**TRAINING REPORT**

**On**

**STOCK MARKET PREDICTOR**

Submitted to MAHARAJA RANJIT SINGH PUNJAB TECHNICAL

UNIVERSITY in partial fulfillment of the requirement for the award of the degree of

**B.TECH**

**in**

**COMPUTER SCIENCE & ENGINEERING**

**Submitted By**

**CHANDRA BHUSHAN**

**Roll No.: 15110151**

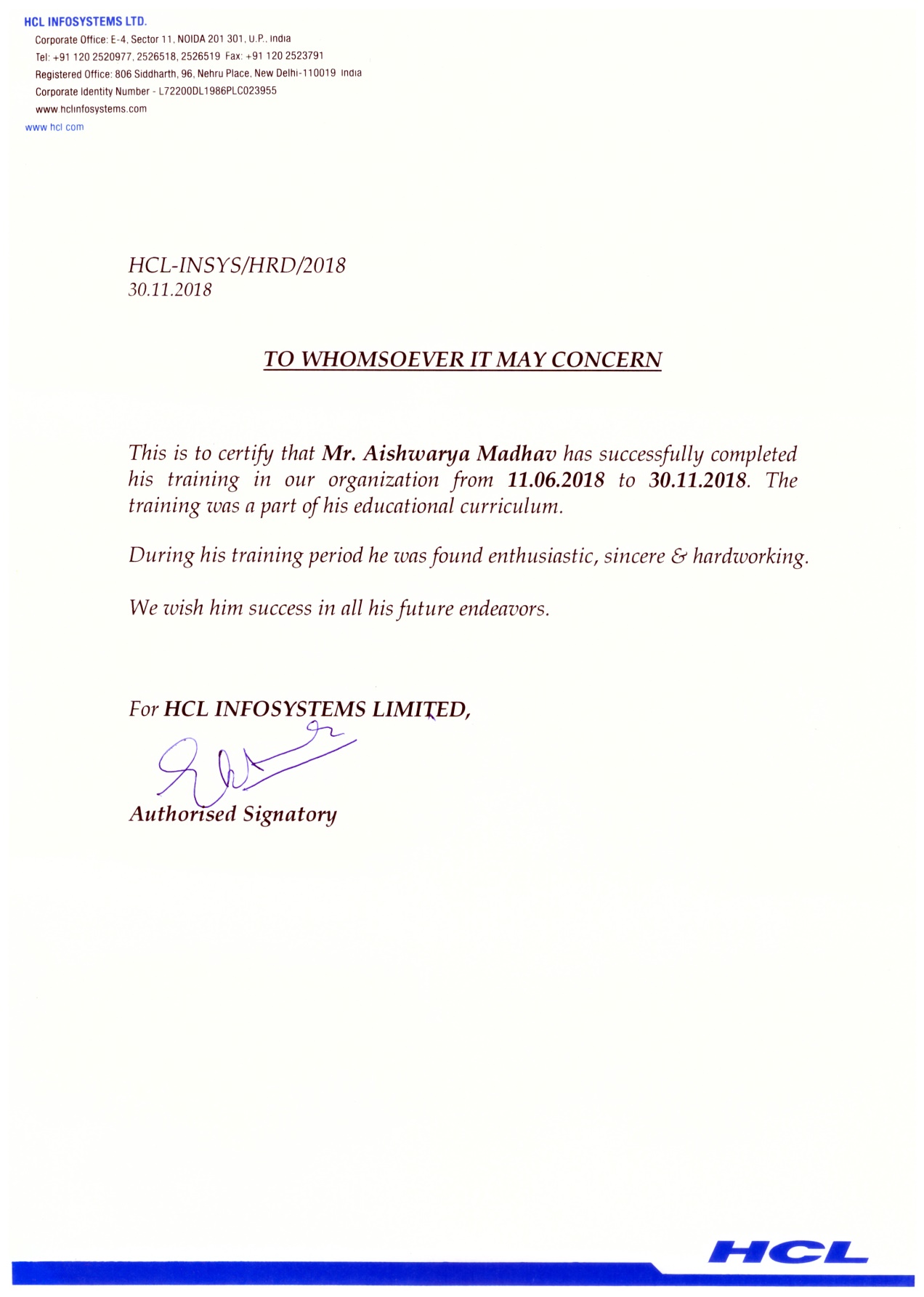
****

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**GIANI ZAIL SINGH CAMPUS COLLEGE OF ENGINEERING & TECHNOLOGY, BATHINDA-151001**

**DEC 2018**

**CERTIFICATE BY COMPANY**



**PREFACE**

Training is an integral part of B.Tech each and every student has to undergo the training for 6 months in a company.

This record is concerned about our practical training during the 7th semester of our B.Tech. We have taken our practical training in (Data Science- RStudio, Python3.5, Anaconda, Spyder, Hive, Tableau, ).

During this training, we got to learn many new things about the industry and the current requirements of companies. This training proved to be a milestone in our knowledge of present industry. Every moment was an experience in itself, an experience which theoretical study can’t provide.

**ACKNOWLEDGEMENT**

It is my pleasure to be indebted to various people, who directly or indirectly contributed in the development of this work and who influenced my thinking, behaviour and acts during the course of study.

I express my sincere gratitude to ***Dr.Naresh Garg*** worthy HOD and ***Er.Dinesh Kumar*** and ***Er. Manpreet Kaur***. Training & Placement In-charge for providing me an opportunity to undergo summer training at Naresh I Technology.

I am thankful to ***Naveen Sir*** for support, cooperation, and motivation provided to me during the training for constant inspiration, presence and blessings.

I also extend my sincere appreciation to ***Naveen Sir*** who provided his valuable

suggestions and precious time in accomplishing my training report.

Lastly, I would like to thank the almighty and my parents for their moral support and my friends with whom I shared my day-to day experience and received lots of suggestions that my quality of work.

**CHANDRA BHUSHAN**

**CANDIDATE’S DECLARATION**

I, Chandra Bhushan, Roll No. 15110151, B.Tech(Semester-VII) of the **Giani Zail Singh Campus College** **of Engineering & Technology, Bathinda** hereby declare that Training Report entitled “**STOCK MARKET PREDICTOR”** is an original work and data provided in study is authentic to the best of my knowledge.This report has not been submitted to any other Institute for the award of any other degree.

**Chandra Bhushan**

(RollNo. 15110151)

**Place:** **BATHINDA**

**Date: 12-11-2018**

**INDEX**

|  |  |
| --- | --- |
| **Chapters** | **Page No.** |
| 1. **Introduction** |  |
| * 1. Stock Market Prediction   2. Exiting System   3. Disadvantages of the Existing System   4. Proposed System   5. Advantages of the Proposed System  1. **Literature Survey**    1. Time-series forecasting    2. Support Vector Regression    3. Least Squares Support Vector Regression    4. Hyperparameter Optimization    5. Firefly Algorithm 2. **System Requirement Specification**    1. Non-functional requirements    2. Specific Requirements    3. Software requirements    4. Hardware requirements      1. **System Design**    1. Dataflow Diagram    2. Use Case Diagram    3. Sequence Diagram    4. Flowchart 2. **Implementation**    1. Modules    2. Overall Implementation 3. **Testing**    1. Test Objectives    2. Testing Principles    3. Testing Design    4. Testing Strategies    5. Levels of Testing 4. **Performance Analysis**    1. IBM Stock Prediction    2. CISCO Stock Prediction    3. Multiple Stock Performance    4. Overall System Performance 5. **Results** 6. **Screenshots** 7. **Future Enhancement**    1. Limitations    2. Future Enhancement 8. **Conclusion**   **References** | 1  2  2  3  3  4  4  4  5  5  6  7  7  8  9  10  11  13  14  16  17  17  17  18  19  24  26  28  29  30  33  35  35  36  37 |

**LIST OF FIGURES AND TABLES**

|  |  |
| --- | --- |
| **FIGURES** | **Page No.** |
| 1. Dataflow diagram of proposed system 2. Use case diagram of the proposed system 3. Sequence diagram of the proposed system 4. Flowchart of the proposed system 5. IBM Stock Prediction for 90 days 6. IBM: Day wise accuracy 7. IBM: Predictive Performance 8. Cisco Stock Prediction for 90 days 9. Cisco Day wise accuracy 10. Cisco: Predictive Performance 11. Individual Predictive Performance 12. Overall System Performance 13. IBM Stock Prediction for 10 days 14. Interface of the system 15. The screenshot of system 16. Predictive performance of the entire system | 9  10  12  13  24  25  26  26  27  27  28  29  32  33  34  34 |
|  |  |

**LIST OF TABLES**

|  |  |
| --- | --- |
| **TABLES**   * 1. Unit Test Case 1   2. Unit Test Case 2   3. Functional testing items   7.1 Input provided for performance analysis  8.1 Requesting input for prediction  8.2 Predicted closing stock prices  8.3 Basic summary statistics  8.4 Results with actual and predicted prices | 20  20  21  24  30  30  31  31 |

**ABSTRACT**

Time series forecasting has been widely used to determine the future prices of stock, and the analysis and modelling of finance time series importantly guide investors’ decisions and trades. In addition, in a dynamic environment such as the stock market, the non-linearity of the time series is pronounced, immediately affecting the efficacy of stock price forecasts. Thus, this work proposes an intelligent time series prediction system that uses sliding-window metaheuristic optimization for the purpose of predicting the stock prices of multiple companies taken at random. It may be of great interest to home brokers who do not possess sufficient knowledge to invest in such companies. The system has a graphical user interface and functions as a stand-alone application. The developed hybrid system exhibited outstanding prediction performance and it improves overall profit for investment performance. The proposed model is a promising predictive technique for highly non-linear time series, whose patterns are difficult to capture by traditional models.

**Chapter-1**

**INTRODUCTION**

* 1. **Stock Market Prediction**

Financial markets are highly volatile and generate huge amounts of data daily. Investment is a commitment of money or other resources to obtain benefits in the future. Stock is one type of securities. It is the most popular financial market instrument and its value changes quickly. It can be defined as a sign of capital participation by a person or an enterprise in a company or a limited liability company. The stock market provides opportunities for brokers and companies to make investments on neutral ground [1].

Stock prices are predicted to determine the future value of companies’ stock or other financial instruments that are marketed on financial exchanges. However, the stock market is characterized by nonlinearities, discontinuities, and high-frequency multi-polynomial components because it interacts with many factors such as political events, general economic conditions, and traders’ expectations. Therefore, making precise predictions of stock values are challenging [2].

Investors can buy stocks that are related to the construction firms that design infrastructure projects, hire contractors and handle paperwork, and decision-makers of construction firms can buy stocks from other companies. When the direction of the market is successfully predicted, investors may be better guided and monetary rewards will be substantial. The challenge in today’s environment, where bad news can always be heard, is to forecast proactively, rather than reactively. Therefore, construction corporations are trying to predict stock prices which is important to be considered on a financial exchange, against sudden drops in the market.

Time series forecasting consists in a research area designed to solve various problems, mainly in the financial area. It is noteworthy that this area typically uses tools that assist in planning and making decisions to minimize investment risks. This objective is obvious when one wants to analyze financial markets and, for this reason, it is necessary to assure a good accuracy in forecasting tasks [3].

