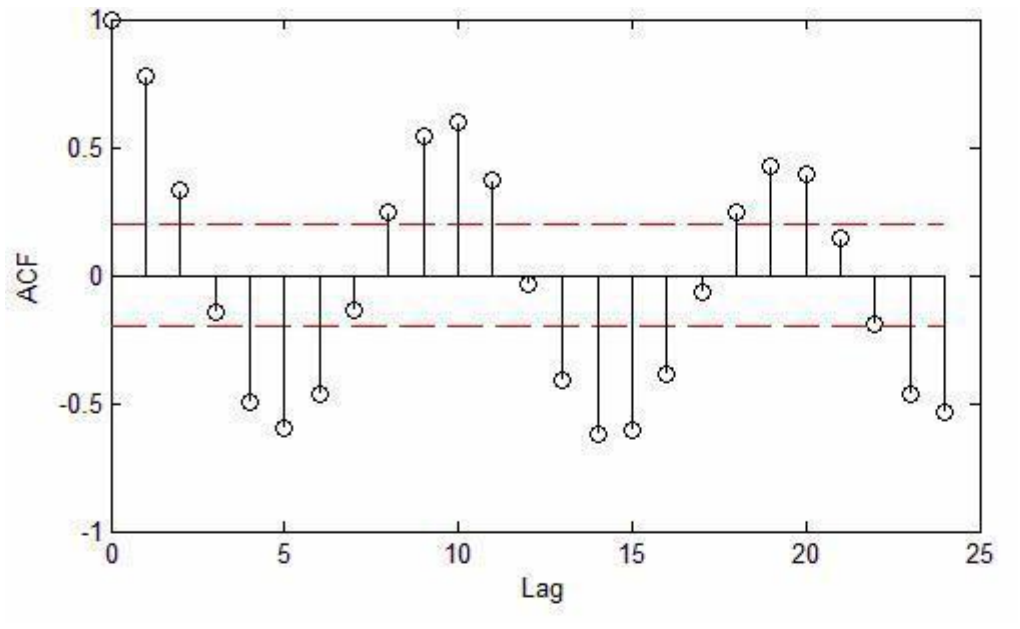


Fig. 5.2.2: Sample ACF plot for lynx series

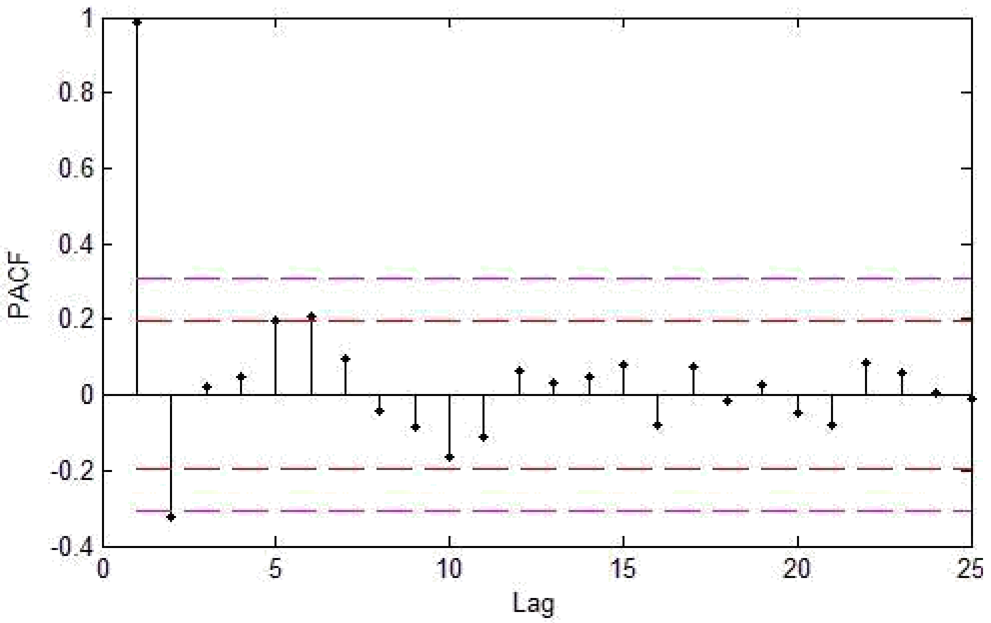


Fig. 5.2.3: Sample PACF plot for lynx Series

Following Hipel and McLeod [23], we have considered the ACF and PACF values up to lag 25, i.e. one-fourth of the training size and 95% confidence level. From these plots we can determine the type and order of the adequate model required to fit the series. Moreover as the ACF values attenuate rapidly for increasing lags, we can assume that the lynx series is stationary.

Later this fact is also justified by the unit root test.

Similar to [8, 26] we have transformed the lynx series using logarithms to the base Then we have employed AR(12), ARMA(12, 9), ANN and SVM to the series. Following G. P. Zhang [8],

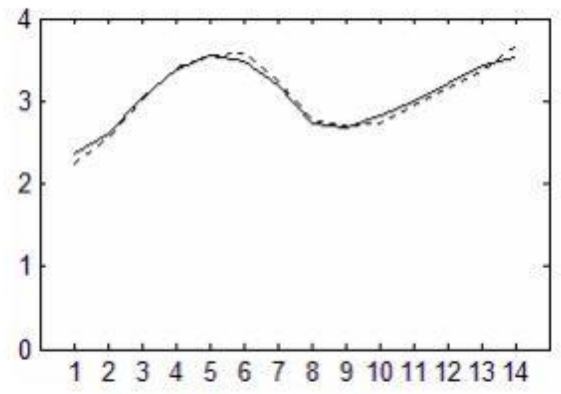
a 7 × 5 ×1 ANN structure is used. Though the fitted ARMA(12, 9) model is not parsimonious, but still it generates good forecasts, as can be seen from Table 7.2. Also in this table, for SVM we have given in bracket the optimal hyper-parameter values found by cross validation.

The performance measures obtained for the lynx series are shown in Table 7.2:

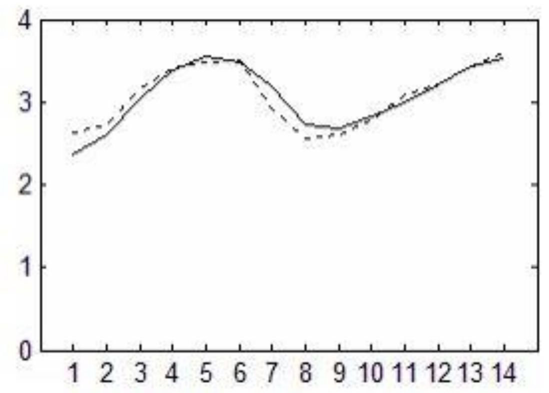
Table 5.2: Forecast results for Canadian lynx time series

Method	**MSE**	**MAD**	**RMSE**	**MAPE**	**Theil's U Statistics**
AR(12)	0.005123	0.058614	0.071577	1.950160%	0.007479
ARMA(12,9)	0.016533	0.096895	0.128581	3.409039%	0.013402
ANN	0.012659	0.066743	0.112512	2.392407%	0.017836
SVM $\sigma = 0.8493$, $C = 1.4126$, $n = 3$, $N = 97$	0.052676	0.173318	0.229513	5.811812%	0.023986

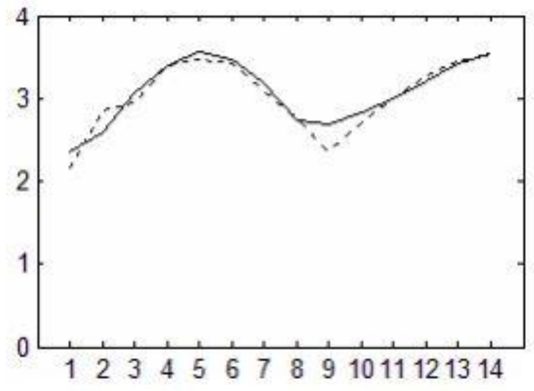
It can be seen from the above table that the best performance is obtained by using AR(12) model. The four forecast diagrams for lynx series are shown in Fig. 7.2.4.



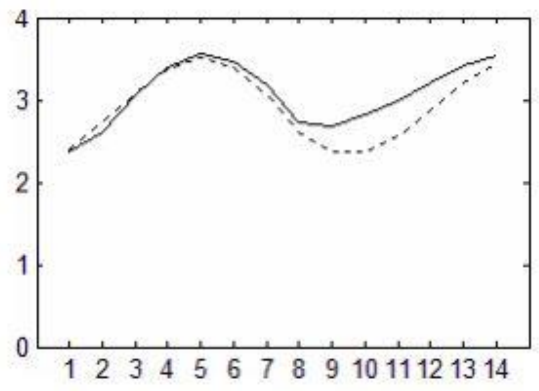
(a) AR(12) forecast diagram



(b) ARMA(12, 9) forecast diagram



(c) ANN forecast diagram



(d) SVM forecast diagram

Fig. 7.2.4: Forecast diagrams for lynx series

The four forecast diagrams presented in Fig. 7.2.4 are corresponding to the transformed lynx series. The good forecasting performance of the fitted AR(12) model can be visualized from Fig. 7.2.4 (a) as the two graphs are very close to each other.

6.3 The Wolf's Sunspot Dataset

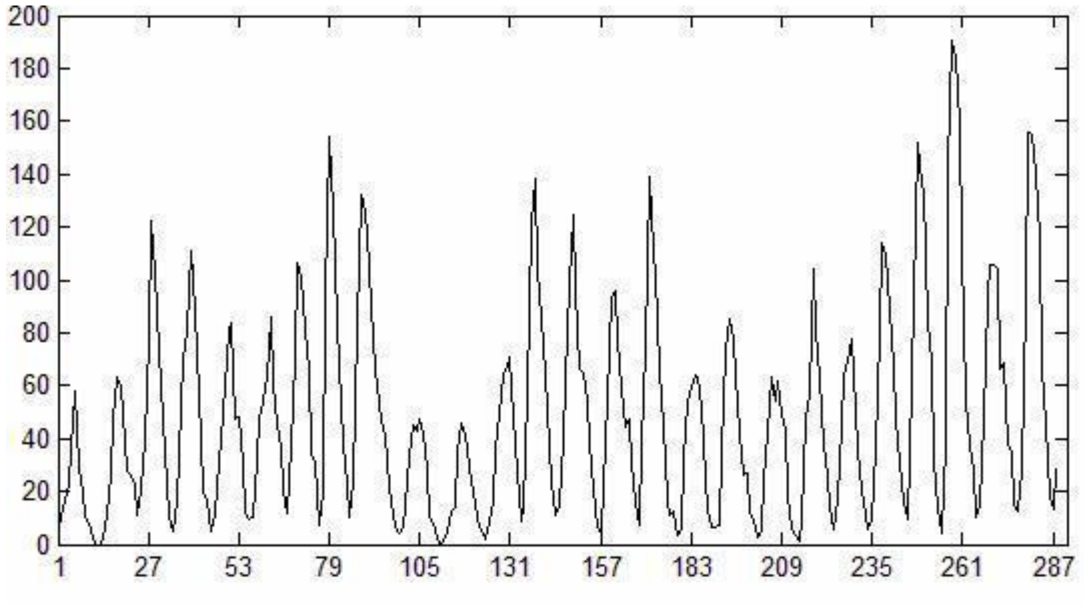


Fig. 5.3.1: Wolf's sunspot data series (1700-1987)

The Wolf's sunspot series shown in Fig. 7.3.1 represents the annual number of sunspots from 1700 to 1987 [8]. This data series is considered as non-linear and it has important practical applications in many domains, such as geophysics, environmental science and climatology [8]. From Fig. 7.3.1 it can be seen that the sunspot series has a mean cycle of about 11 years.

The sunspot dataset has a total of 288 observations, from which the first 221 (i.e. 1700 to 1920) are used for training and the remaining 67 (i.e. 1921 to 1987) for testing. An AR(9) model has been found to be the most parsimonious ARIMA model for this series [8]. The sample ACF and PACF plots with 95% confidence level for the sunspot series are shown in the two figures below:

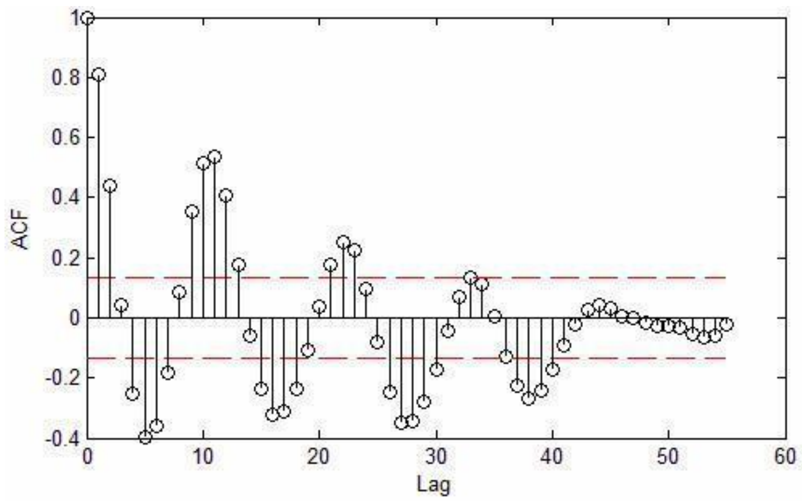


Fig. 5.3.2: Sample ACF plot for sunspot series

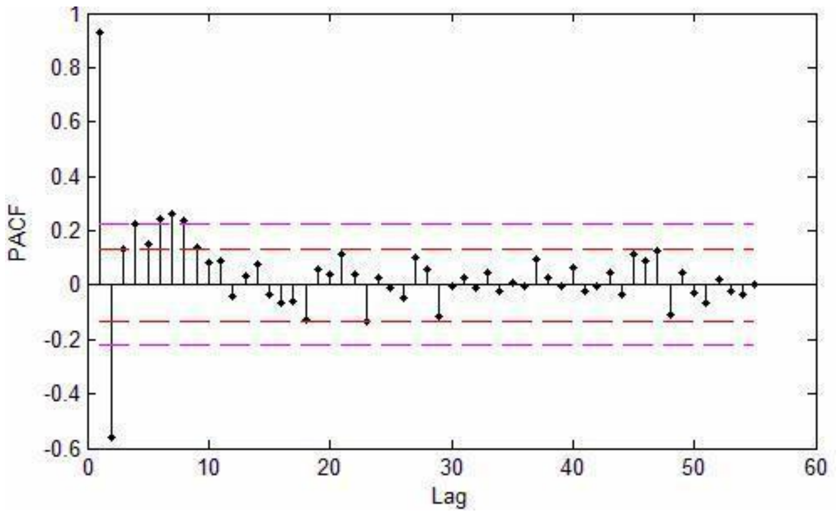
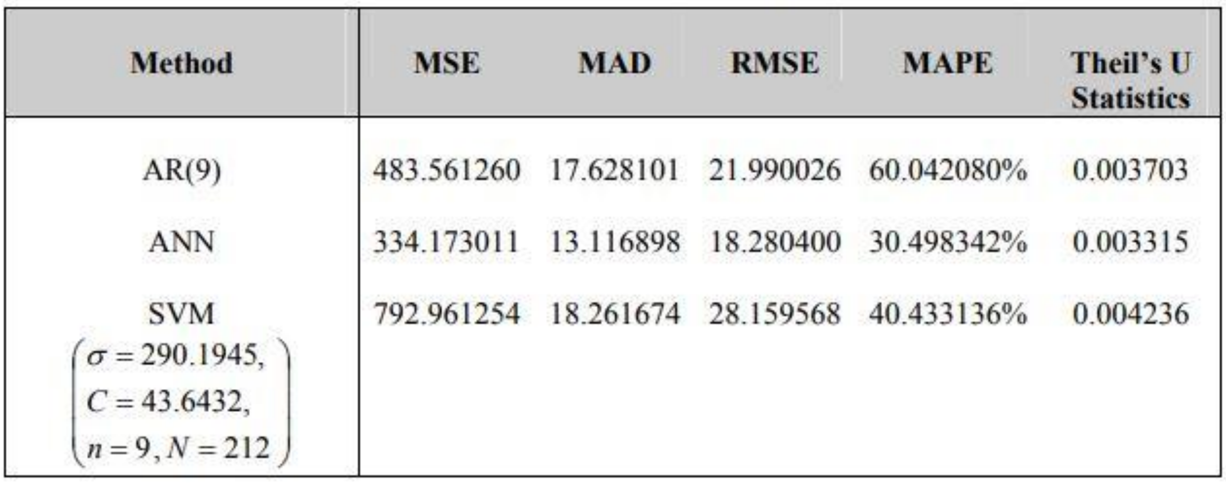


Fig. 5.3.3: Sample PACF plot for sunspot series

The sunspot series follows stationary behavior, which can be seen from the above two plots. We have also analytically verified this fact by unit root test. The models we have fitted to the sunspot series are AR(9), ANN (a 4 × 4 ×1 structure) and SVM.

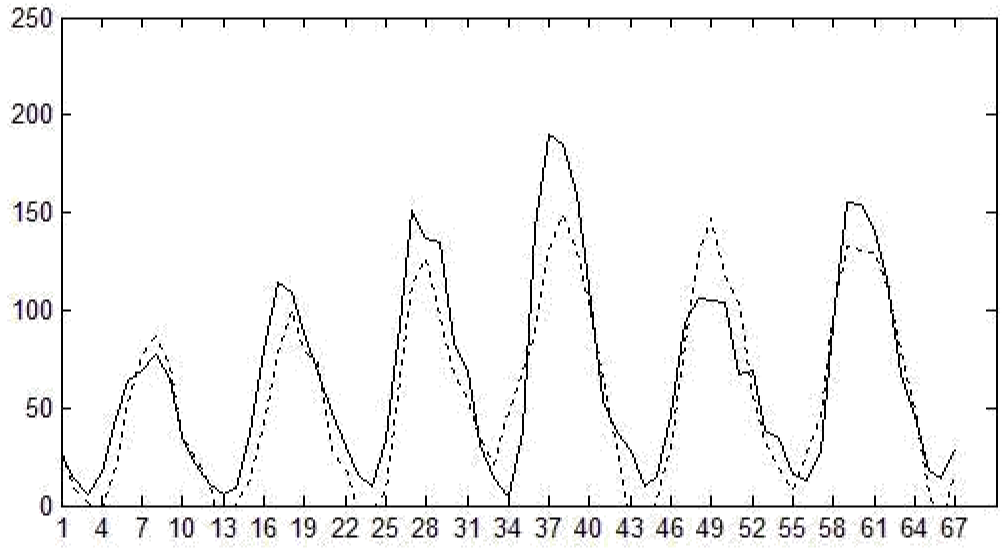
The obtained performance measures for the sunspot series are shown in the Table 7.3:

Table 7.3: Forecast results for sunspot series

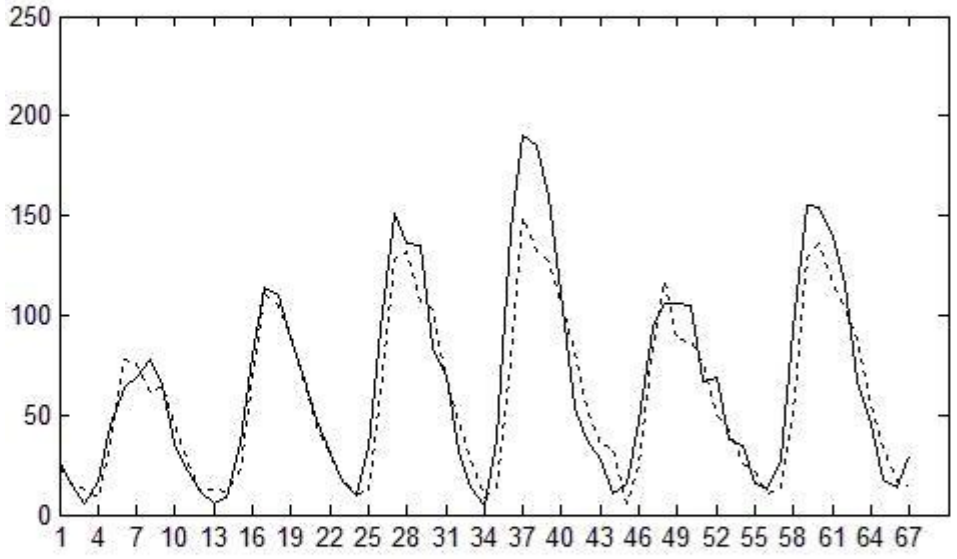


Method	MSE	MAD	RMSE	MAPE	Theil's U Statistics
AR(9)	483.561260	17.628101	21.990026	60.042080%	0.003703
ANN	334.173011	13.116898	18.280400	30.498342%	0.003315
SVM $\begin{pmatrix} \sigma = 290.1945, \\ C = 43.6432, \\ n = 9, N = 212 \end{pmatrix}$	792.961254	18.261674	28.159568	40.433136%	0.004236

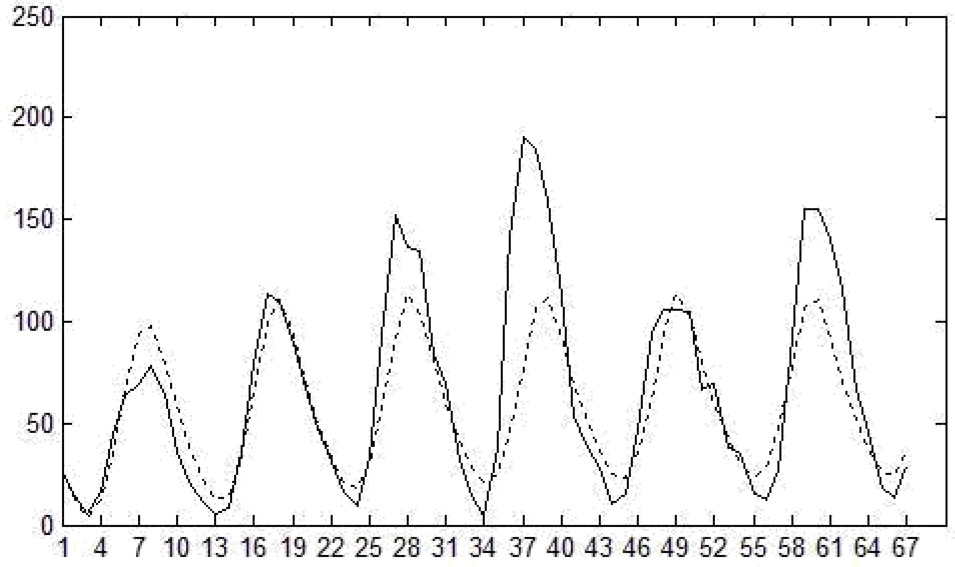
From the above table we can see that the forecasting performance of ANN is best in our experiments for the sunspot series. We have transformed the series in the range [1 2] while fitting ANN model and in the range [100 400] while fitting SVM model to achieve good forecast results. The optimal SVM hyper-parameters obtained by crossvalidation for the transformed sunspot series are shown in Table 7.3. The three forecast diagrams for the sunspot series are shown in the figures below:



(a)AR(9) forecast diagram



(b) ANN forecast diagram



(c) SVM forecast diagram

Fig. 5.3.4: Forecast diagrams for sunspot series

From the three forecast diagrams presented in Fig. 7.3.4, we can have a graphical comparison between the actual and forecasted observations for the sunspot series, obtained by using three different techniques. It can be seen that in Fig. 7.3.4 (b) the ANN forecasted series closely resembles with the original one. However in case of SVM prediction a significant deviation between the forecasted and test values can be seen from Fig. 7.3.4 (c).

5.4 The Airline Passenger Dataset

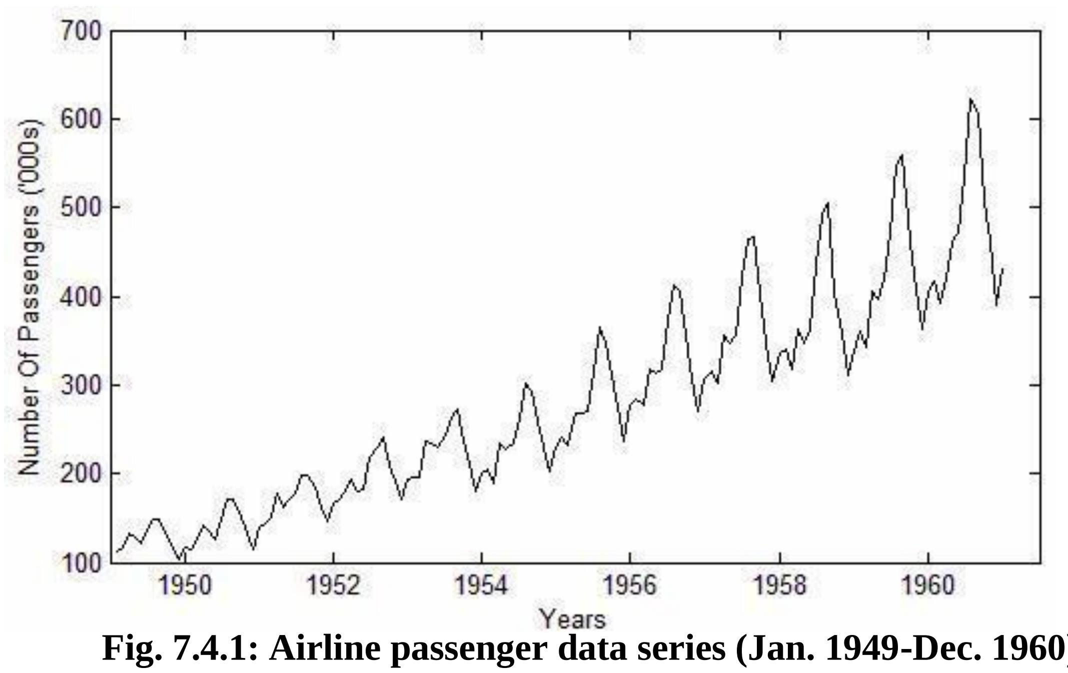


Fig. 7.4.1: Airline passenger data series (Jan. 1949-Dec. 1960)

The airline passenger series, shown in Fig. 7.4.1 represents the monthly total number of international airline passengers (in thousands) from January 1949 to December 1960 [3, 11, 13]. It is a famous time series and is used by many analysts including Box and Jenkins [6]. The important characteristic of this series is that it follows a multiplicative seasonal pattern with an upward trend, as can be seen from Fig. 7.4.1. The airline passenger series has total 144 observations, out of which we have used the first 132 for training and the remaining 12 for testing. Due to the presence of strong seasonal variations, the airline data is non-stationary. This can also be clarified from its sample ACF and PACF plots, as given below:

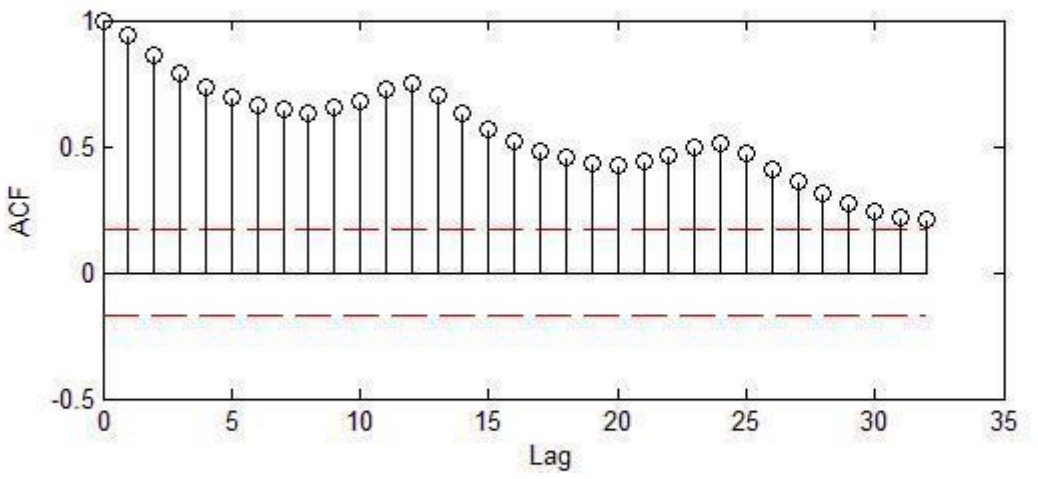


Fig. 5.4.2: Sample ACF plot for airline series

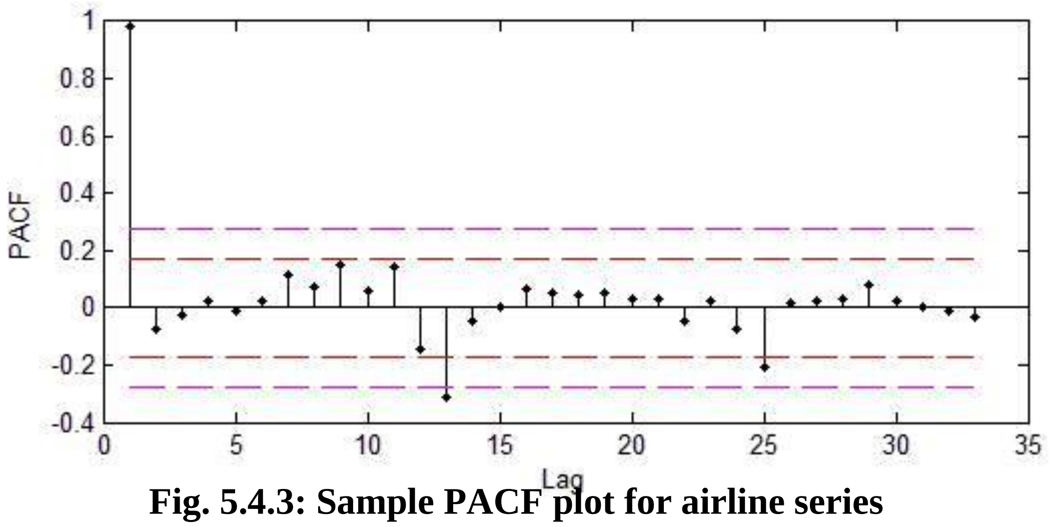


Fig. 5.4.3: Sample PACF plot for airline series

Fig. 6.4.2 shows that there are significantly large sample ACF values at the increasing lags, which do not diminish quickly. This indicates the non-stationarity of the airline data [6, 23]. Box et al. [6] used natural logarithmic transformation and seasonal differencing to remove non-stationarity from the series. Then they have fitted the SARIMA(0,1,1) × $(0,1,1)^{12}$ model, which according to them is the best stochastic model for the airline data. Due to this reason SARIMA(0,1,1) × $(0,1,1)^{12}$ is often termed as the *Airline Model* [11]. Faraway and Chatfield used thirteen different ANN structures to the airline data and compared the results obtained. Also C. Hamzacebi in 2008 [3] experimented by fitting three SANN models to this data.

In this work we have fitted three different ANN, three SANN, SARIMA(0,1,1) × $(0,1,1)^{12}$ and SVM models to the airline data. Following Box et al. [6] we have also used natural logarithmic transformation and seasonal differencing to the data for fitting the SARIMA model. Also following C. Hamzacebi [3] and Faraway et al.we have rescaled the data after dividing by 100 for fitting ANN and SANN models. However for fitting the SVM model we have used the original airline data.

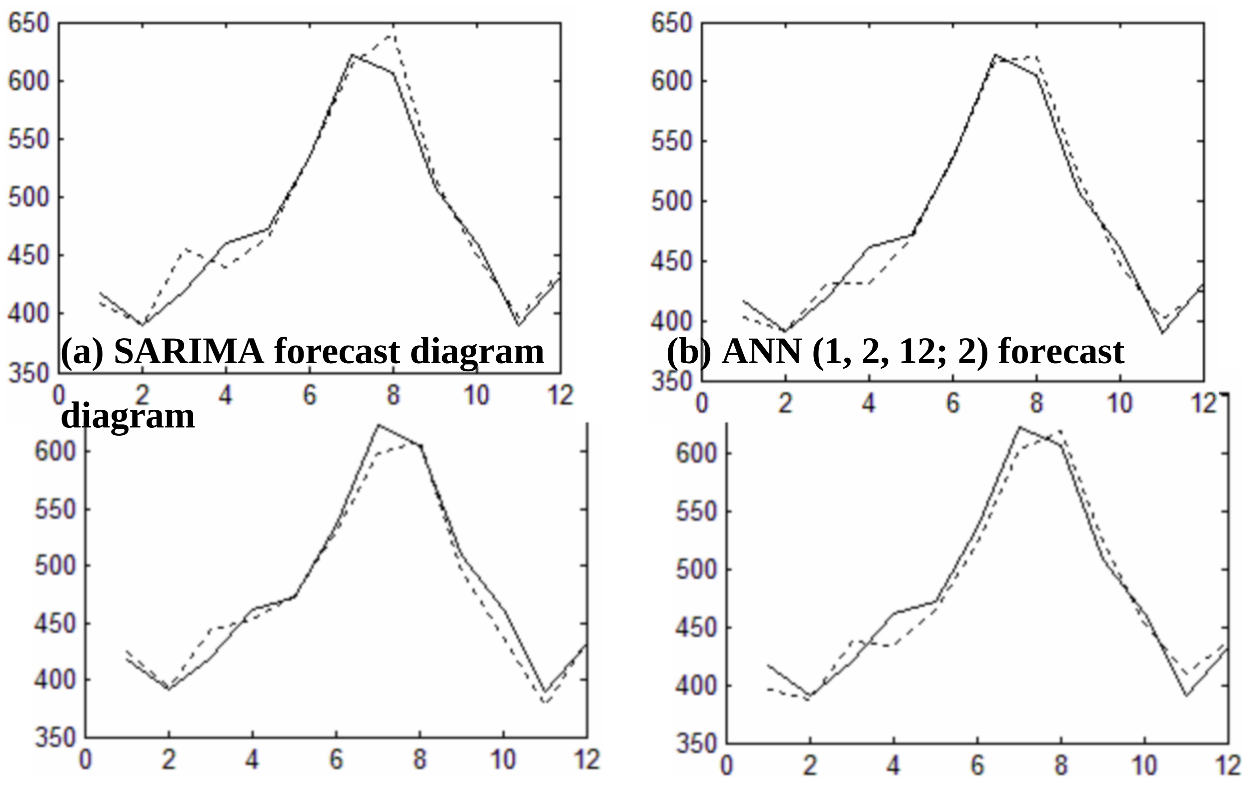
The forecast performance measures, we obtained for the airline data by using the above mentioned models are presented in Table 7.4. As usual the optimal SVM hyper-parameters are shown in bracket in this table. These measures are calculated using the untransformed test and forecasted observations.

Table 7.4: Forecast results for airline passenger time series

Method	MSE	MAD	RMSE	MAPE	Theil's U Statistics
$\text{SARIMA}(0,1,1)\times(0,1,1)^{12}$	189.333893	10.539463	13.759865	2.244234%	0.000060
ANN (1, 12; 2)	285.633562	15.225263	16.900697	3.234460%	0.000074
ANN (1, 2, 12; 2)	248.794863	14.159554	15.773232	3.025739%	0.000068
ANN (1, 12 13; 2)	2532.238561	41.438166	50.321353	8.454268%	0.000232
SANN (1 hidden node)	676.481142	24.311987	26.009251	5.138674%	0.000118
SANN (2 hidden nodes)	275.720525	11.888831	16.604834	2.486088%	0.000071
SANN (3 hidden nodes)	556.054822	17.693719	23.580815	3.624587%	0.000098
SVM $\begin{pmatrix}\sigma = 1.5195\times10^{7}, \\ C = 1.2767\times10^{10}, \\ n = 35, N = 97\end{pmatrix}$	176.885301	10.849932	13.299823	2.336608%	0.000057

From Table 7.4 we can see that SARIMA and SVM generated quite good forecasts for the airline data. Except MAD, all other performance measures obtained by SVM are least relative to the other applied models. ANN (1, 12; 2), ANN (1, 2, 12; 2) and SANN with 2 hidden nodes also produced reasonably well forecasts.

Below we present the four forecast diagrams obtained using $\text{SARIMA}(0,1,1) \times (0,1,1)^{12}$, ANN (1, 2, 12; 2), SANN (2 hidden nodes) and SVM for the airline data:



(a) SARIMA forecast diagram (b) ANN (1, 2, 12; 2) forecast diagram

(c) SANN forecast diagram (d) SVM forecast diagram Fig. 5.4.4: Forecast diagrams for airline passenger series

The four forecast diagrams in Fig. 7.4.4 shows the success of our applied techniques to produce forecasts for the airline data. In particular Fig. 7.4.4 (d) depicts the excellent forecasting performance of SVM for this dataset.

5.5 The Quarterly Sales Dataset

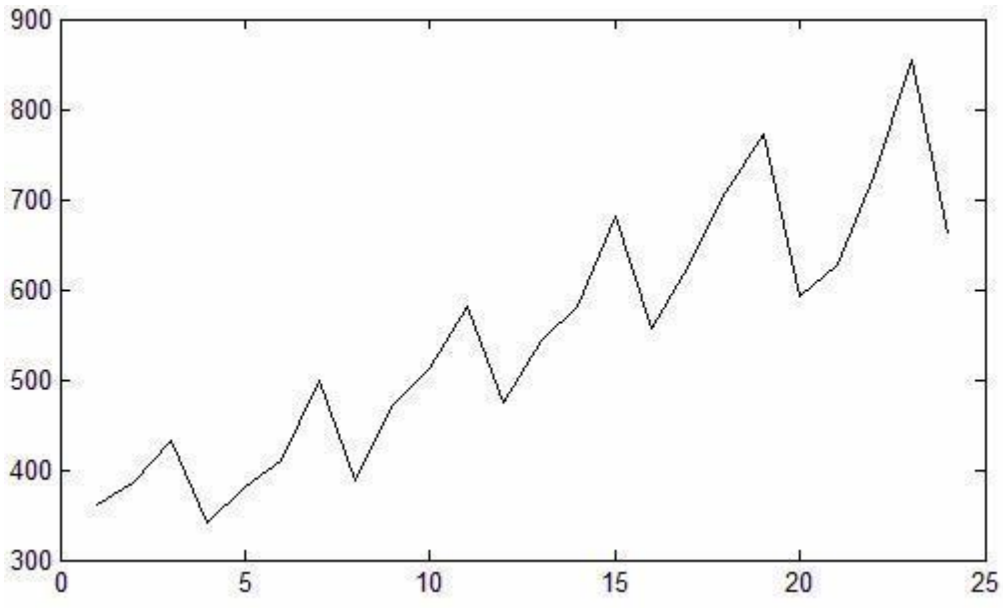


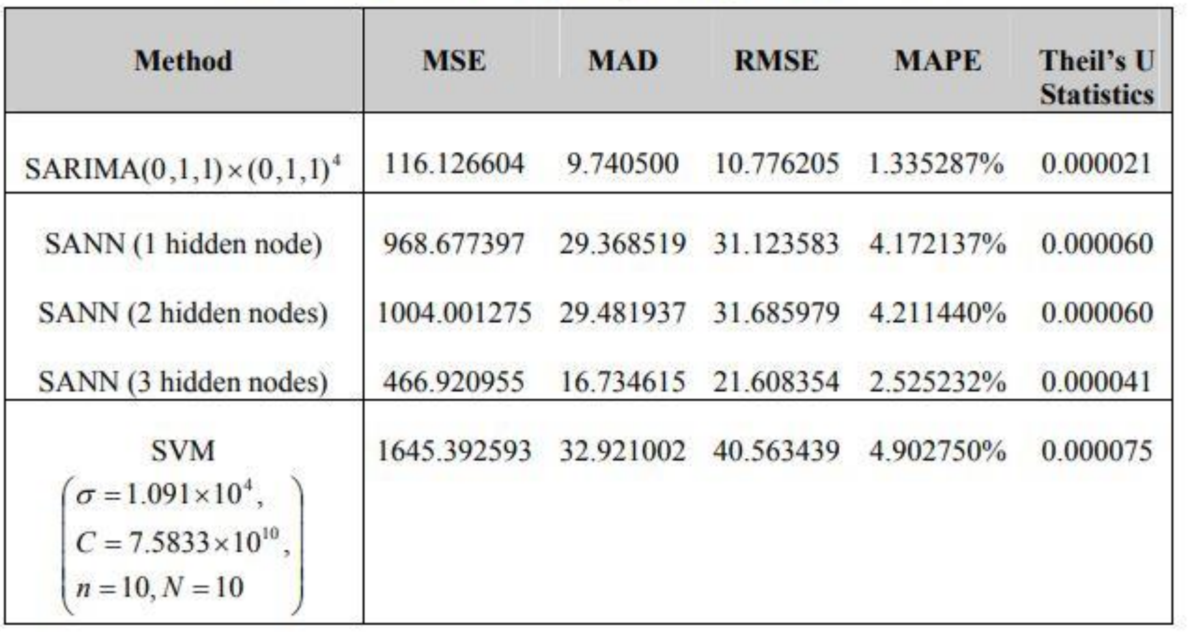
Fig. 5.5.1: Quarterly sales time series (for 6 years)

Fig. 7.5.1 represents the quarterly export data of a French firm for six years [3]. It is a non-stationary seasonal time series (with seasonal period 4) and was used by C. Hamzacebi [3] to assess the performance of SANN for quarterly data. This dataset has total 24 observations.