* 1. **Existing System**

Time series forecasting consists in a research area designed to solve various problems, mainly in the financial area. It is noteworthy that this area typically uses tools that assist in planning and making decisions to minimize investment risks. This objective is obvious when one wants to analyze financial markets and, for this reason, it is necessary to assure a good accuracy in forecasting tasks. [3]

Machine learning (ML) is coming into its own that can play a key in a wide range of critical applications. In machine learning, support vector machines (SVMs) have many advanced features that are reflected in their good generalization capacity and fast computation. They are also not very sensitive to assumptions about error terms and they can tolerate noise and chaotic components. Notably, SVMs are increasingly used in materials science, the design of engineering systems and financial risk prediction. [1]

Also, most methods that are in use are only applicable to a small portion of stock markets and usually such models do not generalize well to all stocks. Additionally, existing libraries are highly efficient in obtaining the optimal hyperparameters to be used in LSSVM and other algorithms.

* 1. **Disadvantages of the Existing System**

Since time series data can be formulated by regression analysis, LSSVR is very efficient when applied to the issue at hand. However, the efficacy of LSSVR strongly depends on its tuning hyperparameters, which are the regularization parameter and the kernel function. Inappropriate settings of these parameters may lead to significantly poor performance of the model. Therefore, the evaluation of such hyperparameters is a real-world optimization problem. [4]

Because the performance of SVR-based models strongly depends on the setting of its hyperparameters, they used to be set in advance based on the experience of practitioners, by trial-and-error, or using a grid search algorithm. Thus, finding the optimal values of regularization and kernel function parameters for SVR-based models is an important and time-consuming step. Therefore, a means of automatically finding the hyperparameters of SVR, while ensuring its generalization performance, is required. [5]

* 1. **Proposed System**

Decision to buy or sell a stock is very complicated since many factors can affect stock price. This work presents a novel approach, based on least squares support vector regression (LSSVR), to constructing a stock price forecasting expert system, with the aim of improving forecasting accuracy. The intelligent time series prediction system that uses sliding-window metaheuristic optimization is a graphical user interface that can be run as a stand-alone application. The system makes the prediction of stock market values simpler, involving fewer computations, than that using the other method that was mentioned above [1].

Additionally, the proposed system automatically fetches the latest stock data for any given company and date range.

* 1. **Advantages of the Proposed System**

To evaluate the proposed approach, it was applied to five datasets for stocks in Taiwan, and three other stock datasets that have been used in other papers.

* Firstly, to generalize the application of the proposed system, our work uses the proposed system to estimate other stocks in similar emerging markets and mature markets, such as Vietnam, Indonesia, China, Japan, Hong Kong, Korea, Singapore, Europe, USA and India.
* Secondly, the system can be extended to analyze multivariate time series data and import raw dataset directly.
* Thirdly, profit can be maximized even when the construction corporate stock market is bullish. Finally, the development of a web-based application has been considered to improve the user-friendliness and usability of the expert system.

**Chapter 2**

**LITERATURE SURVEY**

Literature review is a text of a scholarly paper, which includes the current knowledge including substantive findings, as well as theoretical and methodological contributions to a particular topic. Literature reviews are secondary sources, and do not report new or original experimental work.

* 1. **Time-series forecasting**

According to Saini (2016), forecasting based on a time series represents a means of providing information and knowledge to support a subsequent decision [6]. Thus, the analysis of time series focuses on achieving dependency relationships among historical data. The two broad categories of forecasting models are linear and nonlinear. For many decades, traditional statistical forecasting models in financial engineering were linear. Some well-known statistical models can be used in time series forecasting [6].

* 1. **Support Vector Regression**

In machine learning, support vector regression (SVR) was developed by Vapnik *et al.* (1995) [7] and is a variant of the SVM. It is typically used to solve nonlinear regression problems by constructing the input-output mapping function. The least squares support vector regression (LSSVR) algorithm is a further development of SVR by Suykens (2001) [8] and involves equality instead of inequality constraints and works with a least squares objective function. The LSSVR approach considerably reduces computational complexity and increases efficiency compared to standard SVR. Hao et al. (2006) examined the feasibility of methods in stock composite index forecasting and improved the accuracy of parameter selection by SVR. They concluded that SVR has high prediction performance [9].

* 1. **Least Squares Support Vector Regression**

Some studies have demonstrated the superiority of LSSVR over standard support vector regression (SVR) for estimating product cost and energy utilization. LSSVR solves linear equations instead of a quadratic programming problem. It is preferred for large-scale regression problems is that demand fast computation.

* 1. **Hyperparameter Optimization**

Since time series data can be formulated by regression analysis, LSSVR is very efficient when applied to the issue at hand. However, the efficacy of LSSVR strongly depends on its tuning hyperparameters, which are the regularization parameter and the kernel function. Inappropriate settings of these parameters may lead to significantly poor performance of the model. Therefore, the evaluation of such hyperparameters is a real-world optimization problem.

Optimization is one of the cornerstones of science and engineering. Recently, the field of nature-inspired optimization algorithms has grown incredibly fast. The algorithms are usually general-purpose and population-based. They are normally referred to as evolutionary algorithms because many of them are motivated by biological evolution. In a broad sense, evolutionary algorithms cover those that iteratively vary a group of solutions based on some nature-inspired operations.

* 1. **Firefly Algorithm**

The firefly algorithm (FA) [10], which is a nature-inspired metaheuristic method, has recently performed extremely well in solving various optimization problems such as stock price forecasting and electricity price prediction. The standard FA was developed by modeling the behavior of tropical fireflies. Notably, the smart firefly algorithm-based LSSVR has been demonstrated to be very effective in solving complex problems in civil engineering.

Recent research suggests that hybrid forecasting models can be usefully applied to the stock market’s fluctuations, yielding satisfactory forecasting precision. The authors used a hybrid model to capture the linear and non-linear characteristics of a stock price time series and confirmed that hybrid forecasting models are powerful tools for practitioners in management science. A review of the literature has indicated that enhancing the effectiveness capability of least squares support vector regression based on a nature-inspired metaheuristic optimization algorithm, such as the firefly algorithm is an unsolved problem in the field of stock price prediction. This work develops an intelligent time series prediction system using sliding-window metaheuristic optimization least squares support vector regression (LSSVR).

**Chapter 3**

**SYSTEM REQUIREMENT SPECIFICATION**

To be used efficiently, all computer software needs certain hardware components or other software resources to be present on a computer. These prerequisites are known as (computer) system requirements and are often used as a guideline as opposed to an absolute rule. Most software defines two sets of system requirements: minimum and recommended. With increasing demand for higher processing power and resources in newer versions of software, system requirements tend to increase over time. Industry analysts suggest that this trend plays a bigger part in driving upgrades to existing computer systems than technological advancements.

Software requirements specification establishes the basis for an agreement between customers and contractors or suppliers on how the software product should function (in a market-driven project, these roles may be played by the marketing and development divisions). The software requirements specification lays out functional and non-functional requirements, and it may include a set of use cases that describe user interactions that the software must provide.

**3.1 Non-functional requirements**

Non-functional requirements are the functions offered by the system. It includes time constraints and constraints on the development process and standards. The non-functional requirements are as follows:

* **Speed:** The system should process the given input into output within appropriate time.
* **Ease of use:** The software should be user friendly. Then the customers can use easily, so it doesn’t require much training time.
* **Reliability:** The rate of failures should be less then only the system is more reliable
* **Portability**: It should be easy to implement in any system.

**3.1.1 Specific Requirements**

The specific requirements are:

* **User Interfaces:** The external users are the clients. All the clients can use this software for indexing and searching.
* **Hardware Interfaces:** The external hardware interface used for indexing and searching is personal computers of the clients. The PC’s may be laptops with wireless LAN as the internet connections provided will be wireless.
* **Software Interfaces:** The Operating Systems can be any version of Windows.
* **Performance Requirements:** The PC’s used must be atleast Pentium 4 machines so that they can give optimum performance of the product.

**3.2 Software requirements**

Software requirements deal with defining software resource requirements and prerequisites that need to be installed on a computer to provide optimal functioning of an application.

These requirements or prerequisites are generally not included in the software installation package and need to be installed separately before the software is installed.

**Operating System** Window/ Ubuntu Linux/MacOS

**Programming Language** R, Python

**IDE/Editor** RStudio, Spyder, Jupyter Notebook

**Required Python Packages** TensorFlow, Keras, Pandas, Numpy, Matplotlib, Sklearn, Pytorch

**3.3 Hardware requirements**

The most common set of requirements defined by any [operating system](http://en.wikipedia.org/wiki/Operating_system) or [software application](http://en.wikipedia.org/wiki/Software_application) is the physical computer resources, also known as [hardware](http://en.wikipedia.org/wiki/Computer_hardware), A hardware requirements list is often accompanied by a [hardware compatibility list](http://en.wikipedia.org/wiki/Hardware_compatibility_list), especially in case of operating systems. An HCL lists tested, compatible, and sometimes incompatible hardware devices for a particular operating system or application. The following sub-sections discuss the various aspects of hardware requirements.

All computer operating systems are designed for a particular computer architecture. Most software applications are limited to particular operating systems running on particular architectures. Although architecture-independent operating systems and applications exist, most need to be recompiled to run on a new architecture.

The power of the central processing unit (CPU) is a fundamental system requirement for any software. Most software running on x86 architecture define processing power as the model and the clock speed of the CPU. Many other features of a CPU that influence its speed and power, like bus speed, cache, and MIPS are often ignored.

This definition of power is often erroneous, as AMD Athlon and Intel Pentium CPUs at similar clock speed often have different throughput speeds.

**System/Processor** Intel Core i3,5,7, 2.4-3.0 GHz

**Hard Disk Space** 500 GB or more

**RAM** 4 GB or more

**Internet Connection** Required to auto-download dataset

**CHAPTER 4**

**SYSTEM DESIGN**

System design is the process of defining the architecture, components, modules, interfaces and [data](http://en.wikipedia.org/wiki/Data) for a [system](http://en.wikipedia.org/wiki/System) to satisfy specified [requirements](http://en.wikipedia.org/wiki/Requirement). One could see it as the application of [systems theory](http://en.wikipedia.org/wiki/Systems_theory) to [product development](http://en.wikipedia.org/wiki/Product_development). There is some overlap with the disciplines of [systems analysis](http://en.wikipedia.org/wiki/Systems_analysis), [systems architecture](http://en.wikipedia.org/wiki/Systems_architecture) and [systems engineering](http://en.wikipedia.org/wiki/Systems_engineering). If the broader topic of [product development](http://en.wikipedia.org/wiki/Product_development) "blends the perspective of marketing, design, and manufacturing into a single approach to product development," then design is the act of taking the marketing information and creating the design of the product to be manufactured. Systems design is therefore the process of defining and developing [systems](http://en.wikipedia.org/wiki/System) to satisfy specified [requirements](http://en.wikipedia.org/wiki/Requirement) of the user.

**4.1 Dataflow Diagram**

A data flow diagram is a graphical representation of the "flow" of data through an information system, modeling its process aspects. Often, they are a preliminary step used to create an overview of the system which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design).