Following C. Hamzacebi [3] we have kept the first five years observations from the quarterly sales dataset for training and the sixth year observations for testing. We have fitted the SARIMA$(0,1,1) \times (0,1,1)^4$ model to the quarterly sales series after making natural logarithmic transformation and seasonal differencing. We have also fitted three SANN models (varying the number of hidden nodes to 1, 2 and 3) and the SVM model to this data. For fitting the SANN models we have used the rescaled quarterly sales data, obtained after diving by 100; for fitting the SVM model we have used the original data.

Our obtained forecast performance measures, calculated on original scale for the quarterly sales time series are presented in Table 7.5:

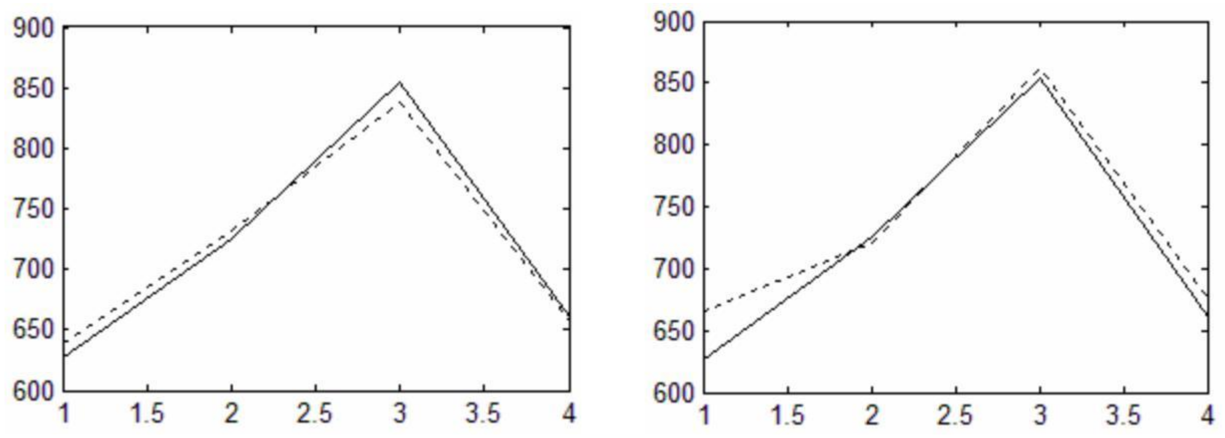
Table 7.5: Forecast results for quarterly sales time series



Method	MSE	MAD	RMSE	MAPE	Theil's U Statistics
$\text{SARIMA}(0,1,1)\times(0,1,1)^4$	116.126604	9.740500	10.776205	1.335287%	0.000021
SANN (1 hidden node)	968.677397	29.368519	31.123583	4.172137%	0.000060
SANN (2 hidden nodes)	1004.001275	29.481937	31.685979	4.211440%	0.000060
SANN (3 hidden nodes)	466.920955	16.734615	21.608354	2.525232%	0.000041
SVM $\begin{pmatrix}\sigma = 1.091\times 10^4, \\ C = 7.5833\times 10^{10}, \\ n = 10, N = 10\end{pmatrix}$	1645.392593	32.921002	40.563439	4.902750%	0.000075

From Table 7.5 we it can be seen that the best forecasts for the quarterly sales series are obtained by fitting SARIMA(0,1,1) × $(0,1,1)^4$ model. The forecasting performances of SANN models are moderate, while those of the SVM model are not up to the expectation. We can suggest that the forecasting performance of SVM may be improved by choosing some appropriate data transformation or rescaling for this time series.

In Fig. 5.5.2 we present the SARIMA and SANN (3 hidden nodes) forecast diagrams for the quarterly sales time series:



(a) SARIMA forecast diagram **(b) SANN forecast diagram**

Fig. 5.5.2: Forecast diagrams for quarterly sales series

From the above two forecast diagrams we can get a visual idea about the forecasting accuracy of the mentioned SARIMA and SANN models for the quarterly sales series.

5.6 The Quarterly U.S. Beer Production Dataset

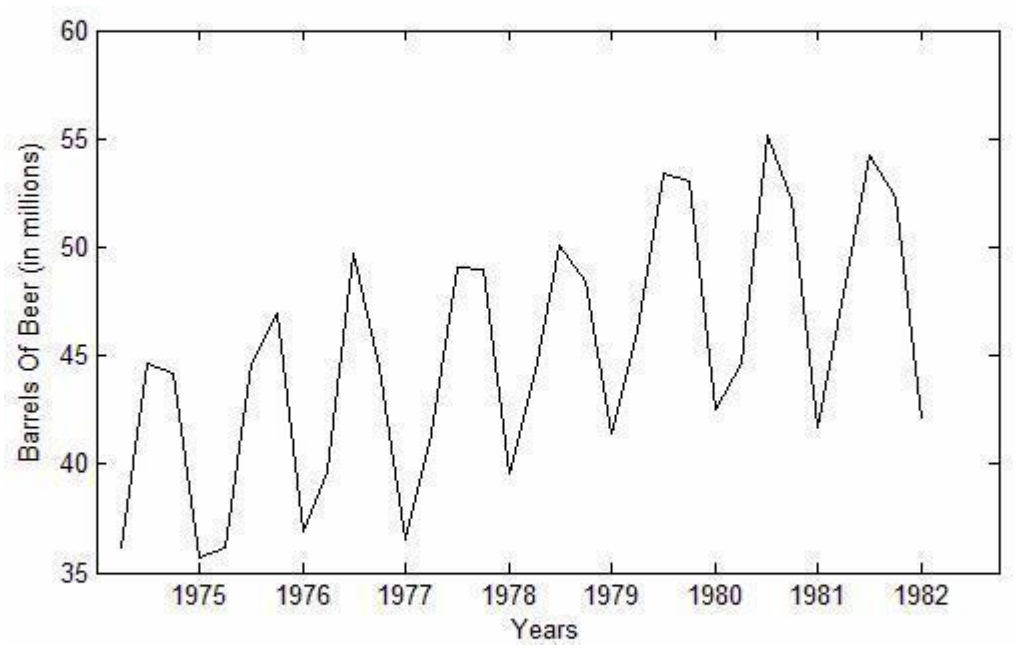
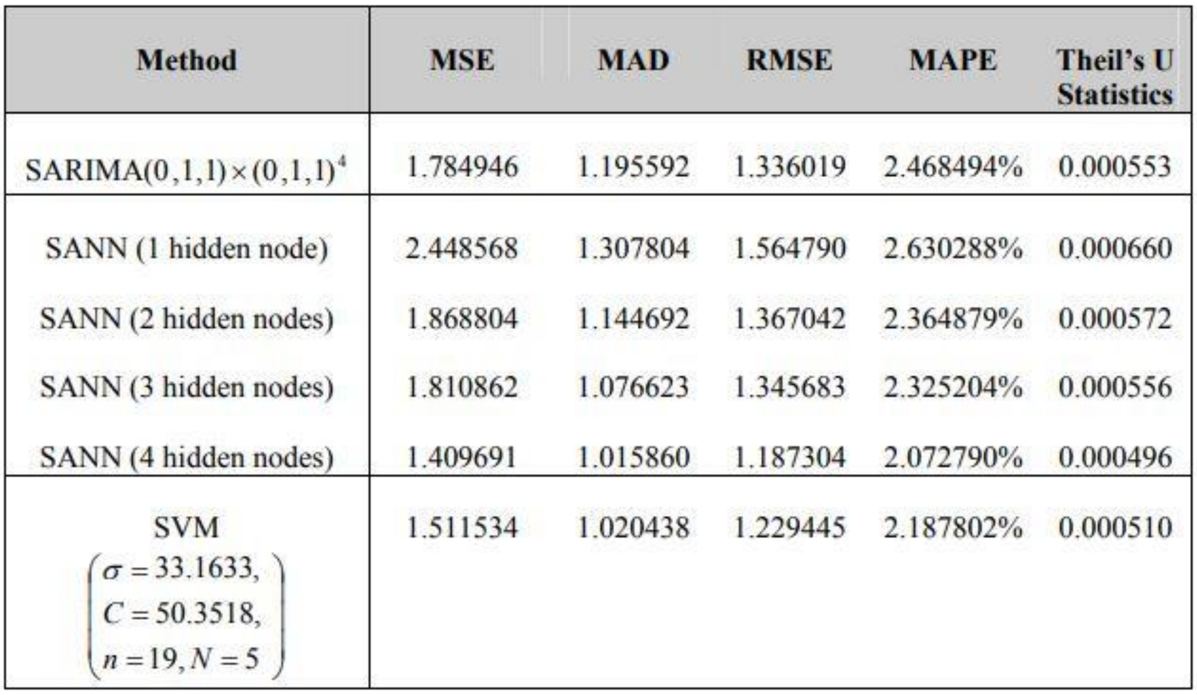


Fig. 7.6.1: Quarterly U.S. beer production time series (1975-1982)

Fig. 7.6.1 shows the quarterly U.S. beer production in millions of barrels from 1975 to 1982. This time series dataset is provided by William W. S. Wei; the associated website link is provided at the end of this ongoing book. Out of the total 32 observations in this time series, we have used the first 24 (i.e. 1975 to 1980) for training and the remaining 8 (i.e. 1981 to 1982) for testing. The forecasting models, we have fitted to this time series are SARIMA(0,1,1) $\times$ $(0,1,1)^4$, SANN (with hidden nodes 1, 2, 3 and 4) and SVM models. As usual for fitting the SARIMA model, we have performed natural logarithmic transformation and seasonal differencing of the data values. Also for fitting SANN models, we have divided the original observations by 10.

Our obtained forecast performance measures for the quarterly U.S. beer production time series are presented in original scale in Table 7.6:

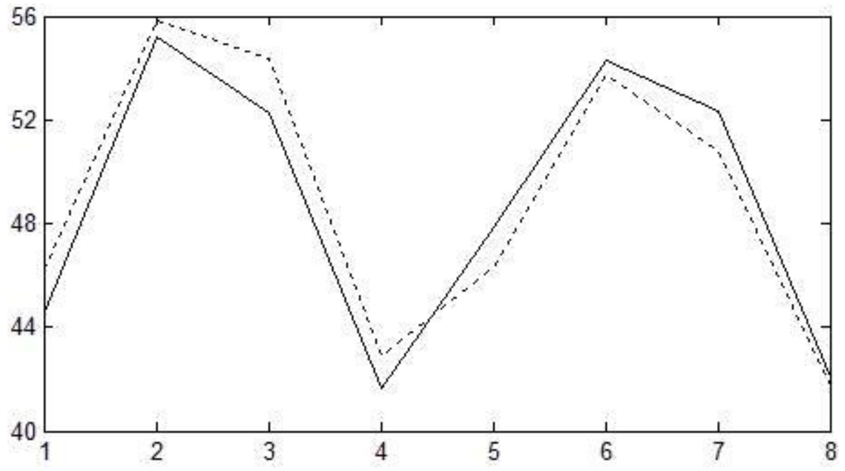
Table 7.6: Forecast results for quarterly U.S. beer production time series



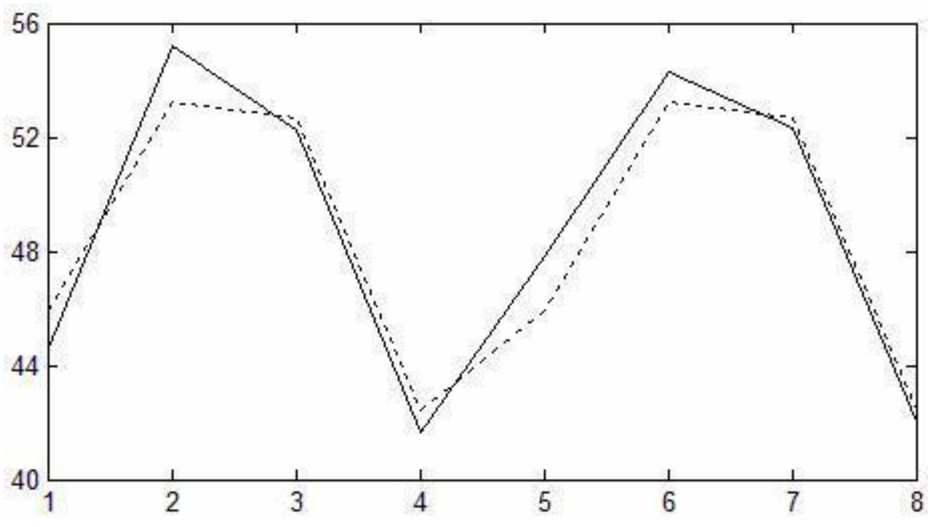
Method	MSE	MAD	RMSE	MAPE	Theil's U Statistics
$\text{SARIMA}(0,1,1)\times(0,1,1)^4$	1.784946	1.195592	1.336019	2.468494%	0.000553
SANN (1 hidden node)	2.448568	1.307804	1.564790	2.630288%	0.000660
SANN (2 hidden nodes)	1.868804	1.144692	1.367042	2.364879%	0.000572
SANN (3 hidden nodes)	1.810862	1.076623	1.345683	2.325204%	0.000556
SANN (4 hidden nodes)	1.409691	1.015860	1.187304	2.072790%	0.000496
SVM $\begin{pmatrix}\sigma = 33.1633, \\ C = 50.3518, \\ n = 19, N = 5\end{pmatrix}$	1.511534	1.020438	1.229445	2.187802%	0.000510

Table 7.6 shows that the relatively best forecast performance measures for quarterly U.S. beer production time series are obtained using SANN model with four hidden nodes. The performances of SARIMA and SVM models are also quite good, as can be seen from the table.

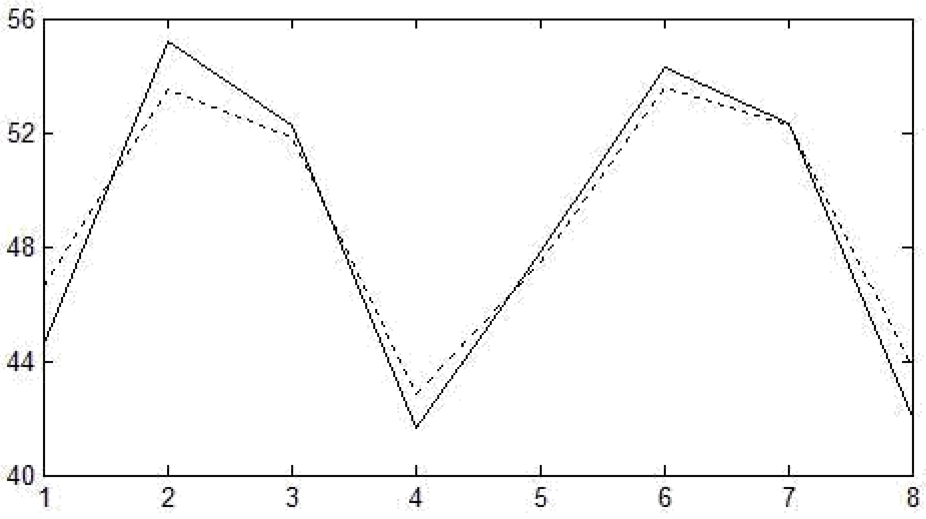
We present the three forecast diagrams for quarterly U.S. beer production time series, corresponding to SARIMA, SANN (4 hidden nodes) and SVM models in Fig. 7.6.2:



(a) SARIMA forecast diagram



(b) SANN forecast diagram



(c) SVM forecast diagram

Fig. 5.6.2: Forecast diagrams for quarterly U.S. beer production series

5.7 The Monthly USA Accidental Deaths Dataset

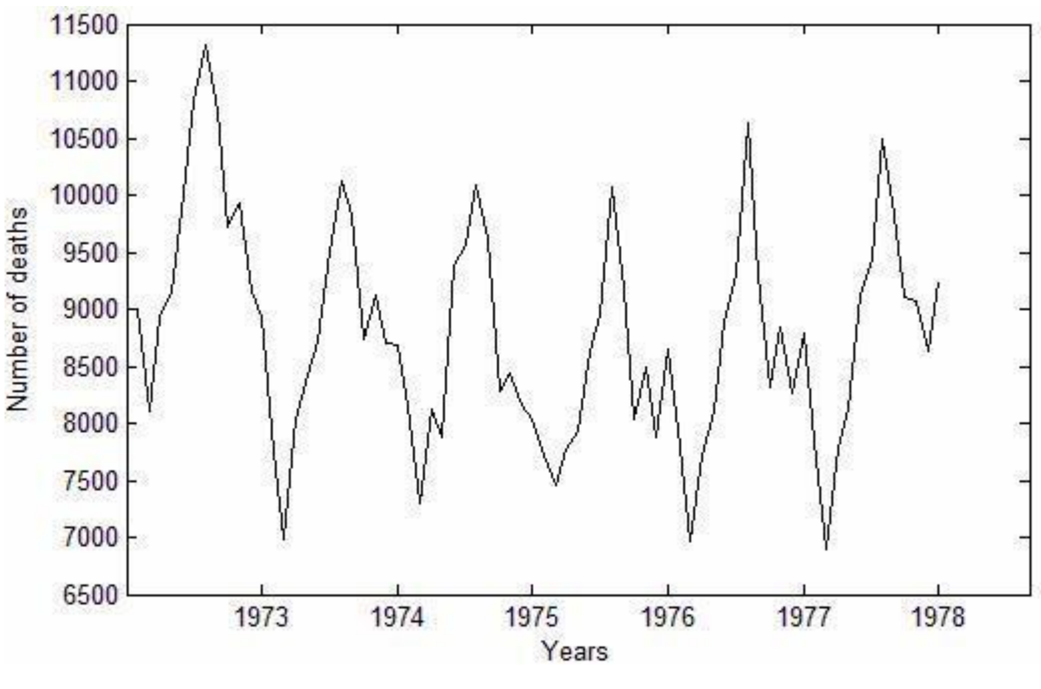


Fig. 7.7.1: Monthly USA accidental deaths time series (1973-1978)

The time series plot in Fig 7.7.1 represents the number of accidental deaths in USA from 1973 to 1978 on a monthly basis. This series is made available by Brockwell and Davis. We have given the associated website source at the end of this book. From the depicted Fig 7.7.1, it can be seen that the series shows seasonal fluctuations with somewhat a constant trend.

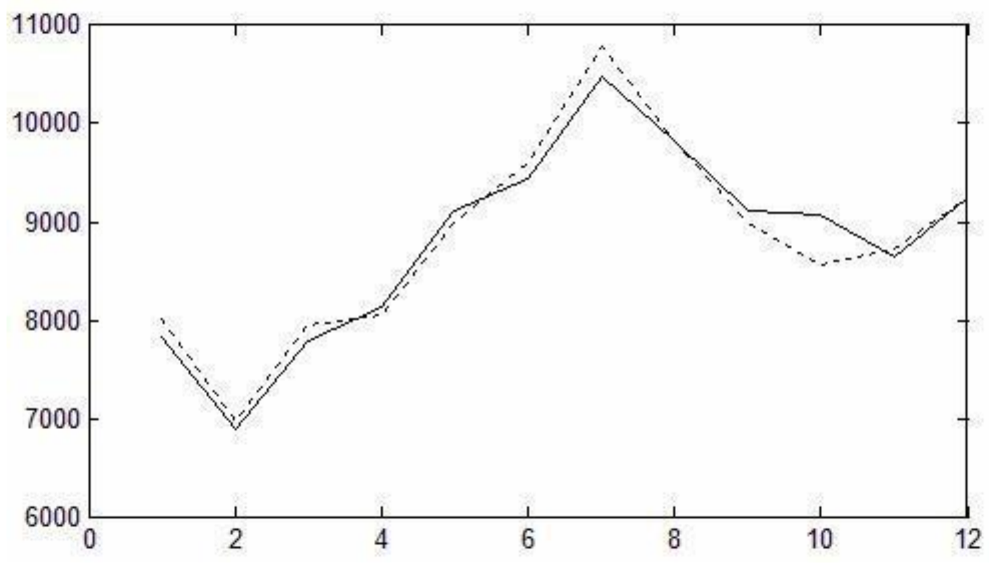
Out of the total 72 observations in the series, we have used the observations of first five years for training and those of the sixth year for testing. Thus the first 60 observations are considered for training and the remaining 12 for testing. Our fitted models for this time series are SARIMA(0,1,1) × $(0,1,1)^{12}$, SANN (with 4 hidden nodes) and SVM. As before for fitting SARIMA model we have applied logarithmic transformation and seasonal differencing. Also For fitting SANN and SVM models the observations are divided by 100. We present the obtained forecast performance measures in Table 7.7. These measures are calculated based on the rescaled dataset (i.e. after dividing by 100).

Table 5.7: Forecast results for monthly USA accidental deaths time series

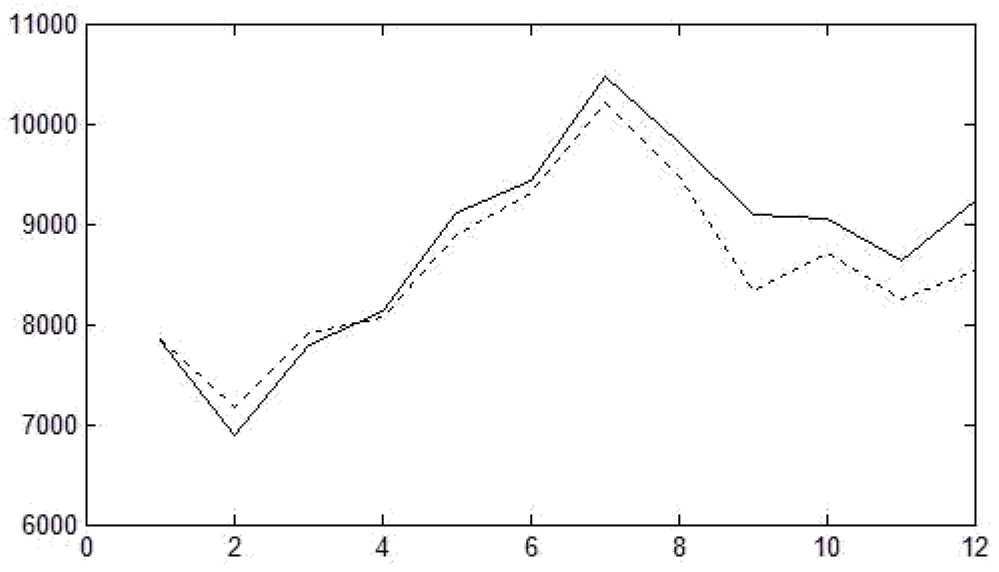
Method	**MSE**	**MAD**	**RMSE**	**MAPE**	**Theil's U Statistics**
SARIMA(0,1,1) × $(0,1,1)^{12}$	3.981148	1.499827	1.995281	1.694144%	0.000254
SANN (4 hidden nodes)	13.994408	2.986875	3.740910	3.344423%	0.000491
SVM $\sigma = 45.8047$, $C = 6.8738$, $n = 13$, $N = 47$	16.203377	3.328154	4.025342	3.708167%	0.000531

From Table 7.7 we see that the minimum performance measures are obtained by the fitted SARIMA(0,1,1) × $(0,1,1)^{12}$ model. Here to fit the SANN model 4 hidden nodes are considered, because our experiments have shown that with other number of hidden nodes the performance measures increased for this dataset. It can be suggested that the SVM performance for this dataset may be improved by performing some other suitable data preprocessing or using some other kernel function.

The two forecast diagrams for monthly USA accidental deaths time series, corresponding to the fitted SARIMA and SANN models are presented in Fig.7.7.2:



(a) SARIMA forecast diagram



(b) SANN forecast diagram

Fig. 5.7.2: Forecast diagrams for monthly USA accidental deaths series

The excellent performance of SARIMA for the monthly USA accidental deaths time series can be visualized from the forecast diagram in Fig. 5.7.2 (a).

We have presented the forecasting results of all the experiments done by us. From the performance measures obtained for each dataset, one can have a relative idea about the effectiveness and accuracy of the fitted models. The six time series datasets, we have considered are taken from non-confidential sources and each of them is freely available for analysis. At the end of the current book, we have listed the original online source for each dataset, together with the corresponding website link from which it is collected.

CHAPTER 6

PROJECT DESCRIPTION

6.1 BASIC CONCEPT OF TIME SERIES MODELLING

6.1.1 Definition of A Time Series

A time series is a sequential set of data points, measured typically over successive times. It is mathematically defined as a set of vectors $x(t), t = 0,1, 2,...$ where t represents the time elapsed [21, 23, 31]. The variable $x(t)$ is treated as a random variable. The measurements taken during an event in a time series are arranged in a proper chronological order.

A time series containing records of a single variable is termed as univariate. But if records of more than one variable are considered, it is termed as multivariate. A time series can be continuous or discrete. In a continuous time series observations are measured at every instance of time, whereas a discrete time series contains observations measured at discrete points of time. For example temperature readings, flow of a river, concentration of a chemical process etc. can be recorded as a continuous time series. On the other hand population of a particular city, production of a company, exchange rates between two different currencies may represent discrete time series. Usually in a discrete time series the consecutive observations are recorded at equally spaced time intervals such as hourly, daily, weekly, monthly or yearly time separations. As mentioned in [23], the variable being observed in a discrete time series is assumed to be measured as a continuous variable using the real number scale. Furthermore a continuous time series can be easily transformed to a discrete one by merging data together over a specified time interval.

6.1.2 Components of a Time Series

A time series in general is supposed to be affected by four main components, which can be separated from the observed data. These components are: *Trend*, *Cyclical*, *Seasonal* and *Irregular* components. A brief description of these four components is given here.

The general tendency of a time series to increase, decrease or stagnate over a long period of time is termed as Secular Trend or simply Trend. Thus, it can be said that trend is a long term movement in a time series. For example, series relating to population growth, number of houses in a city etc. show upward trend, whereas downward trend can be observed in series relating to mortality rates, epidemics, etc.

Seasonal variations in a time series are fluctuations within a year during the season. The important factors causing seasonal variations are: climate and weather conditions, customs, traditional habits, etc. For example sales of ice-cream increase in summer, sales of woolen cloths increase in winter. Seasonal variation is an important factor for businessmen, shopkeeper and producers for making proper future plans.

The cyclical variation in a time series describes the medium-term changes in the series, caused by circumstances, which repeat in cycles. The duration of a cycle extends over longer period of time, usually two or more years. Most of the economic and financial time series show some kind of cyclical variation. For example a business cycle consists of four phases, viz.

i) Prosperity, ii) Decline, iii) Depression and iv)
Recovery. Schematically a typical business cycle can be
shown as below:

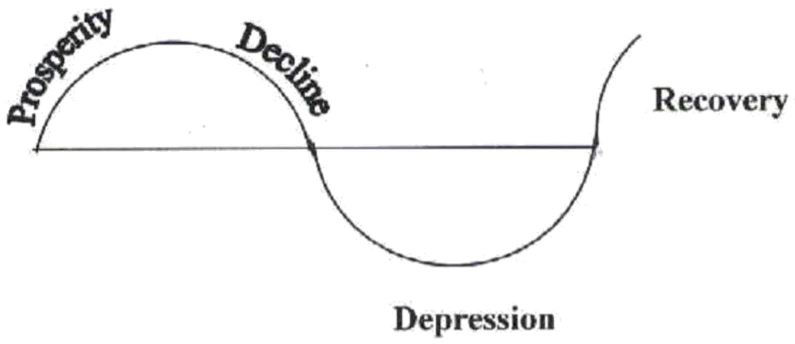


Fig. 6.1: A four phase business cycle

Irregular or random variations in a time series are caused by unpredictable influences, which are not regular and also do not repeat in a particular pattern. These variations are caused by incidences such as war, strike, earthquake, flood, revolution, etc. There is no defined statistical technique for measuring random fluctuations in a time series.

Considering the effects of these four components, two different types of models are generally used for a time series viz. Multiplicative and Additive models.

Multiplicative Model: $Y(t) = T(t) \times S(t) \times C(t) \times I(t)$.

Additive Model: $Y(t) = T(t) + S(t) + C(t) + I(t)$.

Here $Y(t)$ is the observation and $T(t)$, $S(t)$, $C(t)$ and $I(t)$ are respectively the trend, seasonal, cyclical and irregular variation at time t.

Multiplicative model is based on the assumption that the four components of a time series are not necessarily independent and they can affect one another; whereas in the additive model it is assumed that the four components are independent of each other.

6.1.3 Examples of Time Series Data

Time series observations are frequently encountered in many domains such as business, economics, industry, engineering and science, etc [7, 8, 10]. Depending on the nature of analysis and practical need, there can be various different kinds of time series. To visualize the basic pattern of the data, usually a time series is represented by a graph, where the observations are plotted against corresponding time. Below we show two time series plots:

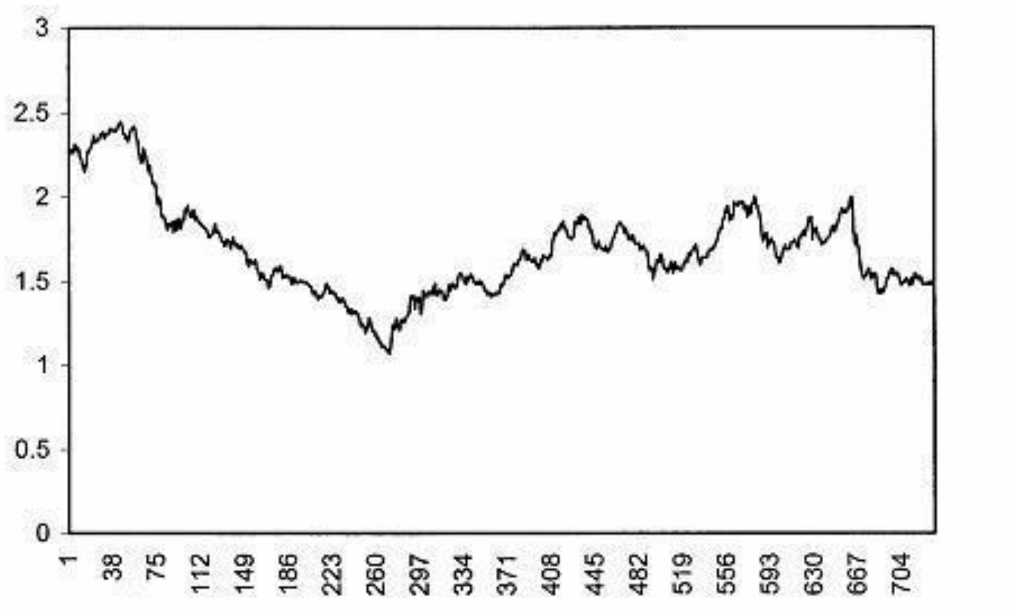


Fig. 6.2: Weekly BP/USD exchange rate series (1980-1993)

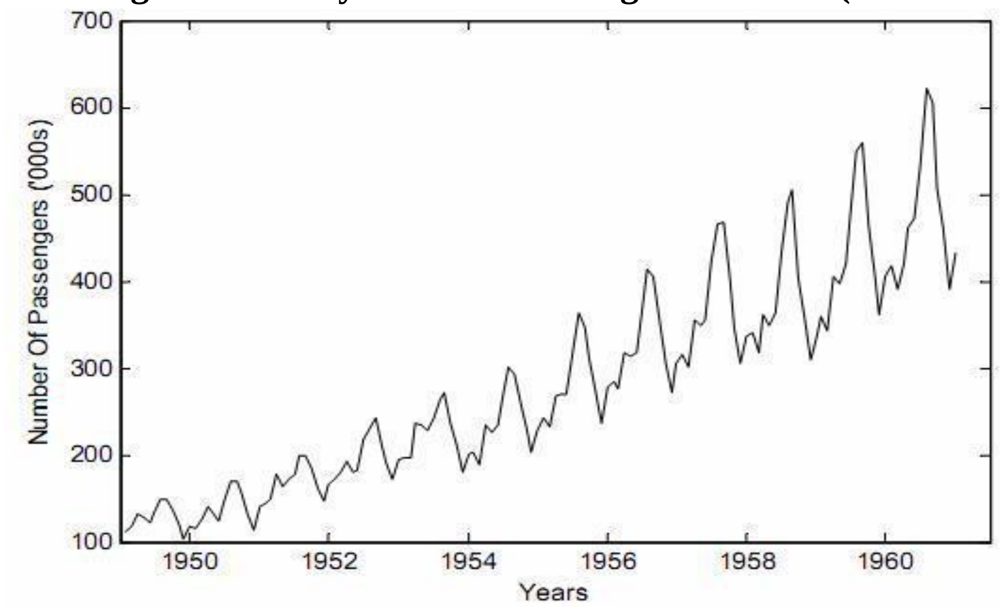


Fig. 6.3: Monthly international airline passenger series (Jan. 1949-Dec. 1960)

The first time series is taken from [8] and it represents the weekly exchange rate between British pound and US dollar from 1980 to 1933. The second one is a seasonal time series, considered in [3, 6, 11] and it shows the number of international airline passengers (in thousands) between Jan. 1949 to Dec. 1960 on a monthly basis.

6.1.4 Introduction to Time Series Analysis

In practice a suitable model is fitted to a given time series and the corresponding parameters are estimated using the known data values. The procedure of fitting a time series to a proper model is termed as *Time Series Analysis* [23]. It comprises methods that attempt to understand the nature of the series and is often useful for future forecasting and simulation.

In time series forecasting, past observations are collected and analyzed to develop a suitable mathematical model which captures the underlying data generating process for the series [7, 8]. The future events are then predicted using the model. This approach is particularly useful when there is not much knowledge about the statistical pattern followed by the successive observations or when there is a lack of a satisfactory explanatory model. Time series forecasting has important applications in various fields. Often valuable strategic decisions and precautionary measures are taken based on the forecast results. Thus making a good forecast, i.e. fitting an adequate model to a time series is vary important. Over the past several decades many efforts have been made by researchers for the development and improvement of suitable time series forecasting models.

6.1.5 Time Series and Stochastic Process

A time series is non-deterministic in nature, i.e. we cannot predict with certainty what will occur in future. Generally a time series $\{x(t), t = 0,1, 2,...\}$ is assumed to follow certain probability model [21] which describes the joint distribution of the random variable x_t . The mathematical expression describing the probability structure of a time series is termed as a stochastic process [23]. Thus the sequence of observations of the series is actually a sample realization of the stochastic process that produced it.

A usual assumption is that the time series variables x_t are independent and identically distributed (i.i.d) following the normal distribution. However as mentioned in [21], an interesting point is that time series are in fact not exactly i.i.d; they follow more or less some regular pattern in long term. For example if the temperature today of a particular city is extremely high, then it can be reasonably presumed that tomorrow's temperature will also likely to be high. This is the reason why time series forecasting using a proper technique, yields result close to the actual value.

6.1.6 Concept of Stationarity

The concept of stationarity of a stochastic process can be visualized as a form of statistical equilibrium [23]. The statistical properties such as mean and variance of a stationary process do not depend upon time. It is a necessary condition for building a time series model that is useful for future forecasting. Further, the mathematical complexity of the fitted model reduces with this assumption. There are two types of stationary processes which are defined below:

A process $\{x(t), t = 0,1,2,...\}$ is *Strongly Stationary* or *Strictly Stationary* if the joint probability distribution function of $\{x_{t-s}, x_{t-s+1}, ..., x_t, ...x_{t+s-1}, x_{t+s}\}$ is independent of t for all s. Thus for a strong stationary process the joint distribution of any possible set of random variables from the process is independent of time [21, 23].

However for practical applications, the assumption of strong stationarity is not always needed and so a somewhat weaker form is considered. A stochastic process is said to be *Weakly Stationary* of order k if the statistical moments of the process up to that order depend only on time differences and not upon the time of occurrences of the data being used to estimate the moments [12, 21, 23]. For example a stochastic process $\{x(t), t = 0,1,2,...\}$ is second order stationary [12, 23] if it has time independent mean and variance and the covariance values $Cov(x_t, x_{t-s})$ depend only on s.

It is important to note that neither strong nor weak stationarity implies the other. However, a weakly stationary process following normal distribution is also strongly stationary [21]. Some mathematical tests like the one given by Dickey and Fuller [21] are generally used to detect stationarity in a time series data.

As mentioned in [6, 23], the concept of stationarity is a mathematical idea constructed to simplify the theoretical and practical development of stochastic processes. To design a proper model, adequate for future forecasting, the underlying time series is expected to be stationary. Unfortunately it is not always the case.

As stated by Hipel and McLeod [23], the greater the time span of historical observations, the greater is the chance that the time series will exhibit non-stationary characteristics. However for relatively short time span, one can reasonably model the series using a stationary stochastic process. Usually time series, showing trend or seasonal patterns are non-stationary in nature. In such cases, differencing and power transformations are often used to remove the trend and to make the series stationary. In the next chapter we shall discuss about the seasonal differencing technique applied to make a seasonal time series stationary.

6.1.7 Model Parsimony

While building a proper time series model we have to consider the principle of parsimony [2, 7, 8, 23]. According to this principle, always the model with smallest possible number of parameters is to be selected so as to provide an adequate representation of the underlying time series data [2]. Out of a number of suitable models, one should consider the simplest one, still maintaining an accurate description of inherent properties of the time series. The idea of model parsimony is similar to the famous *Occam's razor* principle [23]. As discussed by Hipel and McLeod [23], one aspect of this principle is that when face with a number of competing and adequate explanations, pick the most simple one.

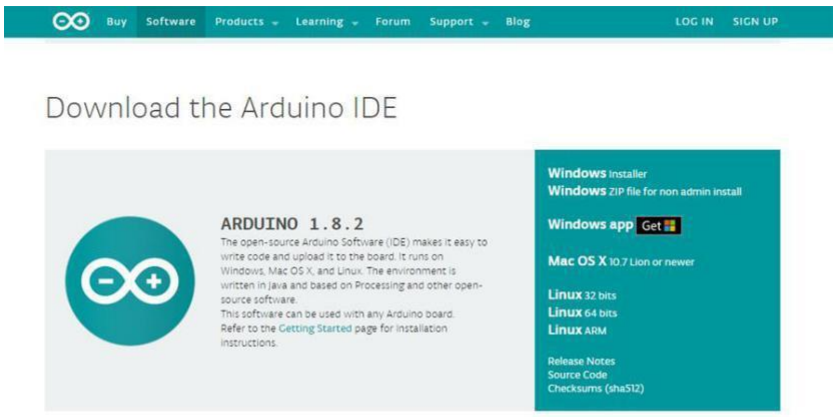
The Occam's razor provides considerable inherent informations, when applied to logical analysis.Moreover, the more complicated the model, the more possibilities will arise for departure from the actual model assumptions. With the increase of model parameters, the risk of overfitting also subsequently increases. An over fitted time series model may describe the training data very well, but it may not be suitable for future forecasting. As potential overfitting affects the ability of a model to forecast well, parsimony is often used as a guiding principle to overcome this issue. Thus in summary it can be said that, while making time series forecasts, genuine attention should be given to select the most parsimonious model among all other possibilities.