Figure 4.1 Dataflow diagram of the proposed system

The DFD is also called as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of the input data to the system, various processing carried out on these data, and the output data is generated by the system.

After initialization the user is asked to select or input a company name using a Ticker. The system fetches the stock data online for a given input date range and plots the, on a graph. The data is sent to the train the LSSVR model. Stock Predictions for n-day ahead are performed and the model is saved for future use.

**4.2 Use Case Diagram**

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well.

A close up of a map

Description generated with high confidence

Figure 4.2 Use case diagram of the proposed system

The following steps are performed:

* Data is initially collected from online sources or the stock exchange, either by the system or the admin
* The data is then used to train the system
* Trained model is saved
* User views the trade exchange and stock of a company
* Using the model, closing prices are predicted

**4.3 Sequence Diagram**

A sequence diagramin a UML is a kind of [interaction diagram](http://en.wikipedia.org/wiki/Interaction_diagram) that shows how processes operate with one another and in what order. It is a construct of a [Message Sequence Chart](http://en.wikipedia.org/wiki/Message_Sequence_Chart). A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams typically are associated with use case realizations in the Logical View of the system under development.

The steps performed are as follows:

* User visits the application
* Previously saved model is loaded
* User requests for a company's stock data
* He requests for prediction to be made

The sequence diagram is shown below:

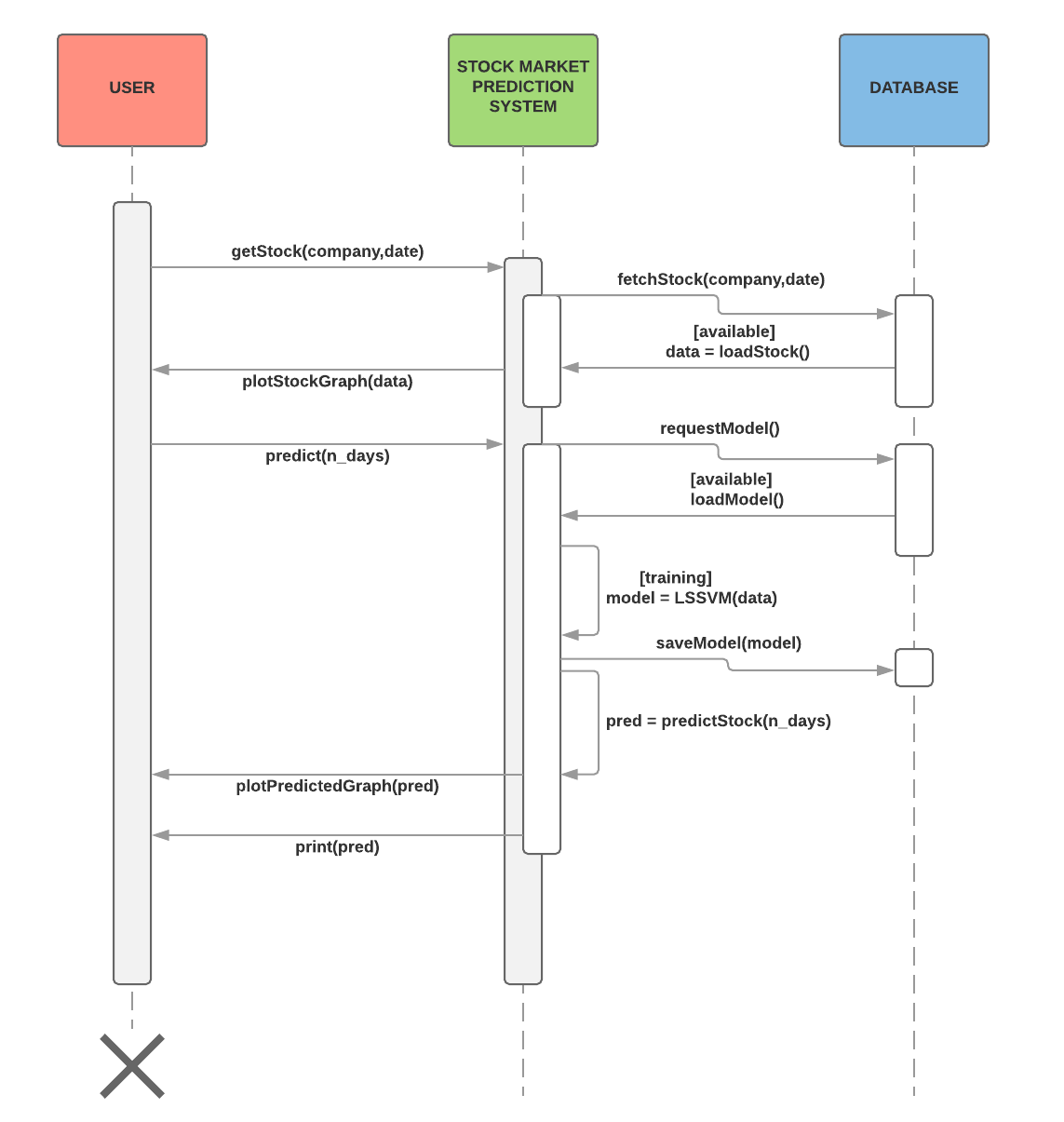


Figure 4.3 Sequence diagram of the proposed system

**4.4 Flowchart**

A flow chart is a graphical or symbolic representation of a process. Each step in the process is represented by a different symbol and contains a short description of the process step. The flow chart symbols are linked together with arrows showing the process flow direction.

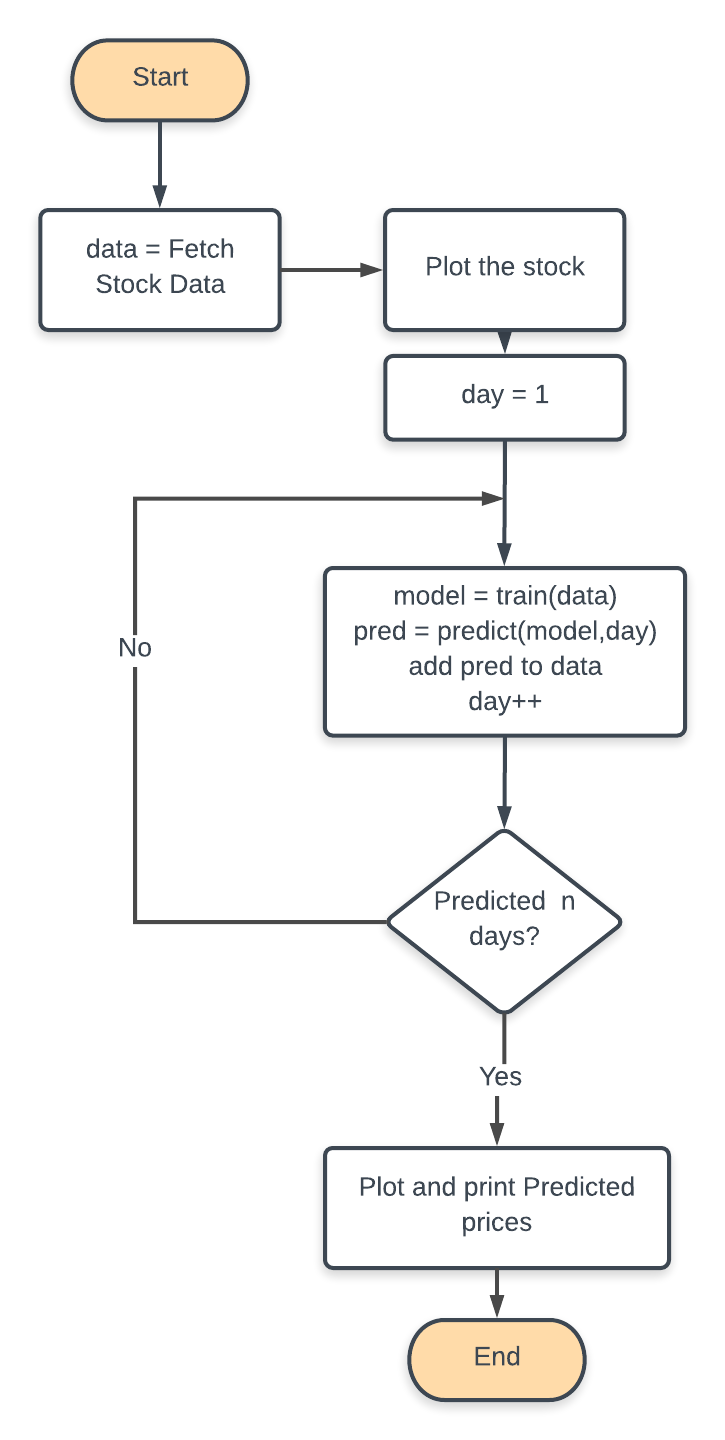


Figure 4.4 Flowchart of the proposed system

**CHAPTER 5**

**IMPLEMENTATION**

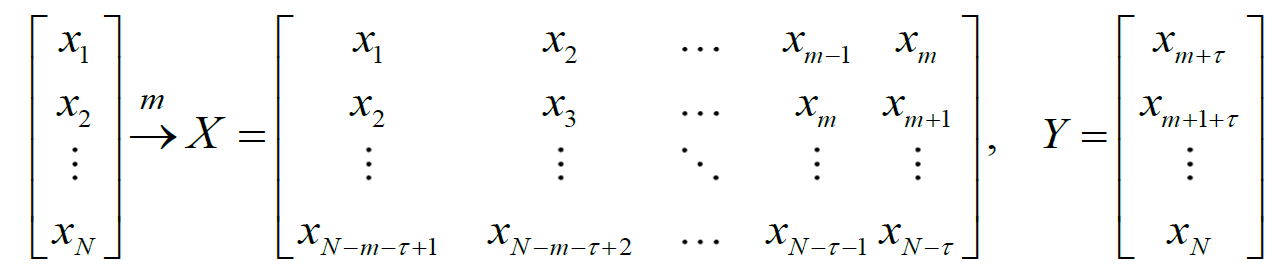
**5.1 Modules**

The proposed system has mainly three modules that together form the main system implementation.

**5.1.1 Phase Space Reconstruction**

In time series prediction, the time series are typically expanded into three or higher-dimensional space to exploit the information that is implicit in them. Selecting a suitable pairing of embedding dimension m (lag) and time delay τ is very important for phase space reconstruction.