CHAPTER 7
INSTALLATION

7.1 MATLAB INSTALLATION

Arduino is open source software required for the purpose of uploading the code into Arduino. This can be downloaded and installed in such a way that it supports most of the operating system efficiently.

STEP 1: Search for Arduino recent IDE version on www.arduino.cc and you will find a screen as shown in figure 7.1.



Download the Arduino IDE

ARDUINO 1.8.2

The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software.

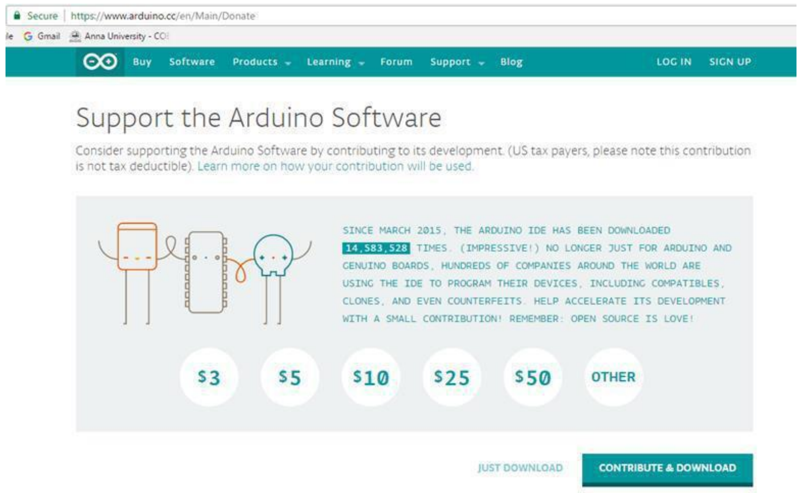
This software can be used with any Arduino board.

Refer to the Getting Started page for installation instructions.

Windows Installer
Windows ZIP file for non admin install
Windows app Get
Mac OS X 10.7 Lion or newer
Linux 32 bits
Linux 64 bits
Linux ARM
Release Notes
Source Code
Checksums (sha512)

Figure 7.1 Arduino IDE Search compatible for OS

STEP 2: Download Arduino IDE appropriate for the Operating System. Figure 7.2 shows the download screen page of Arduino IDE for Windows OS.



Secure | https://www.arduino.cc/en/Main/Donate

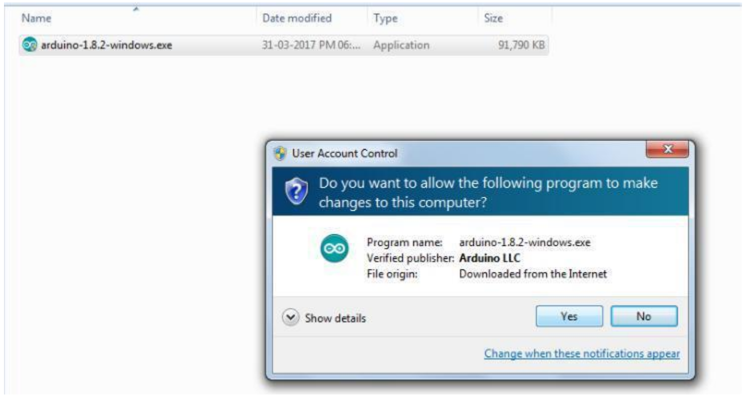
Support the Arduino Software

Consider supporting the Arduino Software by contributing to its development. (US tax payers, please note this contribution is not tax deductible). Learn more on how your contribution will be used.

SINCE MARCH 2015, THE ARDUINO IDE HAS BEEN DOWNLOADED 14,583,528 TIMES. (IMPRESSIVE!) NO LONGER JUST FOR ARDUINO AND GENUINO BOARDS, HUNDREDS OF COMPANIES AROUND THE WORLD ARE USING THE IDE TO PROGRAM THEIR DEVICES, INCLUDING COMPATIBLES, CLONES, AND EVEN COUNTERFEITS. HELP ACCELERATE ITS DEVELOPMENT WITH A SMALL CONTRIBUTION! REMEMBER: OPEN SOURCE IS LOVE!

Figure 7.2 Arduino IDE download page for Windows OS

STEP 3: Download the file on desired location and start running the installation procedure as shown in figure 7.3.



Name	Date modified	Type	Size
arduino-1.8.2-windows.exe	31-03-2017 PM 06:...	Application	91,790 KB

User Account Control

Do you want to allow the following program to make changes to this computer?

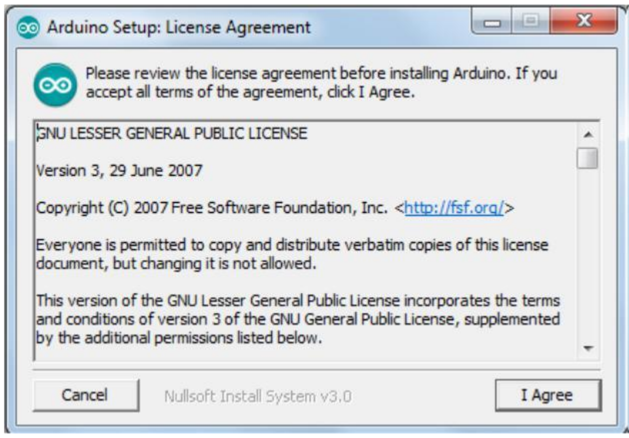
Program name: arduino-1.8.2-windows.exe
Verified publisher: Arduino LLC
File origin: Downloaded from the Internet

Show details Yes No

Change when these notifications appear

Figure 7.3 Installing Arduino IDE

STEP 4: Accept the License Agreement and select the components that you want to install as shown in Figure 7.4 and Figure 7.5 and also browse the path to install as shown in Figure 7.6.



Arduino Setup: License Agreement

Please review the license agreement before installing Arduino. If you accept all terms of the agreement, click I Agree.

GNU LESSER GENERAL PUBLIC LICENSE

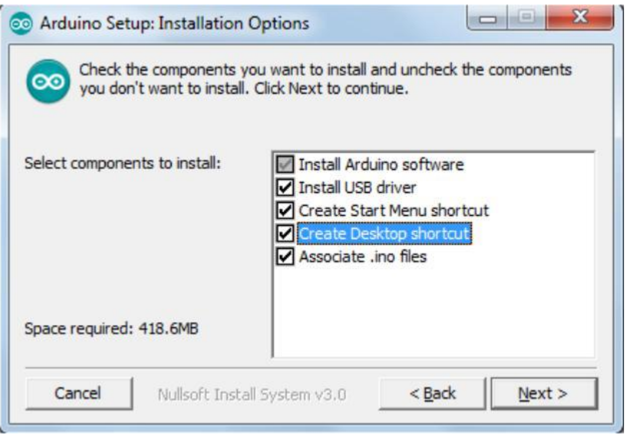
Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

Cancel Nullsoft Install System v3.0 I Agree



Arduino Setup: Installation Options

Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

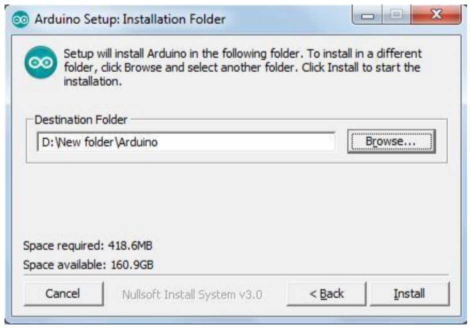
Select components to install:

- ✓ Install Arduino software
- ✓ Install USB driver
- ✓ Create Start Menu shortcut
- ✓ Create Desktop shortcut
- ✓ Associate .ino files

Space required: 418.6MB

Cancel Nullsoft Install System v3.0 < Back Next >

Figure 7.4 License Agreement Figure 7.5 Selecting components



Arduino Setup: Installation Folder

Setup will install Arduino in the following folder. To install in a different folder, click Browse and select another folder. Click Install to start the installation.

Destination Folder

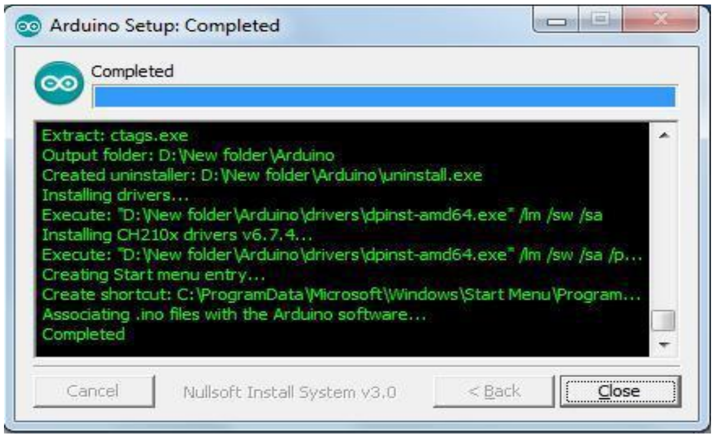
D:\New folder\Arduino Browse...

Space required: 418.6MB
Space available: 160.9GB

Cancel Nullsoft Install System v3.0 < Back Install

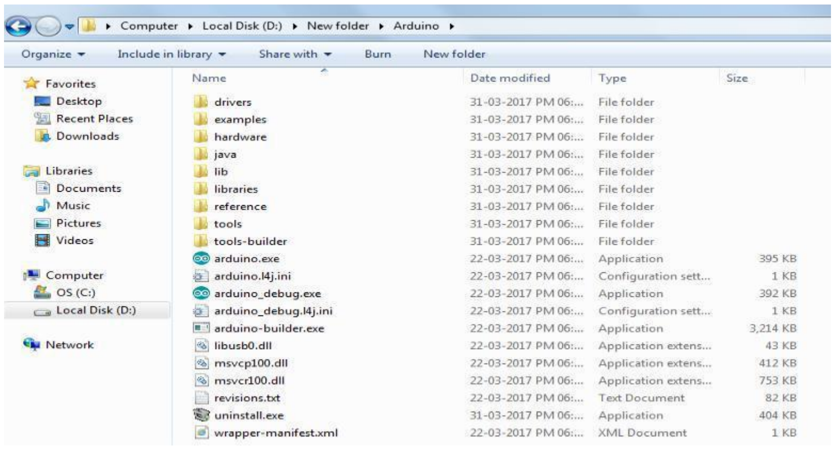
Figure 7.6 Browsing desired location to install Arduino IDE

STEP 5: Once the installation gets completed, check the folder for files as shown in Figure 7.7 and Figure 7.8.



```
Extract: ctags.exe
Output folder: D:\New folder\Arduino
Created uninstaller: D:\New folder\Arduino\uninstall.exe
Installing drivers...
Execute: "D:\New folder\Arduino\drivers\dpinst-amd64.exe" /lm /sw /sa
Installing CH210x drivers v6.7.4...
Execute: "D:\New folder\Arduino\drivers\dpinst-amd64.exe" /lm /sw /sa /p...
Creating Start menu entry...
Create shortcut: C:\ProgramData\Microsoft\Windows\Start Menu\Program...
Associating .ino files with the Arduino software...
Completed
```

Figure 7.7 Installation complete



Name	Date modified	Type	Size
drivers	31-03-2017 PM 06:...	File folder	
examples	31-03-2017 PM 06:...	File folder	
hardware	31-03-2017 PM 06:...	File folder	
java	31-03-2017 PM 06:...	File folder	
lib	31-03-2017 PM 06:...	File folder	
libraries	31-03-2017 PM 06:...	File folder	
reference	31-03-2017 PM 06:...	File folder	
tools	31-03-2017 PM 06:...	File folder	
tools-builder	31-03-2017 PM 06:...	File folder	
arduino.exe	22-03-2017 PM 06:...	Application	395 KB
arduino.l4j.ini	22-03-2017 PM 06:...	Configuration sett...	1 KB
arduino_debug.exe	22-03-2017 PM 06:...	Application	392 KB
arduino_debug.l4j.ini	22-03-2017 PM 06:...	Configuration sett...	1 KB
arduino-builder.exe	22-03-2017 PM 06:...	Application	3,214 KB
libusb0.dll	22-03-2017 PM 06:...	Application extens...	43 KB
msvcp100.dll	22-03-2017 PM 06:...	Application extens...	412 KB
msvcr100.dll	22-03-2017 PM 06:...	Application extens...	753 KB
revisions.txt	22-03-2017 PM 06:...	Text Document	82 KB
uninstall.exe	31-03-2017 PM 06:...	Application	404 KB
wrapper-manifest.xml	22-03-2017 PM 06:...	XML Document	1 KB

Figure 7.8 Arduino folder with files

STEP 6: Openthe file **32rduino.exe,** you will find as shown in Figure 7.9

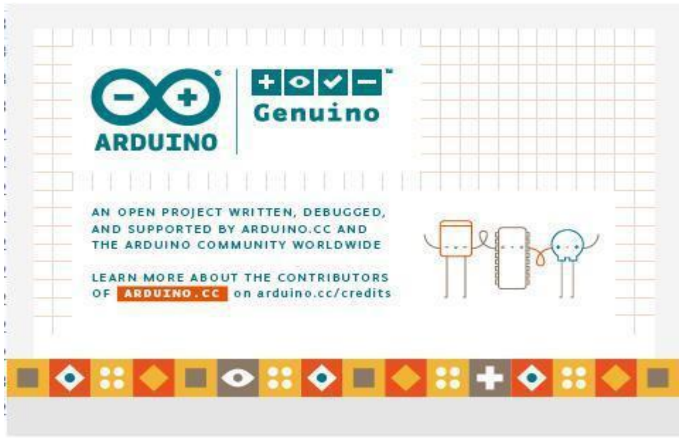
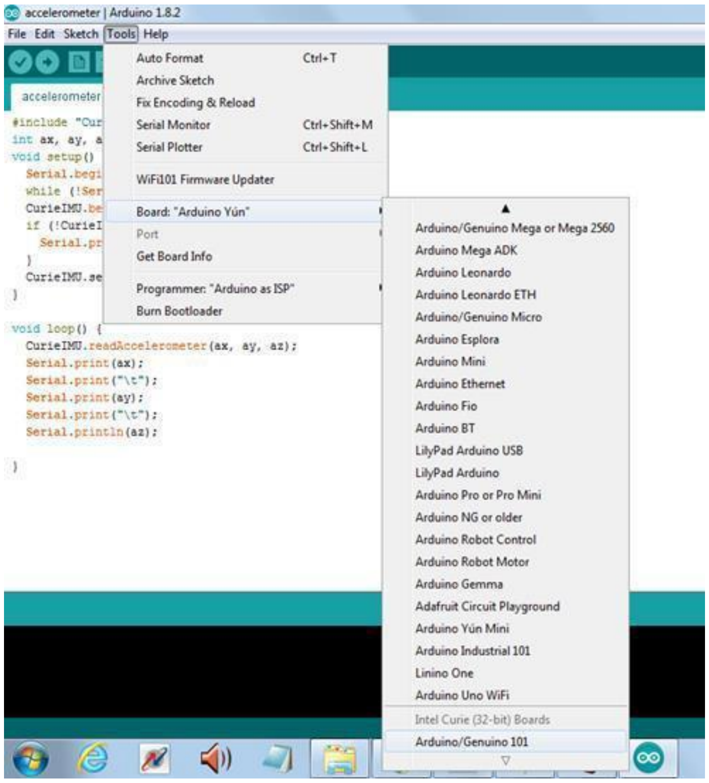

ARDUINO
Genuino
AN OPEN PROJECT WRITTEN, DEBUGGED,
AND SUPPORTED BY ARDUINO.CC AND
THE ARDUINO COMMUNITY WORLDWIDE
LEARN MORE ABOUT THE CONTRIBUTORS
OF ARDUINO.CC on arduino.cc/credits

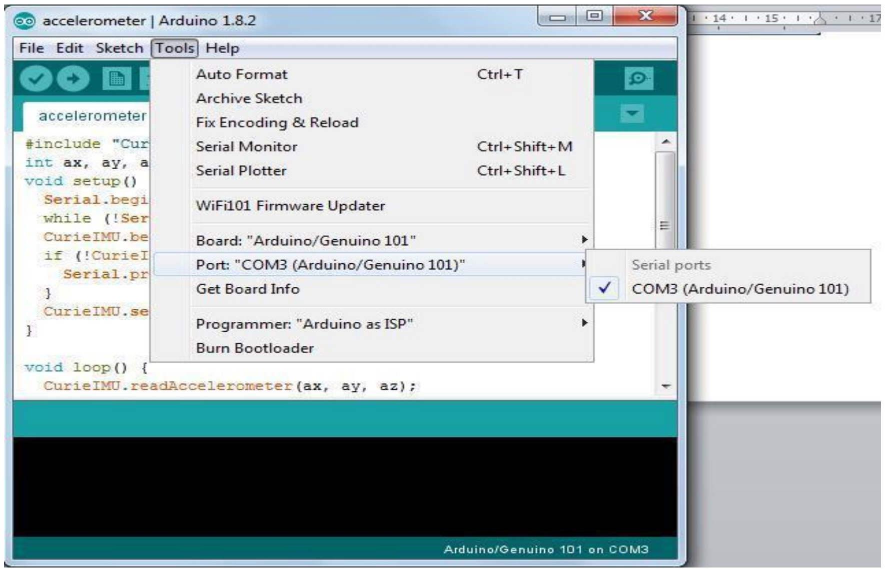
Figure 7.9 Opening Arduino IDE

STEP 7: Select board as ‘Arduino/Genuino 101’ as shown in 7.10 and connect the board via USB port and select the serial port number in which the board is connected as shown in Figure 7.11.



```
#include "Cur
int ax, ay, a
void setup()
  Serial.begi
  while (!Ser
  CurieIMU.be
  if (!CurieI
    Serial.pr
  }
  CurieIMU.se
}

void loop() {
  CurieIMU.readAccelerometer(ax, ay, az);
  Serial.print(ax);
  Serial.print("\t");
  Serial.print(ay);
  Serial.print("\t");
  Serial.println(az);

}
```

Figure 7.10 Selecting board name

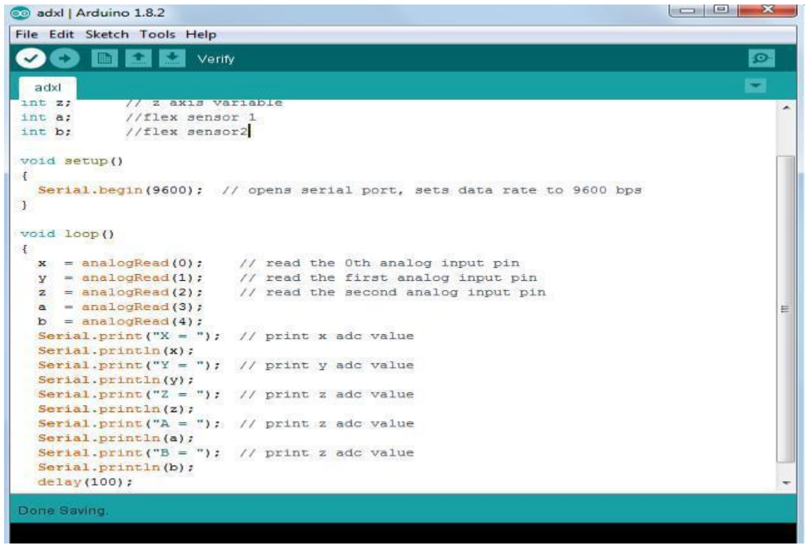

accelerometer | Arduino 1.8.2
File Edit Sketch Tools Help
accelerometer
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 Firmware Updater
Board: "Arduino/Genuino 101"
Port: "COM3 (Arduino/Genuino 101)"
Get Board Info
Programmer: "Arduino as ISP"
Burn Bootloader
Serial ports
COM3 (Arduino/Genuino 101)

```
#include "Cur
int ax, ay, a
void setup()
  Serial.begi
  while (!Ser
  CurieIMU.be
  if (!CurieI
    Serial.pr
  }
  CurieIMU.se
}
void loop() {
  CurieIMU.readAccelerometer(ax, ay, az);
```

```
Arduino/Genuino 101 on COM3
```

Figure 7.11 connecting board and selecting the serial port

STEP 8: Open a new file and code a program to read adxl and flex sensor values via serial port as analog read as shown in Figure 7.12.

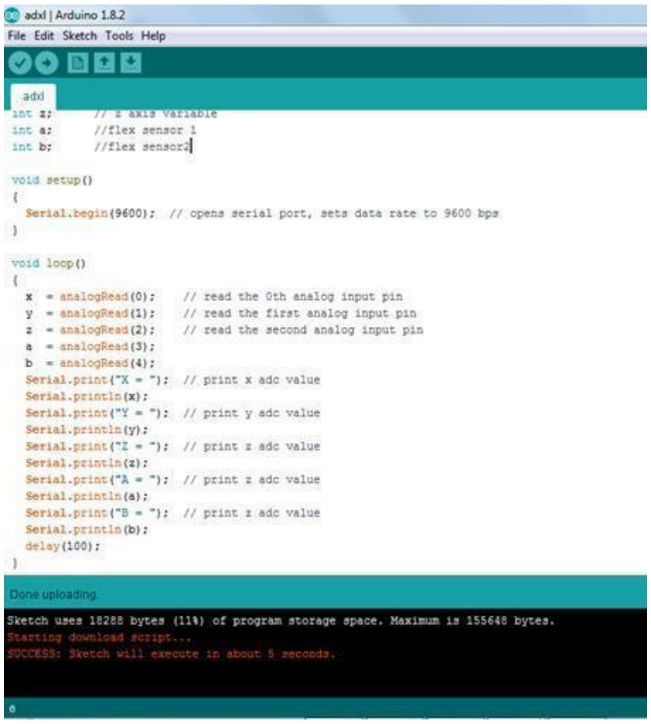


```
int z;       // z axis variable
int a;       //flex sensor 1
int b;       //flex sensor2

void setup()
{
  Serial.begin(9600);  // opens serial port, sets data rate to 9600 bps
}

void loop()
{
  x  = analogRead(0);    // read the 0th analog input pin
  y  = analogRead(1);    // read the first analog input pin
  z  = analogRead(2);    // read the second analog input pin
  a  = analogRead(3);
  b  = analogRead(4);
  Serial.print("X = ");  // print x adc value
  Serial.println(x);
  Serial.print("Y = ");  // print y adc value
  Serial.println(y);
  Serial.print("Z = ");  // print z adc value
  Serial.println(z);
  Serial.print("A = ");  // print z adc value
  Serial.println(a);
  Serial.print("B = ");  // print z adc value
  Serial.println(b);
  delay(100);
```

```
Done Saving.
```

Figure 7.12 Code for reading adxl and flex sensor values

STEP 9: Verify and upload the program on to the board as shown in Figure 7.13.

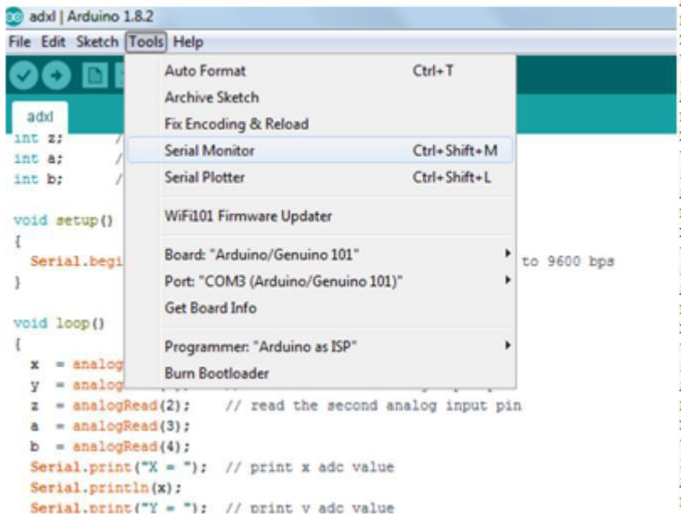


```
int z;       // z axis variable
int a;       //flex sensor 1
int b;       //flex sensor2

void setup()
{
  Serial.begin(9600);  // opens serial port, sets data rate to 9600 bps
}

void loop()
{
  x  = analogRead(0);    // read the 0th analog input pin
  y  = analogRead(1);    // read the first analog input pin
  z  = analogRead(2);    // read the second analog input pin
  a  = analogRead(3);
  b  = analogRead(4);
  Serial.print("X = ");  // print x adc value
  Serial.println(x);
  Serial.print("Y = ");  // print y adc value
  Serial.println(y);
  Serial.print("Z = ");  // print z adc value
  Serial.println(z);
  Serial.print("A = ");  // print z adc value
  Serial.println(a);
  Serial.print("B = ");  // print z adc value
  Serial.println(b);
  delay(100);
}
```

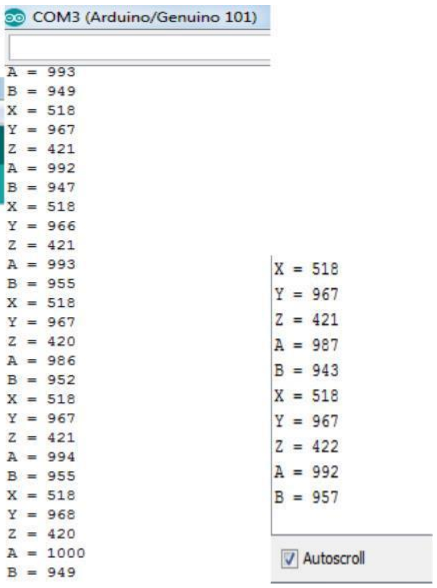
```
Done uploading.
Sketch uses 18288 bytes (11%) of program storage space. Maximum is 155648 bytes.
Starting download script...
SUCCESS: Sketch will execute in about 5 seconds.
```

Figure 7.13 Uploading the code on to the board

STEP 10: Open Serial Monitor to note the values as shown in Figure 7.14 and the values will be like shown in Figure 7.15.



```
int z;      /
int a;      /
int b;      /

void setup()
{
  Serial.begi                                     to 9600 bps
}

void loop()
{
  x  = analog
  y  = analog
  z  = analogRead(2);    // read the second analog input pin
  a  = analogRead(3);
  b  = analogRead(4);
  Serial.print("X = ");  // print x adc value
  Serial.println(x);
  Serial.print("Y = ");  // print y adc value
```



```
A = 993
B = 949
X = 518
Y = 967
Z = 421
A = 992
B = 947
X = 518
Y = 966
Z = 421
A = 993
B = 955
X = 518
Y = 967
Z = 420
A = 986
B = 952
X = 518
Y = 967
Z = 421
A = 994
B = 955
X = 518
Y = 968
Z = 420
A = 1000
B = 949
```

```
X = 518
Y = 967
Z = 421
A = 987
B = 943
X = 518
Y = 967
Z = 422
A = 992
B = 957
```

Figure 7.14 Opening Serial Monitor Figure 7.15 Adxl values

7.2 MATLAB INSTALLATION

STEPS: Installation of MATLAB Software for window XP/VISTA / Windows 7/8

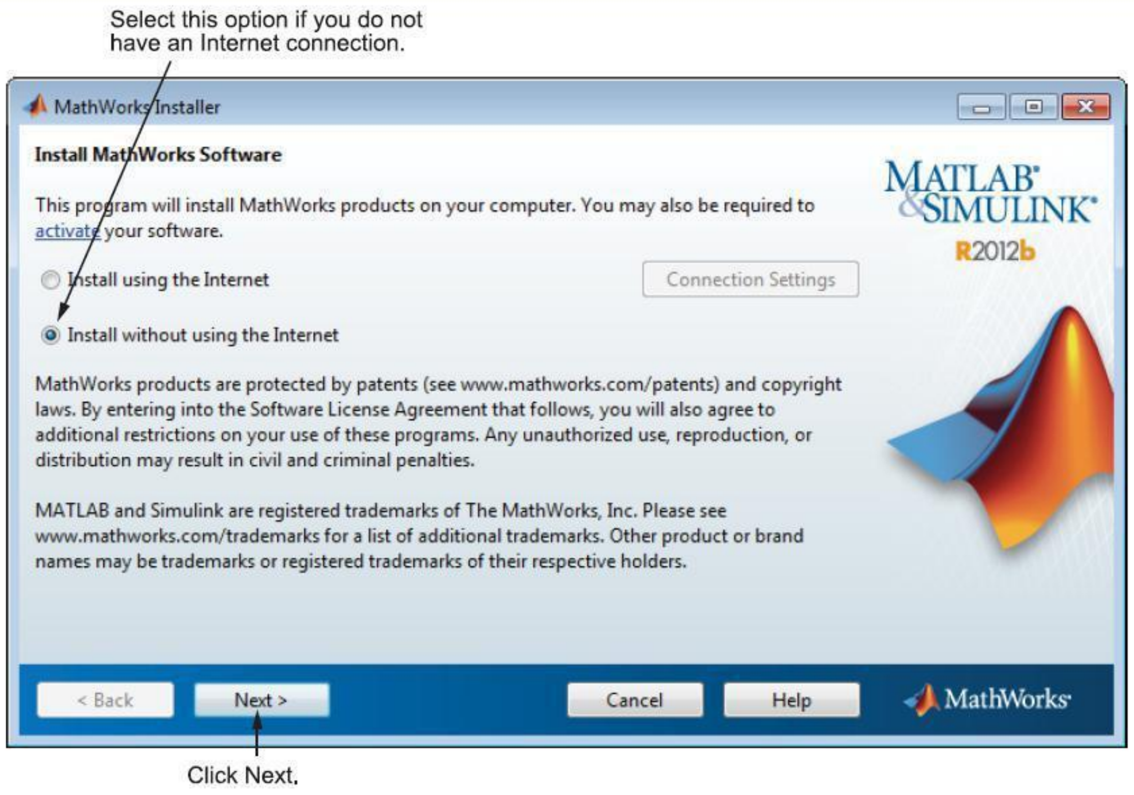
Get administrator privileges for the system on which you plan to
install MATLAB. Use WinRAR to extract RAR file

Step1: Start the installer

For Windows, double-clicking on setup.exe

Step 2: Choose to Install Without Using the Internet

When it starts, the installer displays the following dialog box. Select the Install without using the Internet option and Click OK to proceed with installation.



Install MathWorks Software

This program will install MathWorks products on your computer. You may also be required to activate your software.

Install using the Internet

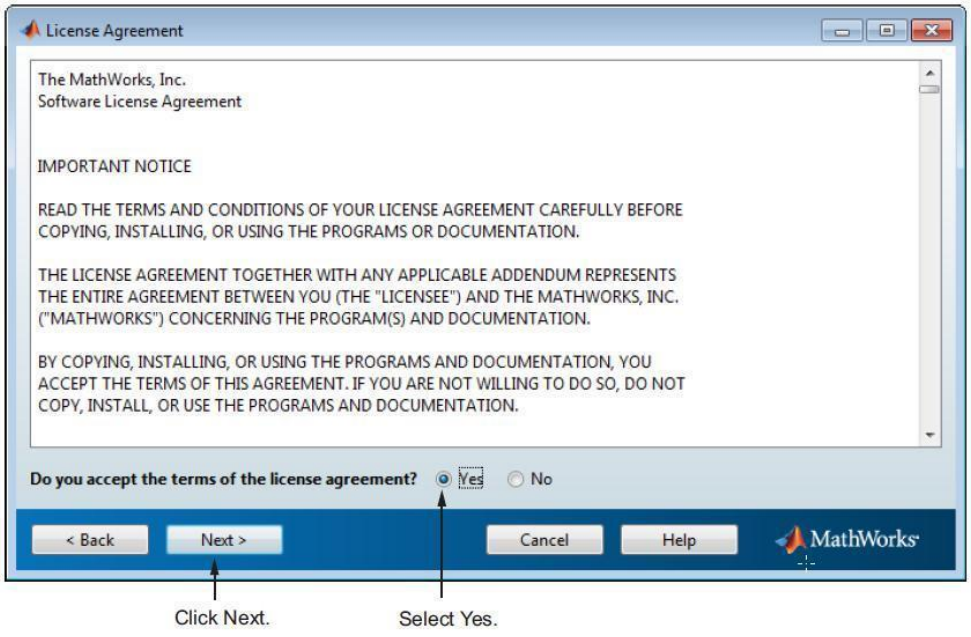
Install without using the Internet

MathWorks products are protected by patents (see www.mathworks.com/patents) and copyright laws. By entering into the Software License Agreement that follows, you will also agree to additional restrictions on your use of these programs. Any unauthorized use, reproduction, or distribution may result in civil and criminal penalties.

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. Please see www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Step 3: Review the License Agreement

Review the software licensing agreement and, if you agree to its terms, click Yes.



License Agreement

The MathWorks, Inc.
Software License Agreement

IMPORTANT NOTICE

READ THE TERMS AND CONDITIONS OF YOUR LICENSE AGREEMENT CAREFULLY BEFORE COPYING, INSTALLING, OR USING THE PROGRAMS OR DOCUMENTATION.

THE LICENSE AGREEMENT TOGETHER WITH ANY APPLICABLE ADDENDUM REPRESENTS THE ENTIRE AGREEMENT BETWEEN YOU (THE "LICENSEE") AND THE MATHWORKS, INC. ("MATHWORKS") CONCERNING THE PROGRAM(S) AND DOCUMENTATION.

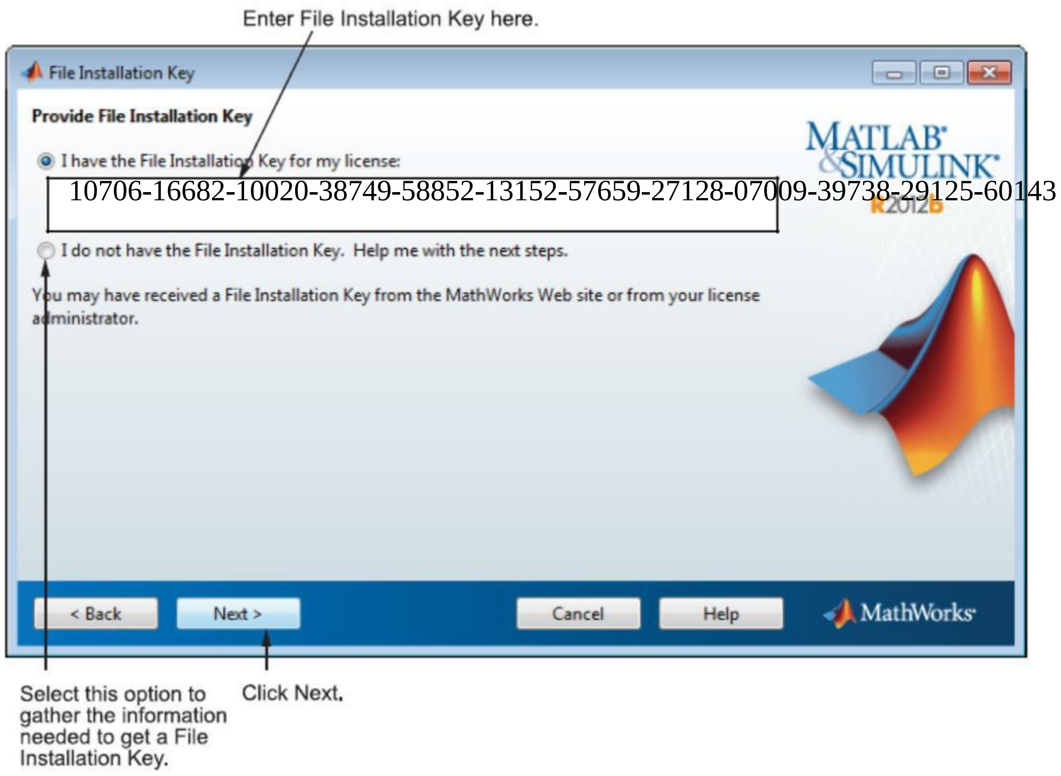
BY COPYING, INSTALLING, OR USING THE PROGRAMS AND DOCUMENTATION, YOU ACCEPT THE TERMS OF THIS AGREEMENT. IF YOU ARE NOT WILLING TO DO SO, DO NOT COPY, INSTALL, OR USE THE PROGRAMS AND DOCUMENTATION.

Do you accept the terms of the license agreement? [✓ Yes] No

Step 4: Enter the File Installation Key

Enter your File Installation Key and Click OK.

10706-16682-10020-38749-58852-13152-57659-27128-07009-39738-29125-60143



File Installation Key

Provide File Installation Key

[✓ I have the File Installation Key for my license:]

10706-16682-10020-38749-58852-13152-57659-27128-07009-39738-29125-60143

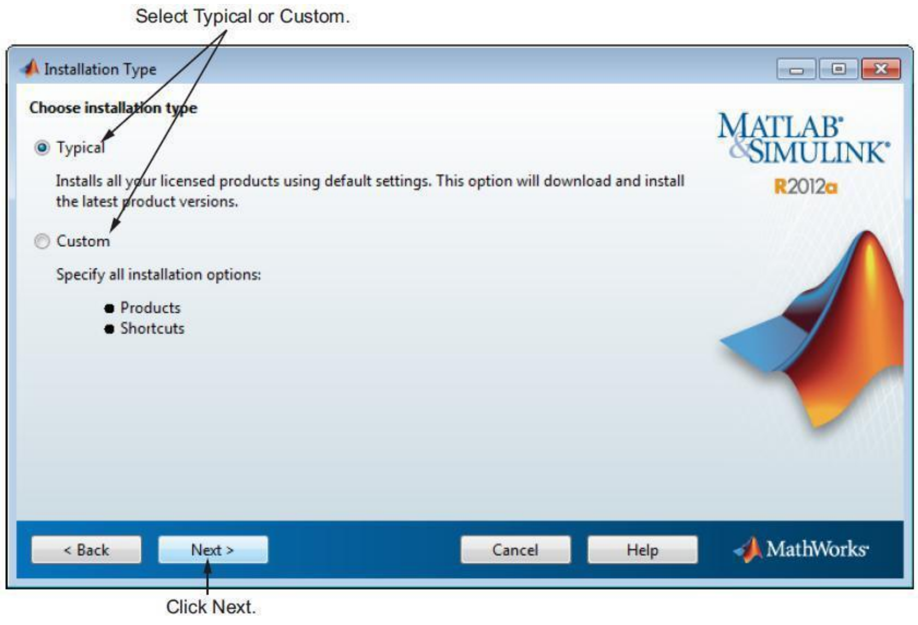
I do not have the File Installation Key. Help me with the next steps.

You may have received a File Installation Key from the MathWorks Web site or from your license administrator.

Enter File Installation Key here.
MATLAB & SIMULINK R2012b
< Back
Next >
Cancel
Help
MathWorks
Select this option to gather the information needed to get a File Installation Key.
Click Next.

Step 5: Choose the Installation Type

In the Installation Type dialog box, specify whether you want to perform a Custom installation and click **Next**.



Choose installation type

Typical

Installs all your licensed products using default settings. This option will download and install the latest product versions.

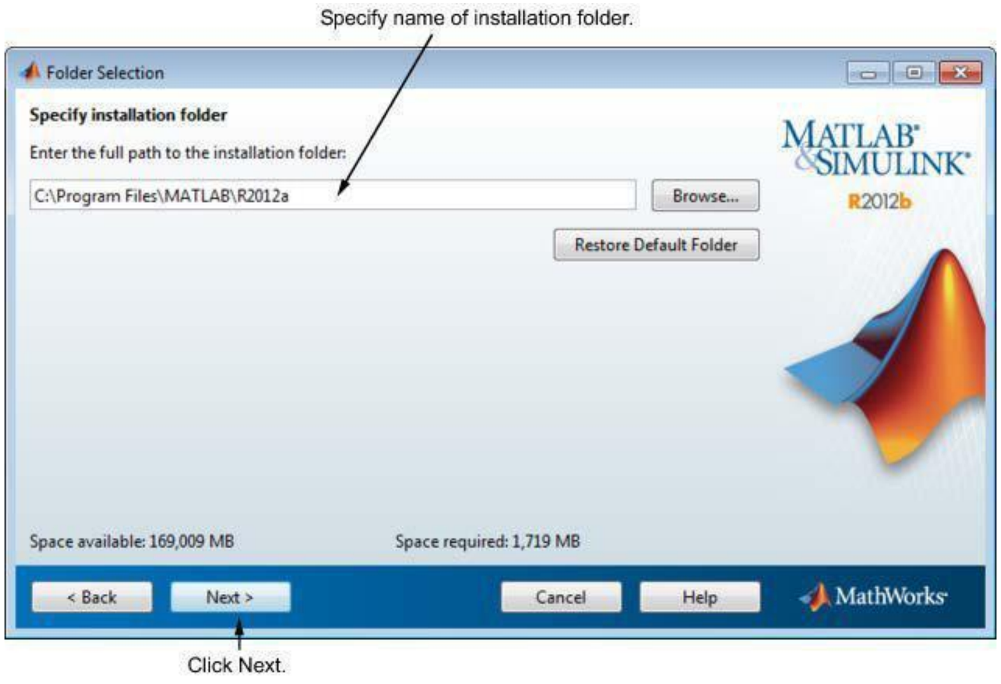
Custom

Specify all installation options:

- Products
- Shortcuts

Step 6: Specify the Installation Folder

Specify the name of the folder where you want to install MathWorks products. Accept the default installation folder or click Browse to select a different one. If the folder doesn't exist, the installer creates it.