Consider a time series .The time-delay vectors can be reconstructed as follows, where X is the input matrix and Y is the corresponding output matrix. The output of the analysis is fed back to the input and future values are predicted from previous values in the time series. [1]

**** (5.1)

**5.1.2 Sliding-window method**

As suggest in [1], the learning dataset used in this study was collected within a sliding-window. Fig. 1 depicts the sliding-window and phase space construction. Since the forecast is one step ahead (hence the term, “one-step ahead forecasting”), the forecast horizon is 1. In the first validation, the working window includes p historical observations which are used to forecast the next value . In the second validation, the oldest value is removed from the window and the latest value is added, keeping the length of the sliding window constant at *p.* The next forecast value will be . The window continues to slide until the end of the dataset is reached. If the number of observations is N, then the total number of validations is (N-p)

The algorithm for the sliding window is given as follows:

**Algorithm : slidingWindow**

**Input :** *data* [stock data]

**Output :** A data frame of a lagged stock data

1. LAG ← 1
2. *y* ← remove first LAG rows from data
3. reset row indices of *y*
4. *x* ← remove last LAG rows from data
5. *train* ← merge *(x,y)* into a dataframe
6. rename column name to *x* and *y*
7. **return** *train*

Algorithm 5.1 Sliding window algorithm

The above algorithm implemented as code in the R language is as follows:

**slidingWindow** = **function**(data)

{

y = data[-(1:LAG),]

**rownames**(y) = NULL

x= data[1:(DATA\_SIZE-LAG),]

train = **data.frame**(x[,2],y[,2])

**colnames**(train) = c("x","y")

**return**(train)

}

Snippet 5.1 Sliding window code

**5.1.3 Least Squares Support Vector Regression**

The LSSVR approach proposed by Suykens *et al.* (2002) [8] is a well-developed ML technique with many advanced features that support a high generalization capacity and fast computation. The LSSVR training process entails the use of a least squares cost function to obtain a linear set of equations in a dual space to minimize the computational cost. Accordingly, iterative methods, such as the conjugate gradient method are typically used to derive a solution by efficiently solving a set of linear equations. To reduce the computational burden of the LSSVR for function estimation, the regression model in this study uses a quadratic loss function.

The least squares version of the SVM classifier is obtained by reformulating the minimization problem as:

A close up of a logo

Description generated with very high confidence (5.2)

For the kernel function *K*() one typically has the Radial Basis Function:

A close up of a logo

Description generated with very high confidence (5.3)

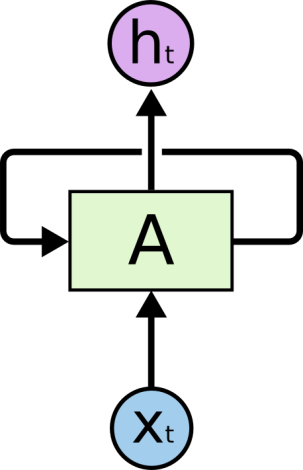
The LSSVR involves equality instead of inequality constraints and works with a least squares objective function. The LSSVR approach considerably reduces computational complexity and increases efficiency compared to standard SVM. LSSVR solves linear equations instead of a quadratic programming problem.

**Recurrent Neural Networks**

Humans don’t start their thinking from scratch every second. As you read this essay, you understand each word based on your understanding of previous words. You don’t throw everything away and start thinking from scratch again. Your thoughts have persistence.

Traditional neural networks can’t do this, and it seems like a major shortcoming. For example, imagine you want to classify what kind of event is happening at every point in a movie. It’s unclear how a traditional neural network could use its reasoning about previous events in the film to inform later ones.

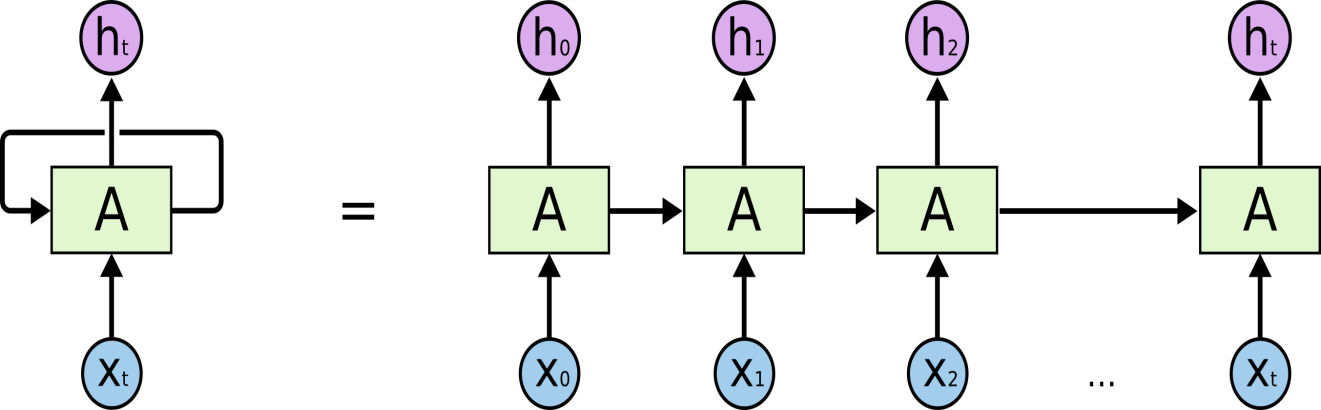
Recurrent neural networks address this issue. They are networks with loops in them, allowing information to persist.



**Recurrent Neural Networks have loops.**

In the above diagram, a chunk of neural network, AA, looks at some input xtxt and outputs a value htht. A loop allows information to be passed from one step of the network to the next.

These loops make recurrent neural networks seem kind of mysterious. However, if you think a bit more, it turns out that they aren’t all that different than a normal neural network. A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. Consider what happens if we unroll the loop:



**An unrolled recurrent neural network.**

This chain-like nature reveals that recurrent neural networks are intimately related to sequences and lists. They’re the natural architecture of neural network to use for such data.

And they certainly are used! In the last few years, there have been incredible success applying RNNs to a variety of problems: speech recognition, language modeling, translation, image captioning… The list goes on. I’ll leave discussion of the amazing feats one can achieve with RNNs to Andrej Karpathy’s excellent blog post, [The Unreasonable Effectiveness of Recurrent Neural Networks](http://karpathy.github.io/2015/05/21/rnn-effectiveness/). But they really are pretty amazing.

Essential to these successes is the use of “LSTMs,” a very special kind of recurrent neural network which works, for many tasks, much much better than the standard version. Almost all exciting results based on recurrent neural networks are achieved with them. It’s these LSTMs that this essay will explore.

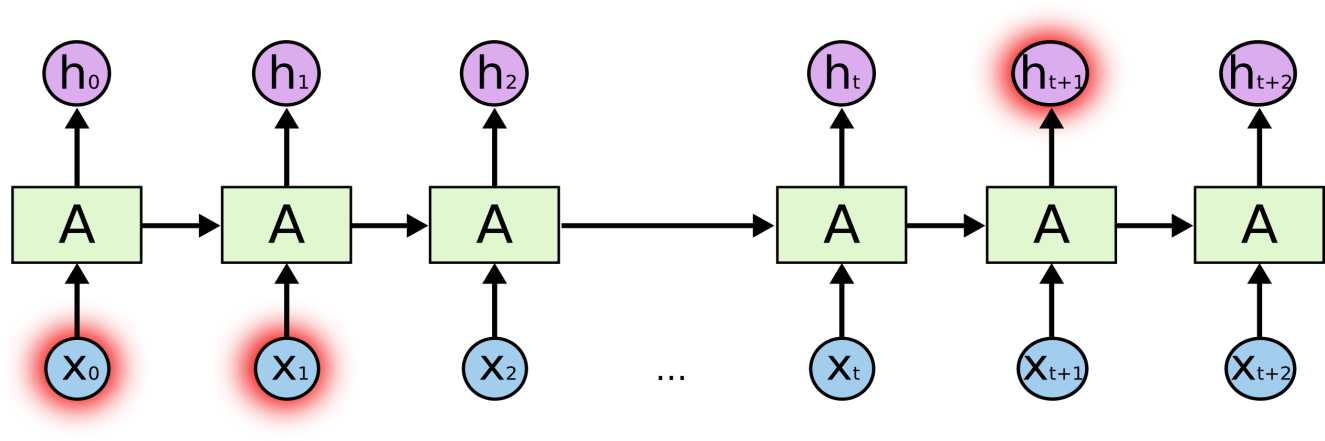
**The Problem of Long Term Dependencies**

One of the appeals of RNNs is the idea that they might be able to connect previous information to the present task, such as using previous video frames might inform the understanding of the present frame. If RNNs could do this, they’d be extremely useful. But can they? It depends.

Sometimes, we only need to look at recent information to perform the present task. For example, consider a language model trying to predict the next word based on the previous ones. If we are trying to predict the last word in “the clouds are in the *sky*,” we don’t need any further context – it’s pretty obvious the next word is going to be sky. In such cases, where the gap between the relevant information and the place that it’s needed is small, RNNs can learn to use the past information.

But there are also cases where we need more context. Consider trying to predict the last word in the text “I grew up in France… I speak fluent *French*.” Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of France, from further back. It’s entirely possible for the gap between the relevant information and the point where it is needed to become very large.

Unfortunately, as that gap grows, RNNs become unable to learn to connect the information.



In theory, RNNs are absolutely capable of handling such “long-term dependencies.” A human could carefully pick parameters for them to solve toy problems of this form. Sadly, in practice, RNNs don’t seem to be able to learn them. The problem was explored in depth by [Hochreiter (1991) [German]](http://people.idsia.ch/~juergen/SeppHochreiter1991ThesisAdvisorSchmidhuber.pdf) and [Bengio, et al. (1994)](http://www-dsi.ing.unifi.it/~paolo/ps/tnn-94-gradient.pdf), who found some pretty fundamental reasons why it might be difficult.

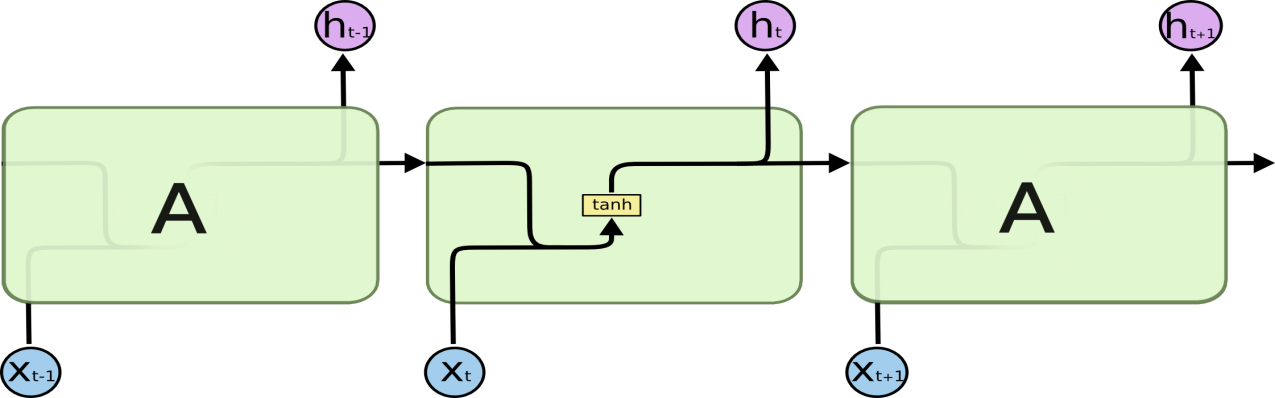
Thankfully, LSTMs don’t have this problem!

**LSTM Networks**

Long Short Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies. They were introduced by [Hochreiter & Schmidhuber (1997)](http://www.bioinf.jku.at/publications/older/2604.pdf), and were refined and popularized by many people in following work.[1](http://colah.github.io/posts/2015-08-Understanding-LSTMs/#fn1) They work tremendously well on a large variety of problems, and are now widely used.