Specify installation folder

Enter the full path to the installation folder:

C:\Program Files\MATLAB\R2012a

Space available: 169,009 MB

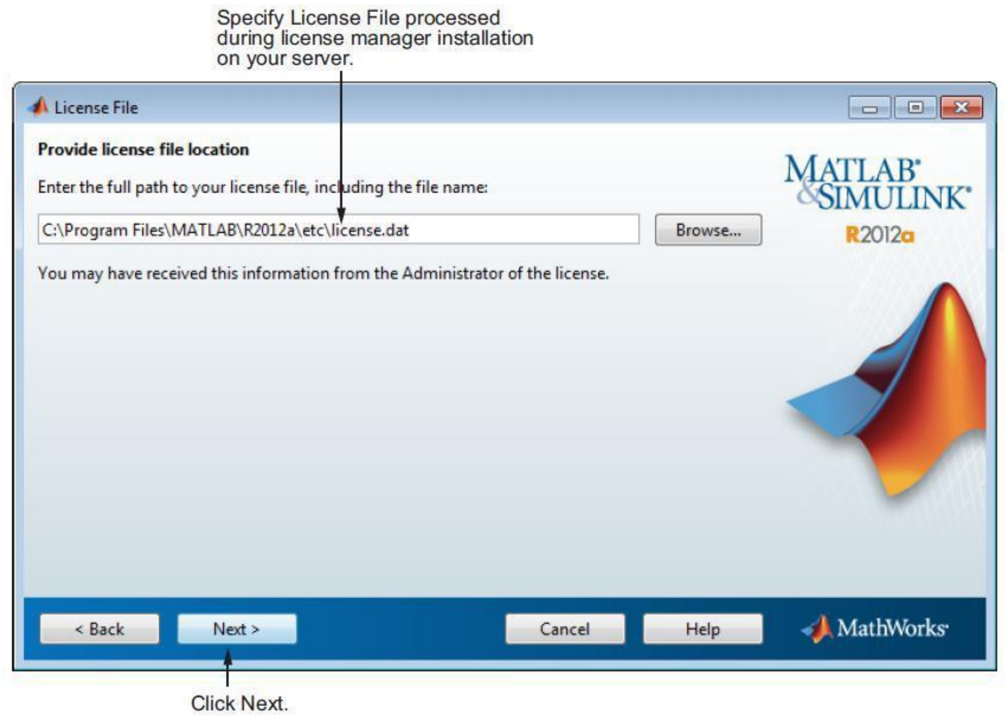
Space required: 1,719 MB

Step 7: Specify Products to Install (Custom Only)

Leave it by default and continue.

Step 8: Specify the Location of the License File

Enter the full path of your License File in the text box (or drag and drop the file) and click **Next**.



Provide license file location

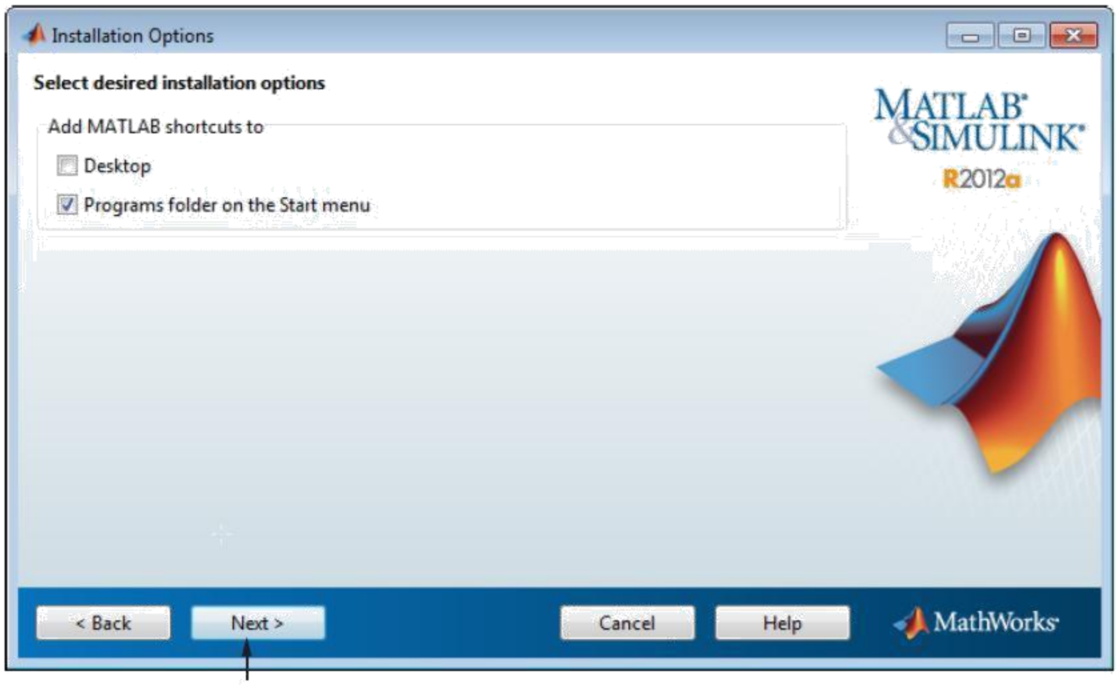
Enter the full path to your license file, including the file name:

C:\Program Files\MATLAB\R2012a\etc\license.dat

You may have received this information from the Administrator of the license.

Step 9: Specify Installation Options (Custom Only)

After selecting installation options, click **Next** to proceed with the installation.



Installation Options

Select desired installation options

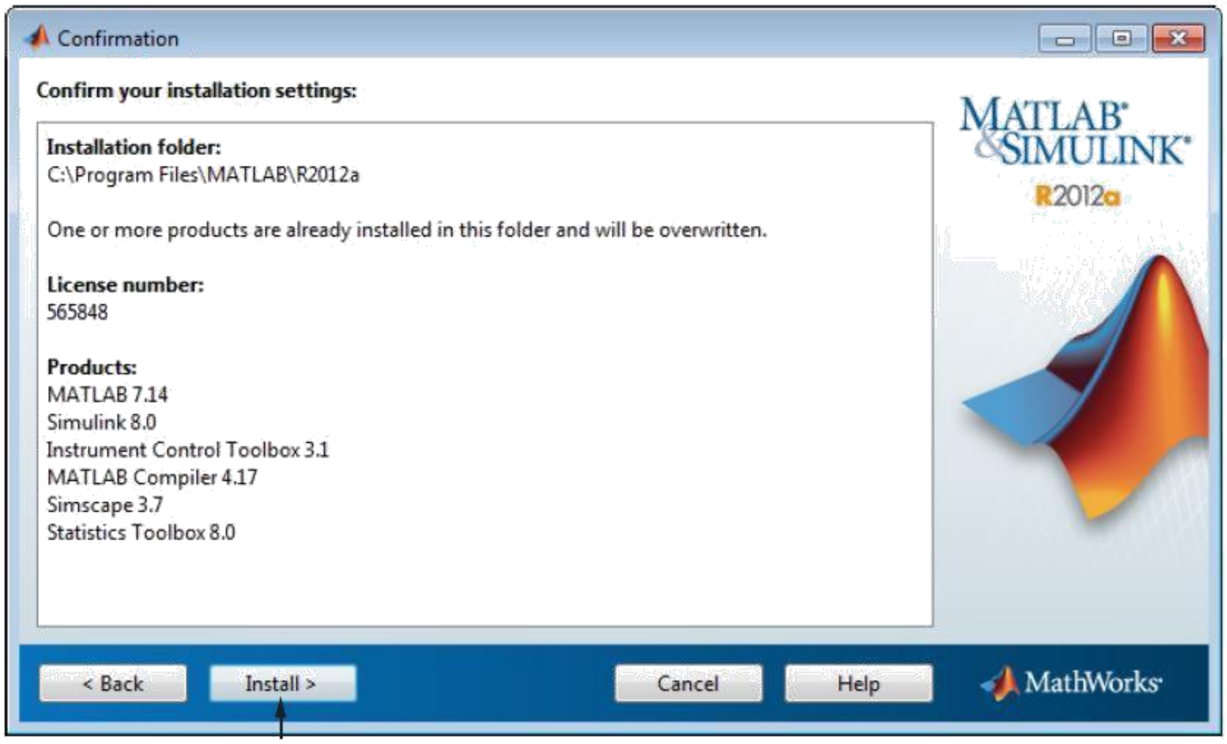
Add MATLAB shortcuts to

- [ ] Desktop
- [x] Programs folder on the Start menu

Click Next.

Step 10: Confirm Your Choices and Begin Copying Files

Before it begins copying files to your hard disk, the installer displays a summary of your installation choices. To change a setting, click **Back**. To proceed with the installation, click **Install**.


Confirmation

Confirm your installation settings:

Installation folder:
C:\Program Files\MATLAB\R2012a

One or more products are already installed in this folder and will be overwritten.

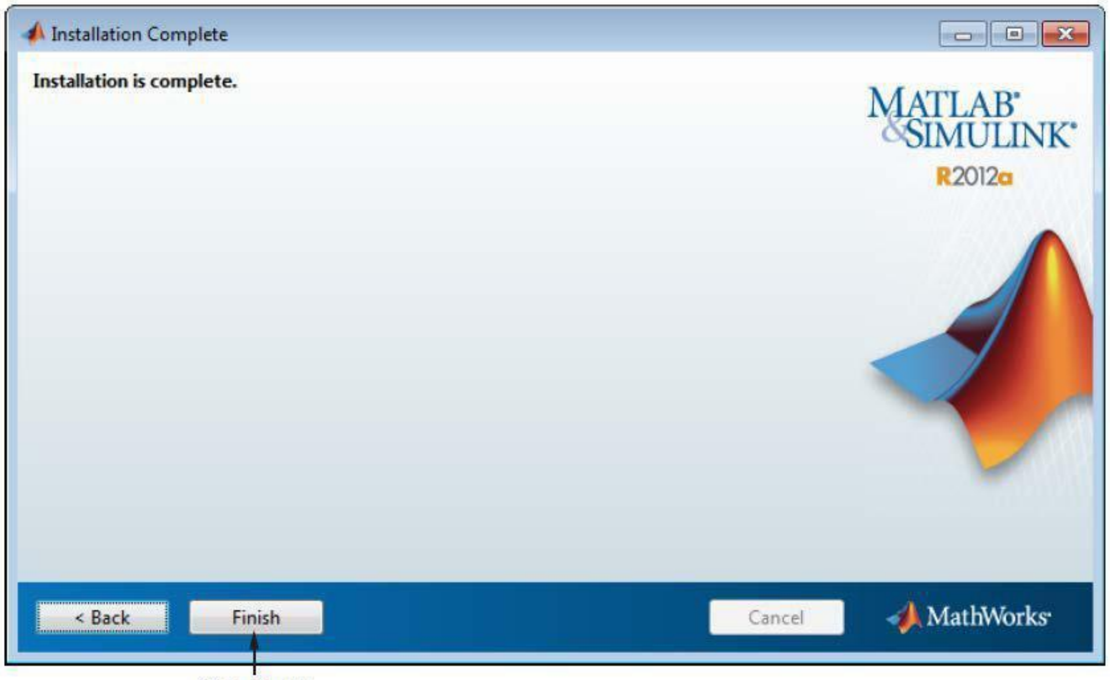
License number:
565848

Products:
MATLAB 7.14
Simulink 8.0
Instrument Control Toolbox 3.1
MATLAB Compiler 4.17
Simscape 3.7
Statistics Toolbox 8.0

Click Install.

As it copies files to your hard drive, the installer displays a status dialog box to show the progress of the installation.

Step 11: Complete the Installation


Installation Complete

Installation is complete.

Click Finish.

Step 12: Set Environment Variables

Click right Button on My Computer

Select Properties……..>Advanced System settings …….>Environment Variables…>New

The environment variable that should be added is as follows:

Variable_ name: MLM_LICENSE_FILE

Variable_value:27000@licmngr1.iitd.ernet.in, 27000@licmngr2.iitd.ernet.in, 27000@ licmanager.cse.iitd.ernet.in.

7.3 REQUIREMENTS

Development of Arduino can be done on a reasonably sized computer. For a nice experience a modern computer is recommended. For example, 2.6GHz CPU with at least 8GB of memory is required.

7.4 CODING

Once the installation procedure is complete the application enters into the coding and testing phase. The coding phase brings the actual system into action by converting the design of the application into the code in a given programming language. Therefore, a good coding style has to be taken and whenever changes are required, it should be easily screwed into the system.

7.5 CODING STANDARDS

Coding standards are guidelines to programming that focuses on the physical structure and appearance of the program. They make the code easier to read, understand and maintain. This phase of the system actually implements the blueprint developed during the design phase. The coding specification should be in such a way that any programmer must be able to understand the code and can bring about changes whenever felt necessary.

CHAPTER 8

IMPLEMENTATION AND RESULT

8.1 IMPLEMENTATION

8.1.1 MATLAB CODE:

```
function varargout = Weather_Prediction(varargin)
% WEATHER_PREDICTION MATLAB code for Weather_Prediction.fig
%      WEATHER_PREDICTION, by itself, creates a new

%      singleton*.
%
%      H = WEATHER_PREDICTION returns the handle to a
new WEATHER_PREDICTION or the handle to
%      the existing singleton*.
%
%      WEATHER_PREDICTION('CALLBACK',hObject,eventData,handles,...)
calls the local
%      function named CALLBACK in WEATHER_PREDICTION.M with
the given input arguments.
%
%      WEATHER_PREDICTION('Property','Value',...) creates a new
WEATHER_PREDICTION or raises the
%      existing singleton*. Starting from the left, property value pairs are
%      applied to the GUI before Weather_Prediction_OpeningFcn gets called.
An
%      unrecognized property name or invalid value makes property application
%      stop. All inputs are passed to Weather_Prediction_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Weather_Prediction

% Last Modified by GUIDE v2.5 21-Feb-2020 20:48:14
% Begin initialization code - DO NOT
EDIT gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton', gui_Singleton, ...
                   'gui_OpeningFcn', @Weather_Prediction_OpeningFcn, ...
                   'gui_OutputFcn', @Weather_Prediction_OutputFcn, ...
                   'gui_LayoutFcn', [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Weather_Prediction is made visible.
function Weather_Prediction_OpeningFcn(hObject, eventdata,
handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA) %
varargin command line arguments to Weather_Prediction (see
VARARGIN)

% Choose default command line output for
Weather_Prediction handles.output = hObject;
axes(handles.axes1); axis off
axes(handles.axes2); axis off
axes(handles.axes3); axis off
axes(handles.axes4); axis off
set(handles.edit2,'String','**');
set(handles.edit3,'String','**');
set(handles.edit4,'String','**');
set(handles.edit5,'String','**');
set(handles.edit6,'String','**');
set(handles.edit7,'String','**');
set(handles.edit8,'String','**');
set(handles.edit9,'String','**');
set(handles.edit10,'String','**');
set(handles.edit11,'String','**');



% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Weather_Prediction wait for user response (see
UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Weather_Prediction_OutputFcn(hObject,
eventdata, handles)
% varargout cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
global a;global skw;
[fname,path]=uigetfile('*.*','Browse
Image'); if fname~=0
   img=imread([path,fname]);
   a=img;
   axes(handles.axes1); imshow(img); title('Original Image');
else
   warndlg('Please Select the necessary Image File');
end

me=mean2(a);%mean
```

```
sd=std2(a);%std dev
en=entropy(a);%entropy
skw=skewness(a(:));%skewness
k=kurtosis(a(:));
set(handles.edit2,'String',me);
set(handles.edit3,'String',sd);
set(handles.edit4,'String',en);
set(handles.edit5,'String',k);
set(handles.edit6,'String',skw);


% preprocessing
nBins=5;
winSize=7;
nClass=6;
I = colImgSeg(a, nBins, winSize, nClass);
axes(handles.axes2); imshow(I);title('Preprocessed Image');
colormap('default');


% segment
I3 = im2double(I);
I3 = I3(:,:,1);
image_Segment = zeros(size(I3));
for ii=1:size(I3,1)
    for jj=1:size(I3,2)
        pixel=I3(ii,jj);
          if pixel<0.5
              new_pixel=0;
          elseif pixel>3
              new_pixel=256;
          else
              new_pixel = pixel;
          end
          image_Segment(ii,jj)=new_pixel;
      end
  end
axes(handles.axes3);imshow(image_Segment,[]);title('Segmented Image');

% classify
BW = imbinarize(I);
[B,L] = bwboundaries(BW,'noholes');
axes(handles.axes4);
```

```
imshow(label2rgb(L, @jet, [.5 .5 .5]))
% hold on
% for k = 1:length(B)
%     boundary = B{k};
%     plot(boundary(:,2), boundary(:,1), 'w', 'LineWidth', 2)
% end
% num_iter = 10;
% delta_t = 1/7;
% kappa = 15;
% option = 2;
% ad = anisodiff(a,num_iter,delta_t,kappa,option);
% m = zeros(size(ad,1),size(ad,2));
% m(90:100,110:135) = 1;
% ad = imresize(ad,.5);
% m = imresize(m,.5);
% axes(handles.axes4); imshow(ad,[]);
% hold on
% if(strcmp(I,'a1.jpg')||strcmp(I,'a.jpg'))
%     rectangle('Position',[40 47 20 22],'EdgeColor','y'); %a1
% end
% if(strcmp(I,'b1.jpg')||strcmp(I,'b.jpg'))
%     rectangle('Position',[61 49 18 20],'EdgeColor','y'); %b1
% end
% if(strcmp(I,'c1.jpg')||strcmp(I,'c.jpg'))
%     rectangle('Position',[35 26 34 40],'EdgeColor','y'); %c1
% %   cd=imcrop(ad,[35 26 34 40]);
% end
%
% hold off
% title('Locating Bounding box');
% % seg = svm(ad, m, 45);
% axes(handles.axes4); imshow(seg);
title('Classified Image');

acc=accuracy_image(label2rgb(L, @jet, [.5 .5 .5]));
sen=Sensitivity_image(label2rgb(L, @jet, [.5 .5
.5])); spe=specificity_image(label2rgb(L, @jet, [.5 .5
.5])); set(handles.edit9,'String',acc);
set(handles.edit10,'String',sen);
set(handles.edit11,'String',spe); g=round(me);

disp(g);
```

```
if(g==125)
s = serial('COM11');
fopen(s)
fprintf(s,'Data received from MATLAB')
pause(2);
fclose(s)
end




% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close Weather_Prediction


% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
axes(handles.axes1); cla(handles.axes1); title(''); axis off
axes(handles.axes2); cla(handles.axes2); title(''); axis off
axes(handles.axes3); cla(handles.axes3); title(''); axis off
axes(handles.axes4); cla(handles.axes4); title(''); axis off
set(handles.edit2,'String','---');
set(handles.edit3,'String','---');
set(handles.edit4,'String','---');
set(handles.edit5,'String','---');
set(handles.edit6,'String','---');
set(handles.edit7,'String','---');
set(handles.edit8,'String','---');
set(handles.edit9,'String','---');
set(handles.edit10,'String','---');
set(handles.edit11,'String','---');




% --- Executes on button press in pushbutton4.
```

```
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
% msgbox('LOAD THE
 DATASET....'); pause(2)
 msgbox('INITIALIZATION....');
 pause(2)
 msgbox('CREATE THE
 DATABASE.....'); pause(2)

for i=1:4
    j=num2str(i);
    imgname=strcat(j,'.png');
    TrainData=imread(imgname);
    TrainData=imresize(TrainData,[250 250]);
    TrainData = im2double(TrainData);
save database TrainData
end
 for i=8:13
    j=num2str(i);
    imgname=strcat(j,'.jpg');
    TrainData=imread(imgname);
    TrainData=imresize(TrainData,[250 250]);
    TrainData = im2double(TrainData);
save database TrainData
end
 msgbox('DATABASE SAVED SUCCESSFULLY....')
 pause(2)



function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a double


% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to edit1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB %
handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit7_Callback(hObject, eventdata, handles)
% hObject    handle to edit7 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit7 as text
%        str2double(get(hObject,'String')) returns contents of edit7 as a double


% --- Executes during object creation, after setting all properties.
function edit7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit7 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB %
handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit8_Callback(hObject, eventdata, handles)
% hObject    handle to edit8 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit8 as text
%       str2double(get(hObject,'String')) returns contents of edit8 as a double


% --- Executes during object creation, after setting all properties.
function edit8_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit8 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB %
handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit9_Callback(hObject, eventdata, handles)
% hObject    handle to edit9 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit9 as text
%       str2double(get(hObject,'String')) returns contents of edit9 as a double


% --- Executes during object creation, after setting all properties.
function edit9_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit9 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB %
handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit10_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to edit10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit10 as text
%        str2double(get(hObject,'String')) returns contents of edit10 as a double


% --- Executes during object creation, after setting all properties.
function edit10_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB %
handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit11_Callback(hObject, eventdata, handles)
% hObject    handle to edit11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit11 as text
%        str2double(get(hObject,'String')) returns contents of edit11 as a double


% --- Executes during object creation, after setting all properties.
function edit11_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB %
handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