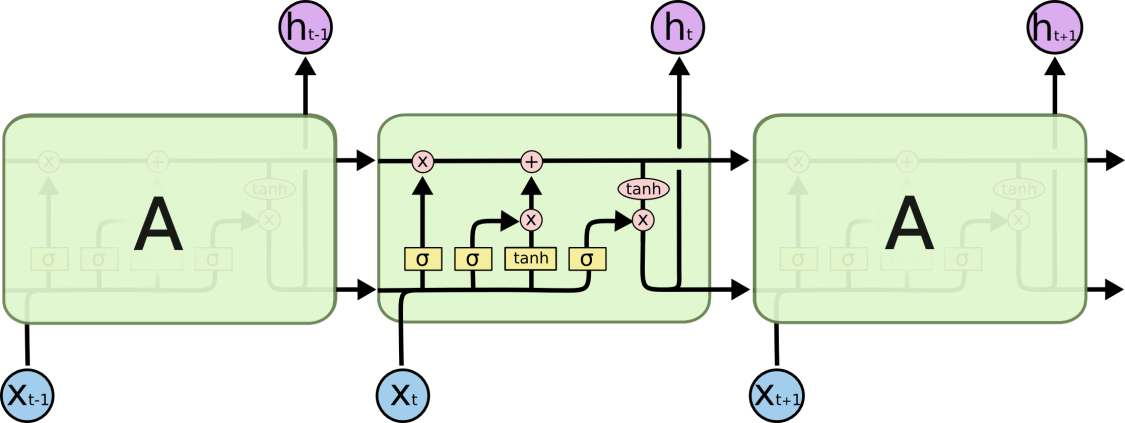
LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn!

All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.



**The repeating module in a standard RNN contains a single layer.**

LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.



**The repeating module in an LSTM contains four interacting layers.**

Don’t worry about the details of what’s going on. We’ll walk through the LSTM diagram step by step later. For now, let’s just try to get comfortable with the notation we’ll be using.



In the above diagram, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denote its content being copied and the copies going to different locations.

**The Core Idea Behind LSTMs**

The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.

The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It’s very easy for information to just flow along it unchanged.



The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.

Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.



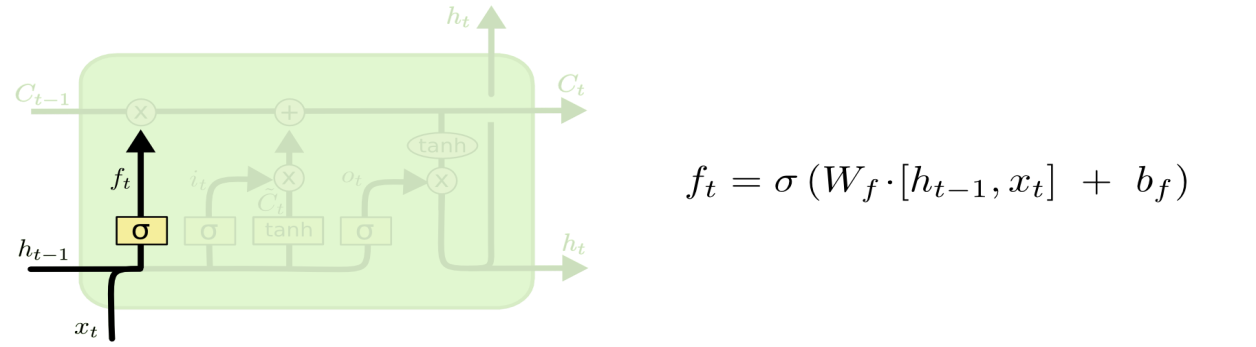
The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means “let nothing through,” while a value of one means “let everything through!”

An LSTM has three of these gates, to protect and control the cell state.

**Step-by-Step LSTM Walk Through**

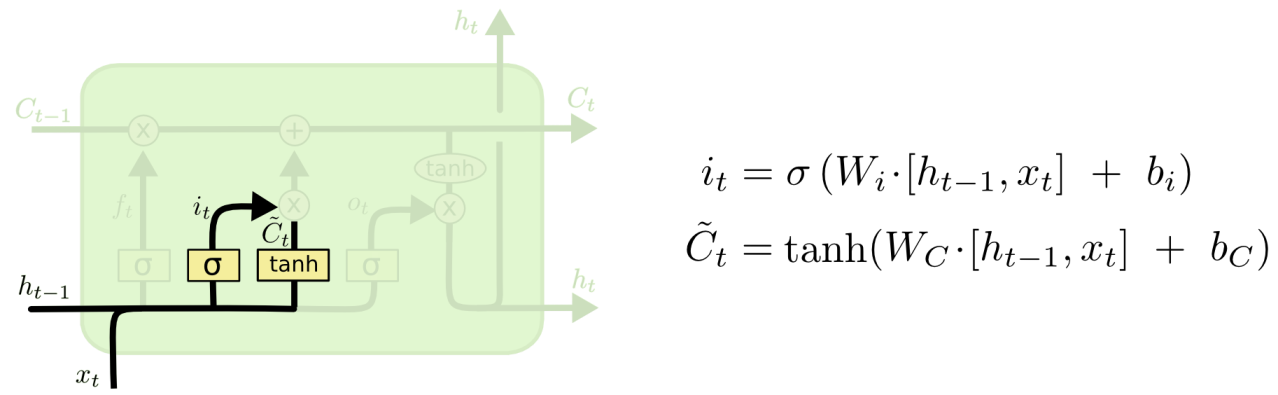
The first step in our LSTM is to decide what information we’re going to throw away from the cell state. This decision is made by a sigmoid layer called the “forget gate layer.” It looks at ht−1ht−1 and xtxt, and outputs a number between 00 and 11 for each number in the cell state Ct−1Ct−1. A 11represents “completely keep this” while a 00 represents “completely get rid of this.”

Let’s go back to our example of a language model trying to predict the next word based on all the previous ones. In such a problem, the cell state might include the gender of the present subject, so that the correct pronouns can be used. When we see a new subject, we want to forget the gender of the old subject.



The next step is to decide what new information we’re going to store in the cell state. This has two parts. First, a sigmoid layer called the “input gate layer” decides which values we’ll update. Next, a tanh layer creates a vector of new candidate values, C~tC~t, that could be added to the state. In the next step, we’ll combine these two to create an update to the state.

In the example of our language model, we’d want to add the gender of the new subject to the cell state, to replace the old one we’re forgetting.



It’s now time to update the old cell state, Ct−1Ct−1, into the new cell state CtCt. The previous steps already decided what to do, we just need to actually do it.

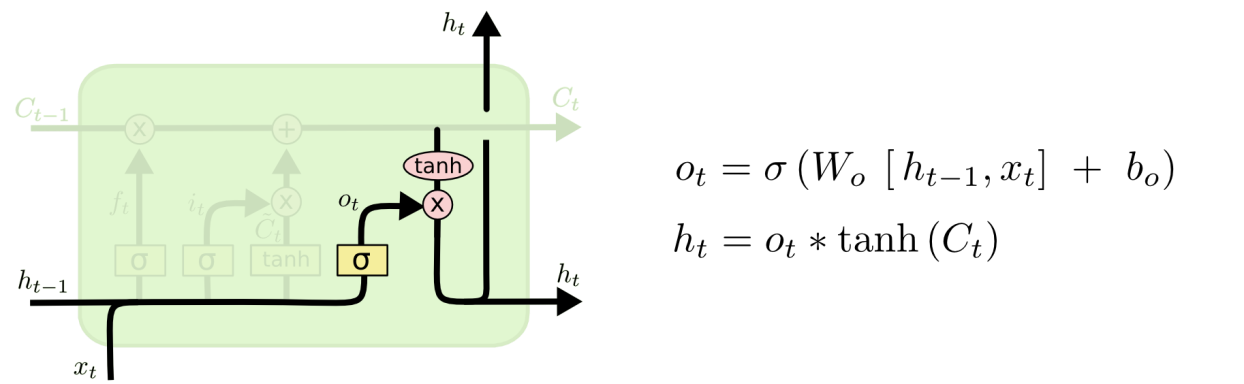
We multiply the old state by ftft, forgetting the things we decided to forget earlier. Then we add it∗C~tit∗C~t. This is the new candidate values, scaled by how much we decided to update each state value.

In the case of the language model, this is where we’d actually drop the information about the old subject’s gender and add the new information, as we decided in the previous steps.



Finally, we need to decide what we’re going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we’re going to output. Then, we put the cell state through tanhtanh (to push the values to be between −1−1 and 11) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

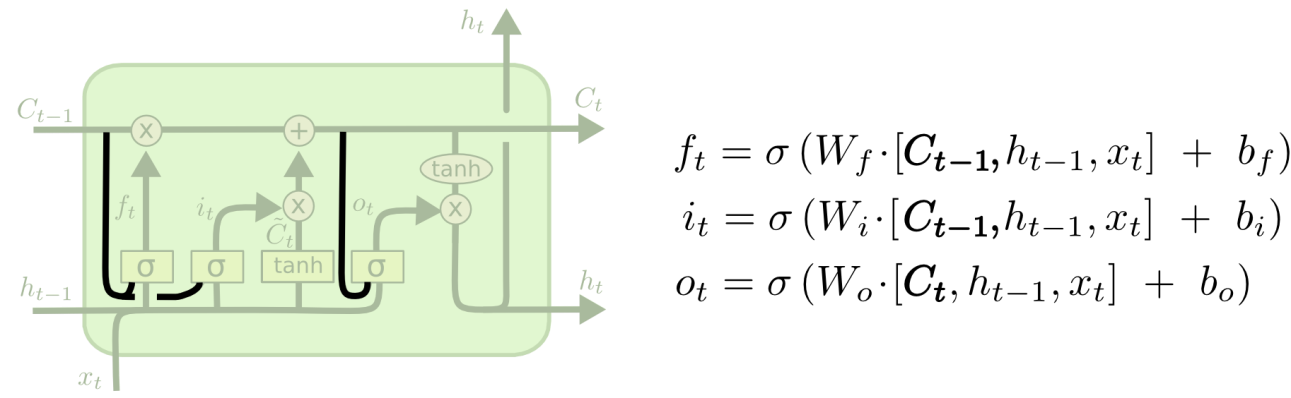
For the language model example, since it just saw a subject, it might want to output information relevant to a verb, in case that’s what is coming next. For example, it might output whether the subject is singular or plural, so that we know what form a verb should be conjugated into if that’s what follows next.



**Variants on Long Short Memory**

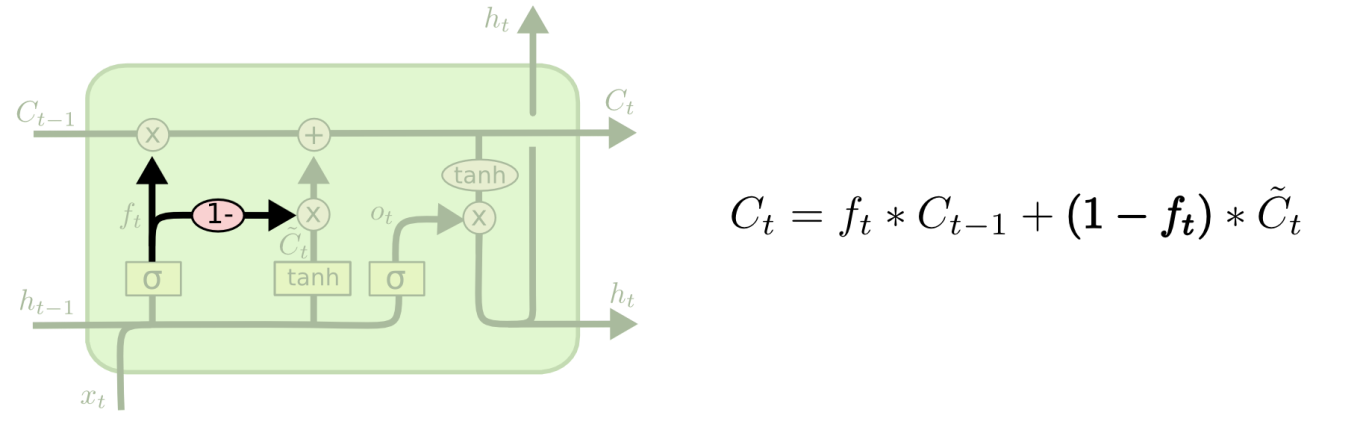
What I’ve described so far is a pretty normal LSTM. But not all LSTMs are the same as the above. In fact, it seems like almost every paper involving LSTMs uses a slightly different version. The differences are minor, but it’s worth mentioning some of them.

One popular LSTM variant, introduced by [Gers & Schmidhuber (2000)](ftp://ftp.idsia.ch/pub/juergen/TimeCount-IJCNN2000.pdf), is adding “peephole connections.” This means that we let the gate layers look at the cell state.



The above diagram adds peepholes to all the gates, but many papers will give some peepholes and not others.

Another variation is to use coupled forget and input gates. Instead of separately deciding what to forget and what we should add new information to, we make those decisions together. We only forget when we’re going to input something in its place. We only input new values to the state when we forget something older.



A slightly more dramatic variation on the LSTM is the Gated Recurrent Unit, or GRU, introduced by [Cho, et al. (2014)](http://arxiv.org/pdf/1406.1078v3.pdf). It combines the forget and input gates into a single “update gate.” It also merges the cell state and hidden state, and makes some other changes. The resulting model is simpler than standard LSTM models, and has been growing increasingly popular.



These are only a few of the most notable LSTM variants. There are lots of others, like Depth Gated RNNs by [Yao, et al. (2015)](http://arxiv.org/pdf/1508.03790v2.pdf). There’s also some completely different approach to tackling long-term dependencies, like Clockwork RNNs by [Koutnik, et al. (2014)](http://arxiv.org/pdf/1402.3511v1.pdf).

Which of these variants is best? Do the differences matter? [Greff, et al. (2015)](http://arxiv.org/pdf/1503.04069.pdf) do a nice comparison of popular variants, finding that they’re all about the same. [Jozefowicz, et al. (2015)](http://jmlr.org/proceedings/papers/v37/jozefowicz15.pdf)tested more than ten thousand RNN architectures, finding some that worked better than LSTMs on certain tasks.

**5.2 Implementation**

|  |
| --- |
| **Algorithm : StockPrediction**  **Input :** *COMP*, *D\_RANGE, N\_PRED* [company, date range, n-day predictions]  **Output :** A vector of predicted prices and graph, RESULTS   1. *data* ← fetch stock for COMP in date range D\_RANGE 2. ***plot****(data)* 3. *train\_data* ← ***slidingWindow***(*data*) *//perform sliding window operation on data* 4. RESULTS ← 0 *//set accuracy vector to zeroes* 5. **for each** day**in** *N\_PRED* **:**   *//pass the training data to LSSVM*   * + 1. *model* ← ***LSSVM***(*train*\_*data*)   *//predict the price given model and day*   * + 1. *pred* ← ***predict***(*model*,*day*)   *//removing last item and adding predicted value*   * + 1. remove first item from *train\_data*     2. *train\_data* ← add *pred* to *train\_data*   *//Last In, First Out.*   * + 1. RESULTS ← add *pred* to RESULTS  1. **end for** 2. ***print***(RESULTS) 3. ***plot***(RESULTS) 4. **return** |

Algorithm 5.2 Overall Implementation

**CHAPTER 6**

**TESTING**

Testing is a critical element which assures quality and effectiveness of the proposed system in (satisfying) meeting its objectives. Testing is done at various stages in the System designing and implementation process with an objective of developing a transparent, flexible and secured system. Testing is an integral part of software development. Testing process, in a way certifies, whether the product, that is developed, complies with the standards, that it was designed to. Testing process involves building of test cases, against which, the product has to be tested.

* 1. **Test Objectives**
* Testing is a process of executing a program with the intent of finding an error.
* A good case is one that has a high probability of finding an undiscovered error.
* A successful test is one that uncovers a yet undiscovered error. If testing is conducted successfully (according to the objectives) it will uncover errors in the software.
* Testing can’t show the absences of defects are present. It can only show that software defects are present.
  1. **Testing Principles**

Before applying methods to design effective test cases, a software engineer must understand the basic principle that guides software testing. All the tests should be traceable to customer requirements.

* 1. **Testing design**

Any engineering product can be tested in one of two ways:

* White Box Testing
* Black Box Testing

**6.3.1 White Box Testing**

This testing is also called as glass box testing. Inthis testing, by knowing the specified function that a product has been designed to perform test can be conducted that demonstrates each function is fully operation at the same time searching for errors in each function. It is a test case design method that uses the control structure of the procedural design to derive test cases.

**6.3.2 Black Box Testing**

Inthis testing by knowing the internal operation of a product, tests can be conducted to ensure that “all gears mesh”, that is the internal operation performs according to specification and all internal components have been adequately exercised. It fundamentally focuses on the functional requirements of the software. The steps involved in black box test case design are:

* Graph based testing methods
* Equivalence partitioning
* Boundary value analysis
* Comparison testing

**6.4 Testing Strategies**

A software testing strategy provides a road map for the software developer. Testing is a set of activities that can be planned in advanced and conducted systematically. For this reason, a template for software testing a set of steps into which we can place specific test case design methods should be defined for software engineering process.

**Any software testing strategy should have the following characteristics:**

* Testing begins at the module level and works outward toward the integration of the entire computer-based system.
* Different testing techniques are appropriate at different points in time.
* The developer of the software and an independent test group conducts testing.
* Testing and debugging are different activities but debugging must be accommodated in any testing strategy.

**6.5 Levels of Testing**

Testing can be done in different levels of SDLC. They are:

* Unit Test
* Integration Test
* Functional Test

**6.5.1 Unit Testing**

The first level of testing is called unit testing. Unit testing verifies on the smallest unit of software designs-the module. The unit test is always white box oriented. In this, different modules are tested against the specifications produced during design for the modules. Unit testing is essentially for verification of the code produced during the coding phase, and hence the goal is to test the internal logic of the modules. It is typically done by the programmer of the module.

These types of tests are usually written by developers as they work on code (white-box style), to ensure that the specific function is working as expected. One function might have multiple tests, to catch corner cases or other branches in the code. Unit testing alone cannot verify the functionality of a piece of software, but rather is used to ensure that the building blocks of the software work independently from each other.

Unit testing is a software development process that involves synchronized application of a broad spectrum of defect prevention and detection strategies in order to reduce software development risks, time, and costs. It is performed by the software developer or engineer during the construction phase of the software development lifecycle. Rather than replace traditional quality analysis focuses, it augments it. Unit testing aims to eliminate construction errors before code is promoted to quality analysis; this strategy is intended to increase the quality of the resulting software as well as the efficiency of the overall development and quality analysis process.

Due to its close association with coding, the coding phase is frequently called “coding and unit testing.” The unit test can be conducted in parallel for multiple modules. We try testing various stock markets and predict the stock for each.

The Test cases in unit testing are as follows:

|  |  |
| --- | --- |
| Test Case ID | Unit Test Case 1 |
| Description | IBM Stock Dataset |
| Input | **Date:** 01-01-2017 – 01-01-2018  **Comp:** IBM  **Days:** 90 |
| Expected output | Prediction Successful |
| Actual Result/Remarks | Got the expected output |
| Passed (?) | Yes |

Table 6.1: Unit Test Case 1

In the next test case, we’ll use the same date range but with Coca Cola stock

|  |  |
| --- | --- |
| Test Case ID | Unit Test Case 2 |
| Description | Coca Cola Stock Dataset |
| Input | **Date:** 01-01-2017 – 01-01-2018  **Comp:** Coca Cola  **Days:** 90 |
| Expected output | Prediction Successful |
| Actual Result/Remarks | Got the expected output |
| Passed (?) | Yes |

Table 6.2: Unit Test Case 2

**6.5.2 Integration Testing**

The second level of testing is called integration testing. Integration testing is a systematic technique for constructing the program structure while conducting tests to uncover errors associated with interfacing. In this, many tested modules are combined into subsystems, which are then tested. The goal here is to see if all the modules can be integrated properly.

There are three types of integration testing:

* *Top-Down Integration*: Top down integration is an incremental approach to construction of program structures. Modules are integrated by moving downwards throw the control hierarchy beginning with the main control module.
* *Bottom-Up Integration*: Bottom up integration as its name implies, begins Construction and testing with automatic modules.
* *Regression Testing*: In this contest of an integration test strategy, regression testing is the re execution of some subset of test that have already been conducted to ensure that changes have not propagated unintended side effects.

**6.5.3 Functional test**

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals. Functional testing is centered on the following items:

|  |  |
| --- | --- |
| Valid Input | Identified classes of valid input must be accepted. |
| Invalid Input | Identified classes of invalid input must be rejected. |
| Functions | Identified functions must be exercised. |
| Output | Identified classes of application outputs must be exercised. |

Table 6.3: Functional Testing items

**Systems/Procedures*:*** Interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

**6.5.4 Validation testing**

At the culmination of integration testing, software is completely assembled as a package; interfacing errors have been covered and corrected, and final series of software tests-validating testing may begin. Validation can be defined in many ways, but a simple definition is that validation succeeds when software functions in a manner that can be reasonably expected by customers. Reasonable expectation is defined in the software requirement specification- a document that describes all user visible attributes of the software. The specification contains a section title “validation criteria”. Information contained in that section forms the basis for validation testing approach

**6.5.5 Alpha testing**

It is virtually impossible for a software developer to foresee how the customer will really use a program. Instructions for use may be misinterpreted; strange combination of data may be regularly used and output that seemed clear to the tester may be unintelligible to a user in field.

When custom software is built for one customer, a series of acceptance tests are conducted to enable the customer to validate all requirements by the end user rather than system developer and acceptable test can range from an informal “test drive” to a planned and systematically executed series of tests. In fact, acceptance testing can be conducted over a period of weeks or months, thereby uncovering cumulative errors that might degrade the system over time. If software is developed as a product to be used by many customers, it is impractical to perform formal acceptance test with each one. Most software product builders use a process called alpha and beta testing to uncover errors that only the end user seems able to find.

A customer conducts the alpha test at the developer’s site. The software is used in a natural setting with the developer “Looking over the shoulder” of the user and recording errors and usage problems. Alpha tests are conducted in controlled environment.