```
end



function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2 as a double


% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB %
handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit3 as text
%        str2double(get(hObject,'String')) returns contents of edit3 as a double


% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB %
handles empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit4 as text
%        str2double(get(hObject,'String')) returns contents of edit4 as a double


% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB %
handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit5_Callback(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit5 as text
%        str2double(get(hObject,'String')) returns contents of edit5 as a double


% --- Executes during object creation, after setting all properties.
```

```
function edit5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB %
handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit6_Callback(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit6 as text
%        str2double(get(hObject,'String')) returns contents of edit6 as a double


% --- Executes during object creation, after setting all properties.
function edit6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB %
handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on selection change in listbox1. function
listbox1_Callback(hObject, eventdata, handles)
% hObject    handle to listbox1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
% Hints: contents = cellstr(get(hObject,'String')) returns listbox1 contents as
cell array
%        contents{get(hObject,'Value')} returns selected item from listbox1


% --- Executes during object creation, after setting all properties.
function listbox1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to listbox1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB %
handles empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit12_Callback(hObject, eventdata, handles)
% hObject    handle to edit12 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit12 as text
%        str2double(get(hObject,'String')) returns contents of edit12 as a double


% --- Executes during object creation, after setting all properties.
function edit12_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit12 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB %
handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

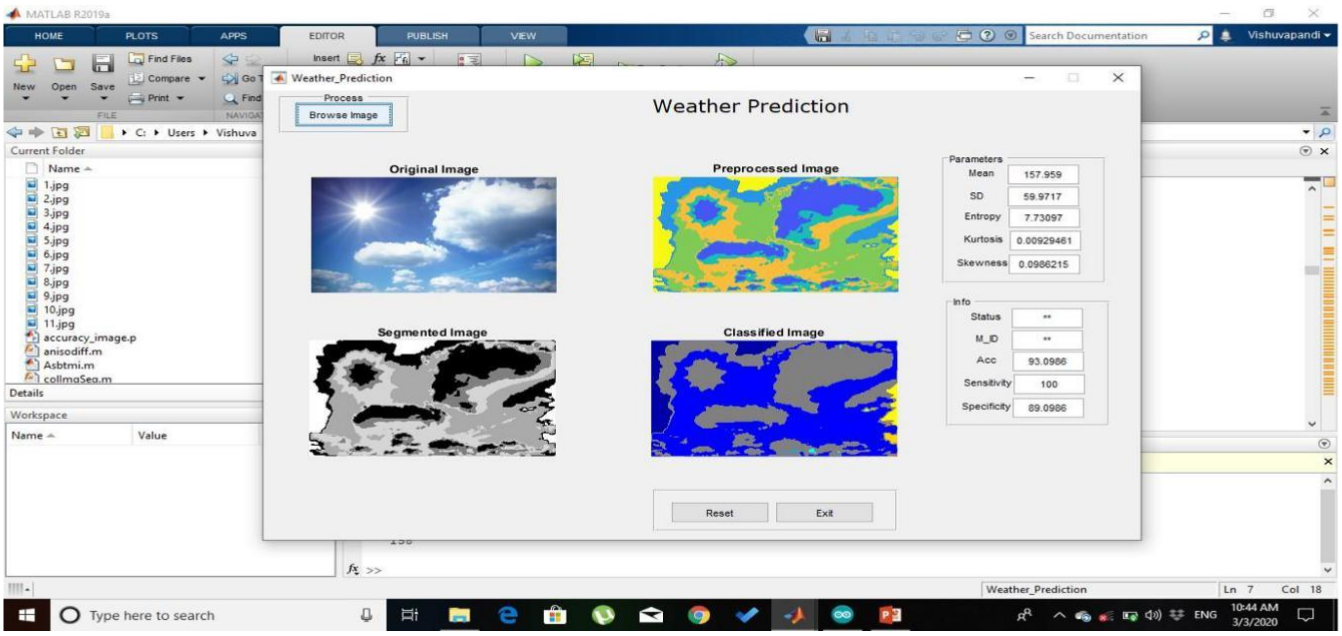
```
function edit13_Callback(hObject, eventdata, handles)
% hObject    handle to edit13 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit13 as text
%        str2double(get(hObject,'String')) returns contents of edit13 as a double


% --- Executes during object creation, after setting all properties.
function edit13_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit13 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB %
handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit14_Callback(hObject, eventdata, handles)
% hObject    handle to edit14 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit14 as text
%        str2double(get(hObject,'String')) returns contents of edit14 as a double


% --- Executes during object creation, after setting all properties.
function edit14_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit14 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB %
handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit15_Callback(hObject, eventdata, handles)
% hObject    handle to edit15 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit15 as text
%        str2double(get(hObject,'String')) returns contents of edit15 as a double


% --- Executes during object creation, after setting all properties.
function edit15_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit15 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB %
handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit16_Callback(hObject, eventdata, handles)
% hObject    handle to edit16 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit16 as text
%        str2double(get(hObject,'String')) returns contents of edit16 as a double


% --- Executes during object creation, after setting all properties.
function edit16_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit16 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB %
handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit17_Callback(hObject, eventdata, handles)
% hObject    handle to edit17 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit17 as text
%        str2double(get(hObject,'String')) returns contents of edit17 as a double


% --- Executes during object creation, after setting all properties.
function edit17_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit17 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB %
handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit18_Callback(hObject, eventdata, handles)
% hObject    handle to edit18 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit18 as text
%        str2double(get(hObject,'String')) returns contents of edit18 as a double
% --- Executes during object creation, after setting all properties.
function edit18_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit18 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB %
handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit19_Callback(hObject, eventdata, handles)
% hObject    handle to edit19 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit19 as text
%        str2double(get(hObject,'String')) returns contents of edit19 as a double


% --- Executes during object creation, after setting all properties.
function edit19_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit19 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB %
handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

8.1.2 ARDUINO CODE:

```
#define BLYNK_PRINT Serial
#include <SPI.h>
#include <Ethernet.h>
#include <BlynkSimpleEsp8266.h>
char auth[] = "EkaVcoEE2gu5jexlJuNL2Cdb26hvtDNv";
char ssid[] = "Admin";
char pass[] = "Admin@123";
String rx;
void setup()
{
Serial.begin(9600);
Blynk.begin(auth, ssid, pass);
Blynk.syncAll();
Blynk.notify("Yaaay... Device detected!");
}

void dataa()
{

  if(Serial.available()>0)
  {
    rx=Serial.readString();
    Blynk.notify(rx);
    int t=dht.temperature;
    int h=dht.humidity;

    Blynk.virtualWrite(V0, t);Blynk.virtualWrite(V1, h);
  }

//  Blynk.virtualWrite(V0, gsr);

}

void loop()
{
Blynk.run();
dataa();
}
```

8.2 RESULT



Parameters	
Mean	157.959
SD	59.9717
Entropy	7.73097
Kurtosis	0.00929461
Skewness	0.0986215

Info	
Status	**
M_ID	**
Acc	93.0986
Sensitivity	100
Specificity	89.0986

FIG 8.1 Screenshot of MATLAB Program Output Screen

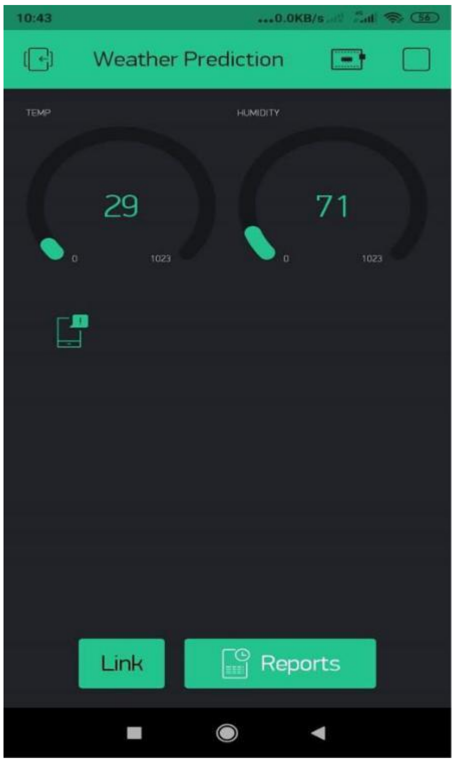


FIG 8.2 SCREENSHOT FROM ANDROID APPLICATION BLYNK

Thus the reports collected from the MATLAB and ARDUINO are processed to give final estimated temperature and humidity of the surrounding.

CHAPTER 9

CONCLUSION

Broadly speaking, in this introductory book we have presented a state-of-the-art of the following popular time series forecasting models with their salient features:

- The Box-Jenkins or ARIMA models for linear time series forecasting.
- Some non-linear stochastic models, such as NMA, ARCH.
- Neural network forecasting models; TLNN and SANN.
- SVM based forecasting models; LS-SVM and DLS-SVM.

It has been seen that, the proper selection of the model orders (in case of ARIMA), the number of input, hidden and output neurons (in case of ANN) and the constant hyper-parameters (in case of SVM) is extremely crucial for successful forecasting. We have discussed the two important functions, viz. AIC and BIC, which are frequently used for ARIMA model selection. For selecting the number of appropriate neurons in ANN and constant hyper-parameters in SVM, crossvalidation should be carried out, as mentioned earlier.

We have considered a few important performance measures for evaluating the accuracy of forecasting models. It has been understood that for obtaining a reasonable knowledge about the overall forecasting error, more than one measure should be used in practice. The last chapter contains the forecasting results of our experiments, performed on six real time series datasets. Our satisfactory understanding about the considered forecasting models and their successful implementation can be observed form the five performance measures and the forecast diagrams, we obtained for each of the six datasets. However in some cases, significant deviation can be seen among the original observations and our forecasted values. In such cases, we can suggest that a suitable data preprocessing, other than those we have used in our work may improve the forecast performances.

Time series forecasting is a fast growing area of research and as such provides many scope for future works. One of them is the *Combining Approach*, i.e. to combine a number of different and dissimilar methods to improve forecast accuracy. A lot of works have been done towards this direction and various combining methods have been proposed in literature [8, 14, 15, 16]. Together with other analysis in time series forecasting, we have thought to find an efficient combining model, in future if possible. With the aim of further studies in time series modeling and forecasting, here we conclude the present book.

CHAPTER 10

REFERENCES

[1] Burges, C.J.C., “A tutorial on support vector machines for pattern recognition”, Data Mining and Knowledge Discovery, 2 (1998), pages: 121-167.

[2] C. Chatfield, “Model uncertainty and forecast accuracy”, J. Forecasting 15 (1996), pages: 495–508.

[3] C. Hamzacebi, “Improving artificial neural networks’ performance in seasonal time series forecasting”, Information Sciences 178 (2008), pages: 4550-4559.

[4] F. Girosi, M. Jones, and T. Poggio, “Priors, stabilizers and basis functions: From regularization to radial, tensor and additive splines.” AI Memo No: 1430, MIT AI Lab, 1993.

[5] G. Zhang, B.E. Patuwo, M.Y. Hu, “Forecasting with artificial neural networks: The state of the art”, International Journal of Forecasting 14 (1998), pages: 35-62.

[6] G.E.P. Box, G. Jenkins, “Time Series Analysis, Forecasting and Control”, Holden-Day, San Francisco, CA, 1970.

[7] G.P. Zhang, “A neural network ensemble method with jittered training data for time series forecasting”, Information Sciences 177 (2007), pages: 5329–5346.

[8] G.P. Zhang, “Time series forecasting using a hybrid ARIMA and neural network model”, Neurocomputing 50 (2003), pages: 159–175.

[9] H. Park, “Forecasting Three-Month Treasury Bills Using ARIMA and GARCH Models”, Econ 930, Department of Economics, Kansas State University, 1999.

[10] H. Tong, “Threshold Models in Non-Linear Time Series Analysis”, Springer-Verlag, New York, 1983.

[11] J. Faraway, C. Chatfield, “Time series forecasting with neural networks: a comparative study using the airline data”, Applied Statistics 47 (1998), pages: 231–250.

[12] J. Lee, “Univariate time series modeling and forecasting (Box-Jenkins Method)”, Econ 413, lecture 4.

[13] J.M. Kihoro, R.O. Otieno, C. Wafula, “Seasonal Time Series Forecasting: A Comparative Study of ARIMA and ANN Models”, African Journal of Science and Technology (AJST) Science and Engineering Series Vol. 5, No. 2, pages: 41-49.

[14] J. Scott Armstrong, “Combining Forecasts: The End of the Beginning or the Beginning of the End?”, International Journal of Forecasting 5 (1989), pages: 585-588

[15] J. Scott Armstrong, “Combining Forecasts”, Principles of Forecasting: A Handbook for Researchers and Practitioners; J. Scott Armstrong (ed.): Norwell, MA: Kluwer Academic Publishers, 2001.

[16] J. Scott Armstrong, “Findings from evidence-based forecasting: Methods for reducing forecast error”, International Journal of Forecasting 22 (2006), pages: 583–598.

[17] J.W. Galbraith, V. Zinde-Walsh, “Autoregression-based estimators for ARFIMA models”, CIRANO Working Papers, No: 2011s-11, Feb 2001.

[18] J.A.K. Suykens and J. Vandewalle, “Least squares support vector machines classifiers”, Neural Processing Letters, vol. 9, no. 3, pp. 293-300, June 1999.

[19] J.A.K. Suykens and J. Vandewalle, “Recurrent least squares support vector machines”. IEEE Trans. Circuits Systems-I, vol. 47, no. 7, pp. 1109 - 1114, July 2000.

[20] Joarder Kamruzzaman, Rezaul Begg, Ruhul Sarker, “Artificial Neural Networks in Finance and Manufacturing”, Idea Group Publishing, USA.

[21] John H. Cochrane, “Time Series for Macroeconomics and Finance”, Graduate School of Business, University of Chicago, spring 1997.

[22] K. Hornik, M. Stinchcombe, H. White, “Multilayer feed-forward networks are universal approximators”, Neural Networks 2 (1989), pages: 359–366.

[23] K.W. Hipel, A.I. McLeod, “Time Series Modelling of Water Resources and Environmental Systems”, Amsterdam, Elsevier 1994.

[24] L.J. Cao and Francis E.H. Tay “Support Vector Machine with Adaptive Parameters in Financial Time Series Forecasting”, IEEE Transaction on Neural Networks, Vol. 14, No. 6, November 2003, pages: 1506-1518.

[25] M. Cottrell, B. Girard, Y. Girard, M. Mangeas, C. Muller, “Neural modeling for time series: a statistical stepwise method for weight elimination”, IEEE Trans. Neural Networks 6 (1995), pages: 1355–1364.

[26] M.J. Campbell, A.M. Walker, “A survey of statistical work on the MacKenzie River series of annual Canadian lynx trappings for the years 1821–1934, and a new analysis”, J. R. Statist. Soc. Ser. A 140 (1977), pages: 411–431.

[27] R. Lombardo, J. Flaherty, “Modelling Private New Housing Starts In Australia”, Pacific-Rim Real Estate Society Conference, University of Technology Sydney (UTS), January 24-27, 2000.

[28] R. Parrelli, “Introduction to ARCH & GARCH models”, Optional TA Handouts, Econ 472 Department of Economics, University of Illinois, 2001.

[29] Satish Kumar, “Neural Networks, A Classroom Approach”, Tata McGraw-Hill Publishing Company Limited.

[30] T. Farooq, A. Guergachi and S. Krishnan, “Chaotic time series prediction using knowledge based Green’s Kernel and least-squares support vector machines”, Systems, Man and Cybernetics, 2007. ISIC. 7-10 Oct. 2007, pages: 373-378.

[31] T. Raicharoen, C. Lursinsap, P. Sanguanbhoki, “Application of critical support vector machine to time series prediction”, Circuits and Systems, 2003. ISCAS ’03.Proceedings of the 2003 International Symposium on Volume 5, 25-28 May, 2003, pages: V-741-V-744.

[32] T. Van Gestel, J.A.K. Suykens, D. Baestaens, A. Lambrechts, G. Lanckriet, B. Vandaele, B. De Moor, and J. Vandewalle, “Financial time series prediction using least squares support vector machines within the evidence framework”, IEEE Trans. Neural Networks, vol. 12, no. 4, pp. 809 - 821, July 2001.

[33] V. Vapnik, “Statistical Learning Theory”, New York: Wiley, 1998.

[34] Yugang Fan, Ping Li and Zhihuan Song, “Dynamic least square support vector machine”, Proceedings of the 6th World Congress on Intelligent Control and Automation (WCICA), June 21-23, 2006, Dalian, China, pages: 4886-4889.

CHAPTER 11

DATASET SOURCES

The Canadian Lynx Dataset

Original source: Elton, C. and Nicholson, M. (1942), "The ten year cycle in numbers of Canadian lynx", J. Animal Ecology, Vol. 11, pages: 215-244.

Website link: http://www.stats.uwo.ca/faculty/aim/epubs/mhsets/ecology/lynx.1

The Wolf's Sunspot Dataset

Original source: H. Tong, "Non-linear Time Series: A Dynamical System Approach", Oxford Statistical Science, Series 6, July 1993, pages: 469-471.

Website link:
http://www.stats.uwo.ca/faculty/aim/epubs/mhsets/annual/sunspt.1

The Airline Passenger Dataset

Original source: R. G. Brown, "Smoothing, Forecasting and Prediction of Discrete Time Series", Prentice-Hall, Englewood Cliffs, 1994. This dataset is also used by Box and Jenkins in [6] and they had named it as Series G.

Website link: http://robjhyndman.com/TSDL/data/airpass.dat

The Quarterly Sales Dataset

Original source: S. Makridakis, S. Wheelwright, R. J. Hyndman, "Forecasting: Methods and Applications", 3rd edition, John Wiley & Sons, New York, 1998.

Website link: http://robjhyndman.com/forecasting/data/qsales.csv

The Quarterly U.S. Beer Production Dataset

Original source: William W. S. Wei, "Time Series Analysis: Univariate and Multivariate Methods", 2nd edition, Addison Wesley; July, 2005.

Website link: http://astro.temple.edu/~wwei/datasets/W10.txt

The Monthly USA Accidental Deaths Dataset (1973-1978)

Original source: P. J. Brockwell, R. A. Davis, “Introduction to Time Series and Forecasting”, 2nd edition, Springer Publication; March, 2003.

Website link: http://robjhyndman.com/forecasting/data/deaths.csv