**6.5.6 Beta testing**

The beta test is conducted at one or more customer sites by the end user of the software. Unlike alpha testing, the developer is generally not present. Therefore, the beta test is a “live” application of the software in an environment that cannot be controlled by the developer. The customer records all problems that are encountered during beta testing and reports these to the developer at regular intervals. As a result of problems reported during beta test, the software developer makes modification and then prepares for release of the software product to the entire customer base.

**6.5.7 System Testing and Acceptance Testing**

System testing is actually a series of different tests whose primary purpose is to fully exercise the computer-based system. Include recovery testing during crashes, security testing for unauthorized user, etc.

Acceptance testing is sometimes performed with realistic data of the client to demonstrate that the software is working satisfactorily. This testing in FDAC focuses on the external behavior of the system.

**CHAPTER 7**

**PERFORMANCE ANALYSIS**

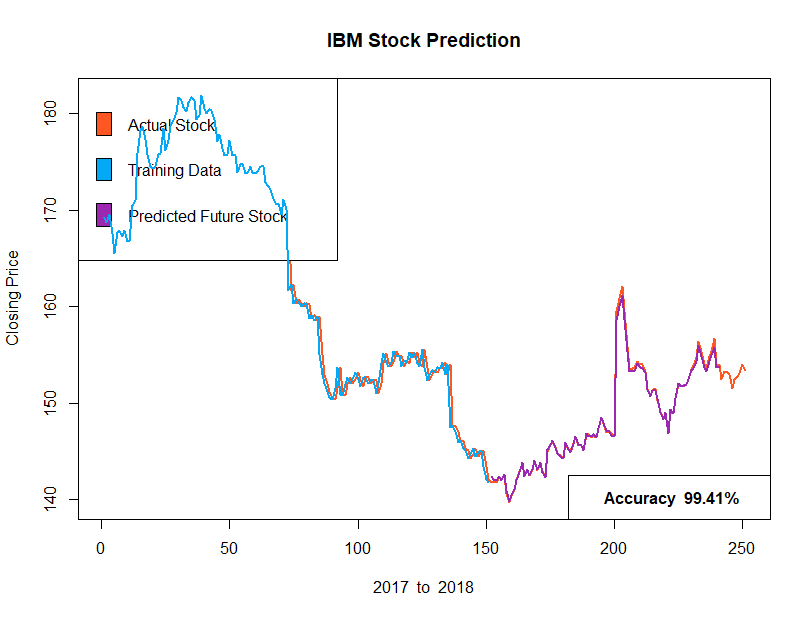
Now that the system has been set up and the results have been obtained, we can do a performance analysis to observe the limits, reliability and accuracy of the proposed system. To achieve this, we provide the following inputs for various stock datasets available online.

|  |  |
| --- | --- |
| **Date Range** | 01-01-2017 to 01-01-2018 |
| **Companies** | Microsoft, IBM, 3M, Coca-Cola, McDonald’s, Intel, Nike, Apple, Cisco, Disney |
| **Forward Days** | **90** Days |

Table 7.1 Input provided for performance analysis

We shall perform the analysis of all the companies and plot a performance graph. The analysis for the first two companies have been given in this literature.

**7.1 IBM Stock Prediction (IBM)**

Figure 7.1 IBM Stock Prediction for 90 Days

As we can see from the above figure, we see the overall stock and also the predicted values. The system appears to perform extremely well with a 99.41% accuracy. The purple line almost perfectly follows the actual orange line.

During each prediction, the accuracy values are calculated and plotted separately as shown in Figure 7.2

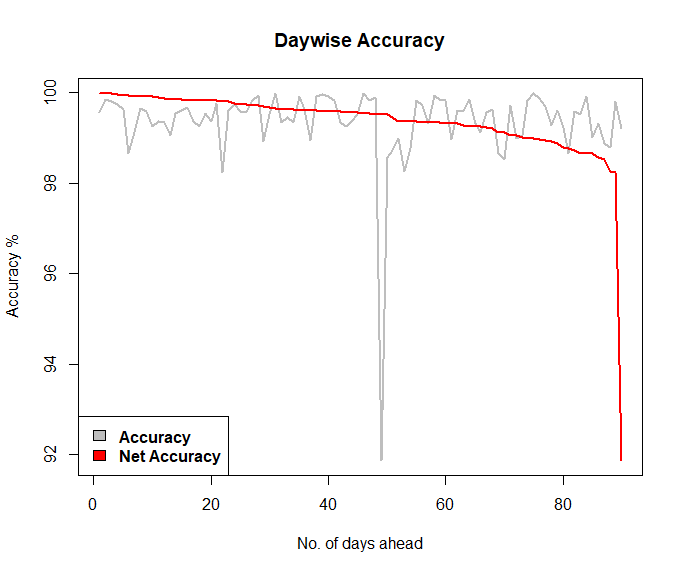


Figure 7.2 IBM: Day wise Accuracy

The above graph shows the day-wise accuracy of the predictions. The accuracy of the predicted and actual values was computed and plotted.

The prediction accuracy day-wise varies sometimes, and we plot them as the grey line. After sorting these values, we plot a red line to show the overall decline in accuracy.

The decline in accuracy over the days can be explained through a mean predictive accuracy graph. We can show this for the given IBM stock data.

The predictive overall performance of the entire prediction is shown in Figure 7.3

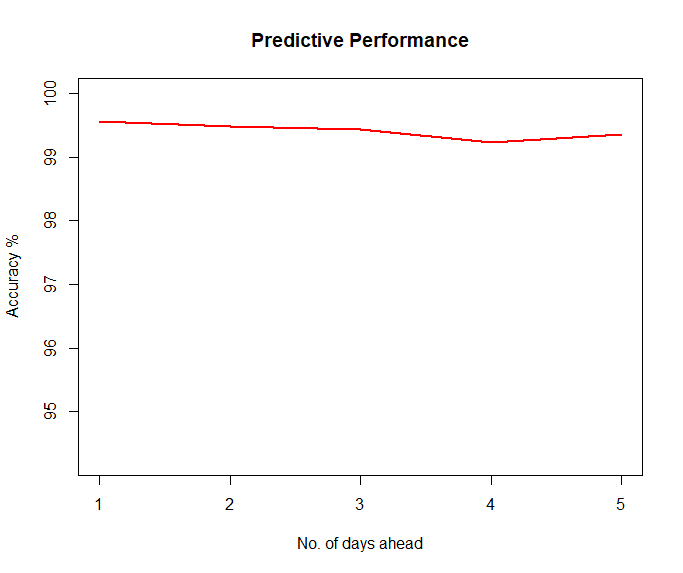


Figure 7.3 IBM: Predictive Performance

**7.2 CISCO Stock Prediction (CSCO)**

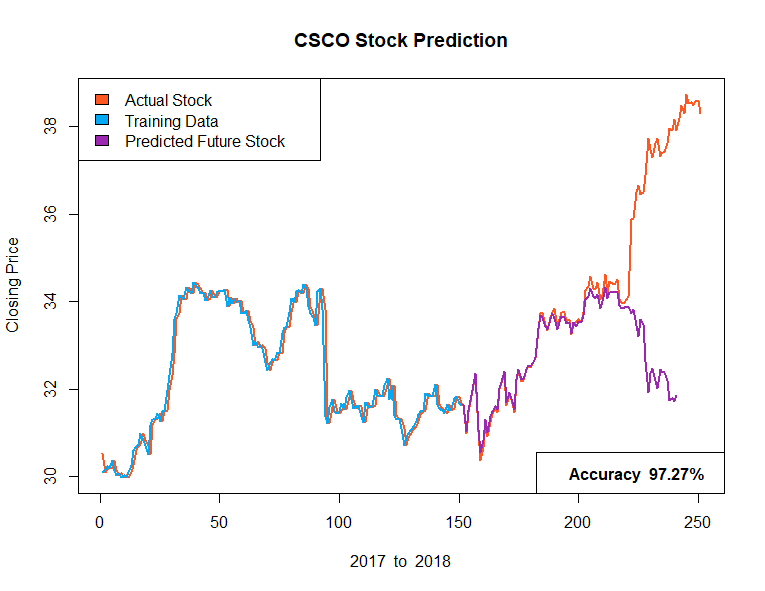


Figure 7.4 Cisco: Stock prediction for 90 days

In the Figure 7.4, we can see that the predicted stock declines steeply after a certain number of days. This sudden decline can be explained due to the external factors that influence stock market. These situations cannot be predicted by a learning system using past data. We can conclude that the performance of the proposed system is consistent only for a range of future days. We shall now see the performance decline in the prediction for CISCO by plotting a Mean Predictive Performance graph.

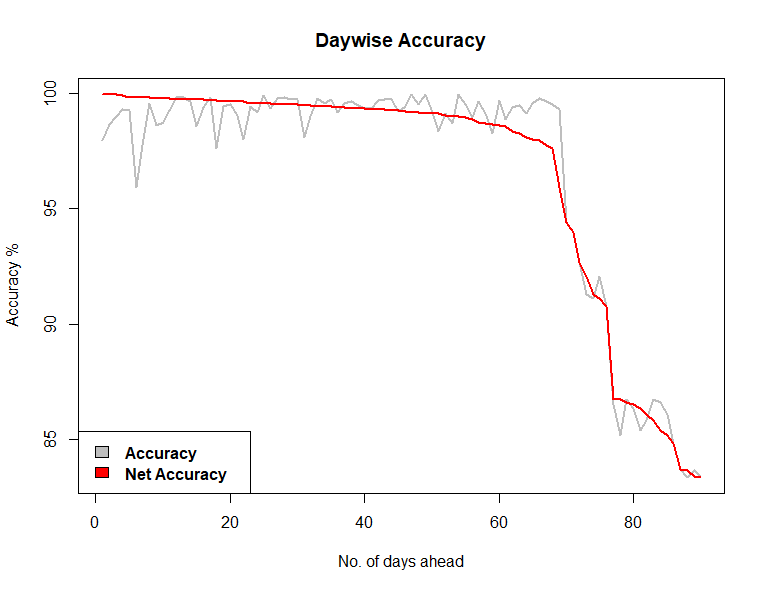


Figure 7.5 Cisco: Day wise accuracy

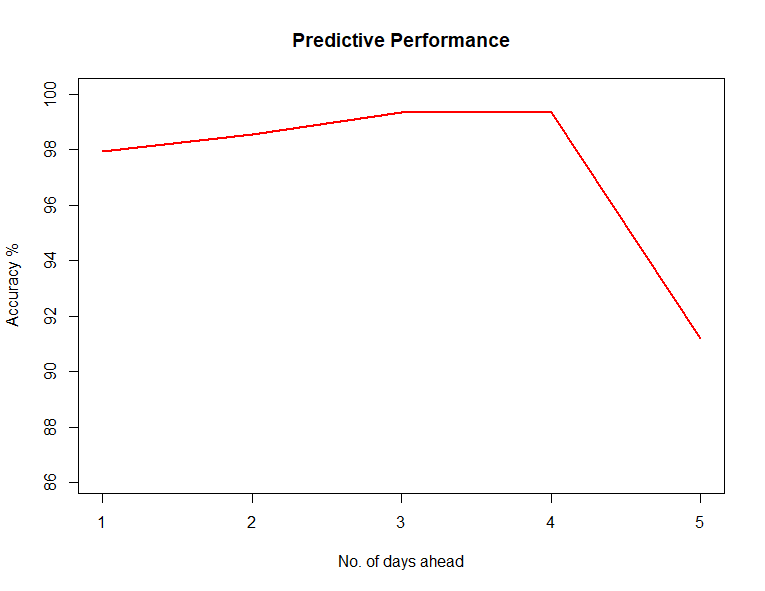
The following graph shows the overall performance over 90 days 

Figure 7.6 Cisco: Predictive Performance

**7.3 Multiple Stock Performance**

Similarly, we now do a complete performance over 10 datasets for a 90-day forward prediction. The resultant graph is as follows:

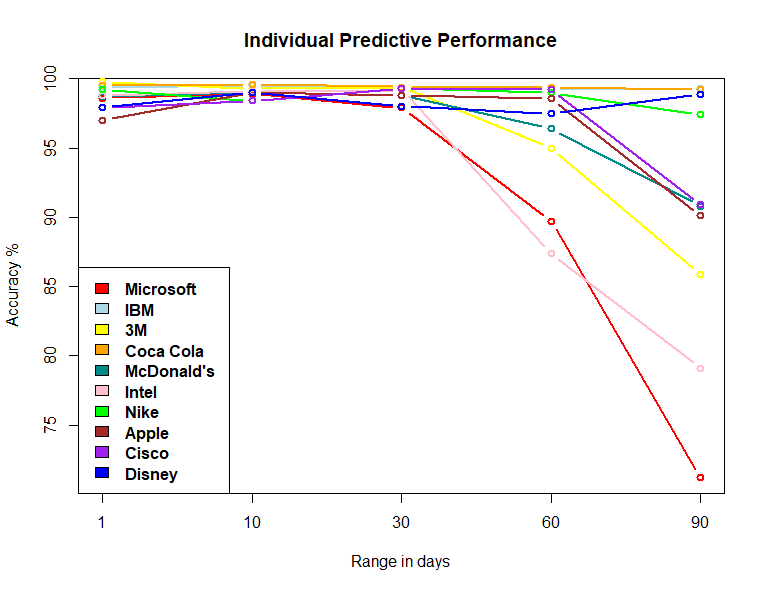


Figure 7.7 Individual Predictive Performance

**7.3 Overall System Performance**

We can now plot the universal performance of the proposed system by averaging the performance for all systems. The plotted graph is as shown in Figure 7.8.

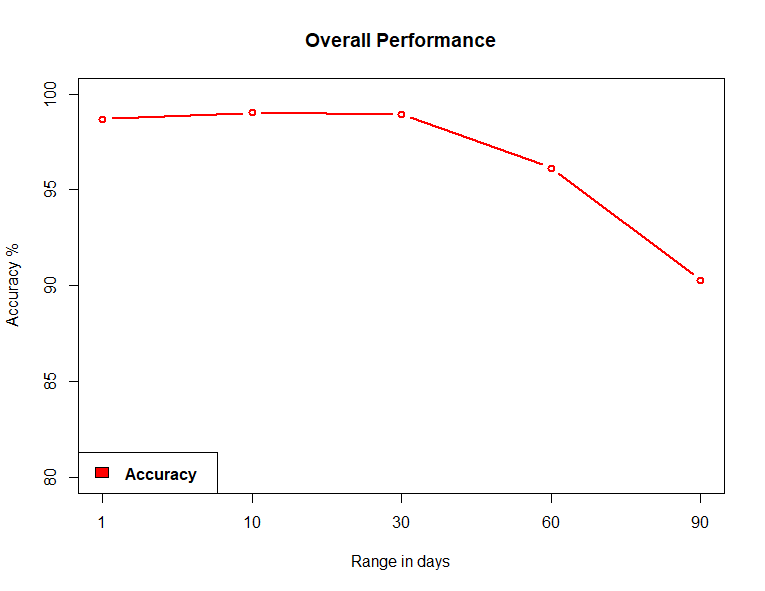


Figure 7.8 Overall System Performance

**CHAPTER 8**

**RESULTS & DISCUSSIONS**

We now observe the results obtained by the proposed system for each test case. When the user initializes the program, the following screen is displayed and the user enters the choice 3, for IBM in the date range 01-01-2017 to 01-01-2018 (which is the default range). The user predicts stock for the next 10 days. The input is given as follows:

|  |  |
| --- | --- |
| Welcome to Stock Market Prediction  Choose the dataset you wish to predict the stock for  1. Microsoft 6. McDonald’s  2. Apple 7. Intel Corp.  3. IBM 8. 3M  4. Nike 9. CocaCola  5. Cisco 10. Walt Disney   |  | | --- | | Enter your choice : 3  Do you wish to use default date range? (y|n) y  Fetching data for IBM  Predict stock for how many days in advance? (1 - 99 ) : 10 | |

Table 8.1 Requesting input for prediction

Once these inputs have been given, the system fetches the data from Quandl or any other online source and feeds this data to the LSSVR and trains the model. When the training is complete, the predicted values are displayed for the next 10 days as follows:

|  |
| --- |
| DAY DATE PREDICTED PRICE  Day 01 2017-08-09 142.5203  Day 02 2017-08-10 141.5793  Day 03 2017-08-11 142.1064  Day 04 2017-08-14 142.6604  Day 05 2017-08-15 142.1869  Day 06 2017-08-16 142.5381  Day 07 2017-08-17 140.6312  Day 08 2017-08-18 141.1423  Day 09 2017-08-21 140.3994  Day 10 2017-08-22 141.1157 |

Table 8.2 Predicted closing stock prices

After the above results are displayed, we also obtain the basic statistics and summary of the prediction for any given input set.

|  |
| --- |
| -----------------------Prediction Summary--------------------  BASIC STATS  No. of days predicted 10  Training data (days) 151  Max. pred. accuracy 99.85%  Min. pred. accuracy 98.71%  Mean pred. accuracy 99.49%  Predictive Accuracy 99.5% |

Table 8.3 Basic Summary Statistics

Now that we have the predicted values, we can later compare them to the actual closing stock prices by using the *my\_summary()* function.

|  |
| --- |
| Date Actual Predicted  1 2017-08-09 141.84 142.5203  2 2017-08-10 141.84 141.5793  3 2017-08-11 142.32 142.1064  4 2017-08-14 142.07 142.6604  5 2017-08-15 142.50 142.1869  6 2017-08-16 140.70 142.5381  7 2017-08-17 139.70 140.6312  8 2017-08-18 140.33 141.1423  9 2017-08-21 141.01 140.3994  10 2017-08-22 142.14 141.1157 |

Table 8.4 Results with actual and predicted prices

From the table given above we see the 10-day forward prediction for the given test case, and the actual prices that were observed. The obtained and trained values are plotted on a graph as shown in the next section.

The trained data are in Blue colour and the predicted values are in purple. The actual values of the stock on any given day is given in Orange.

We now obtain the graph that is plotted along with actual, predicted and trained data. In the graph show in Figure x, we see the blue line represents the existing training data that is fed into the LSSVR to train the model. We shall assume that the end of the training set represents the “Current Day” and that the days that follow are the future.

The future closing prices are predicted and plotted as the purple line. The orange line represents the actual closing stock prices on that day. In Figure 8.1, the prediction is encircled in red.

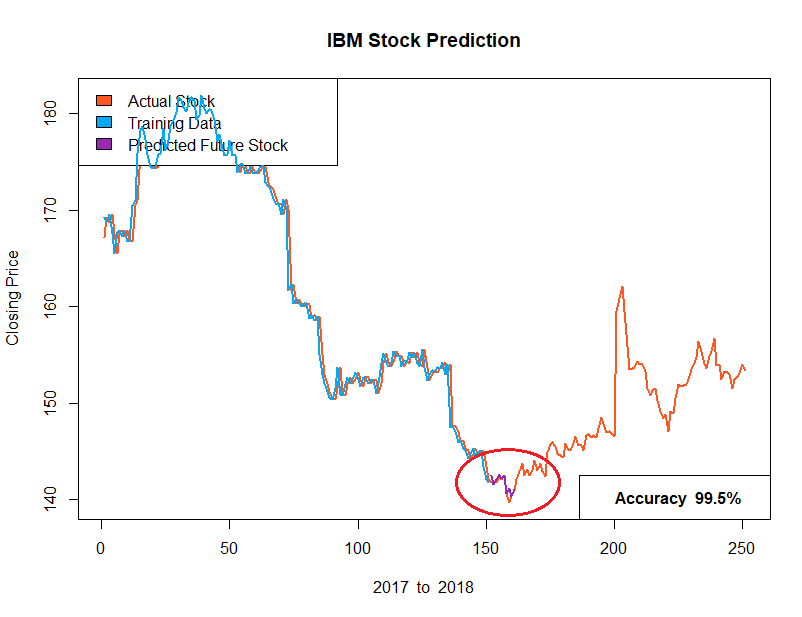


Figure 8.1 IBM Stock Prediction for 10 days

We see, that for 10 days, the mean prediction is given by dividing the actual and predicted values and finding the average. It has been observed for 10 days, the accuracy is around 99.5%.

We shall now do a performance review of the system with multiple dataset chosen at random in the same data range for a forward prediction of up to 90 days.

**CHAPTER 9**

**SCREENSHOTS**

An interface to the proposed system has been prepared and are shown in this section.

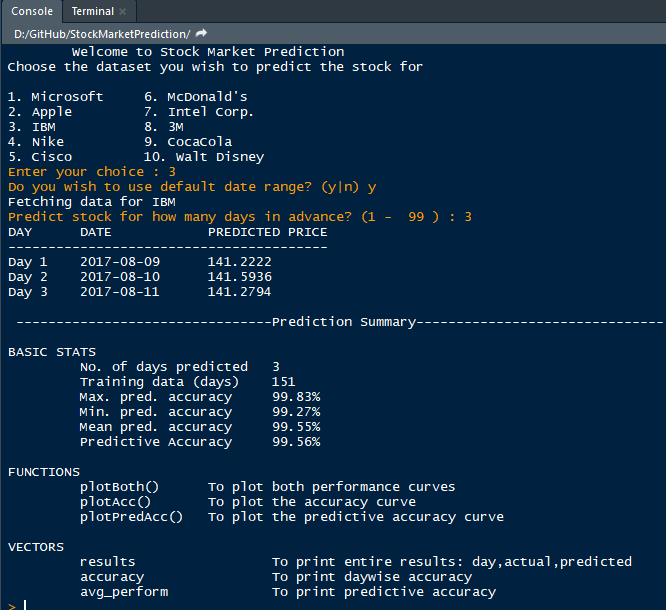


Figure 9.1 Interface of the system

The above screenshot shows the user interface where the user enters the details. The user is free to choose any of the given options for stock after which he is asked for the date range. With a slight modification the user can also enter his own choice for Company ticker to fetch the stock.

A *my\_summary()* function is also provided that displays a summary of statistics, which can be used for further analysis such as plotting and forwarding results

When the program is executed, the graph is displayed side-by side instantly as soon as the stock is fetched and updated with the prediction when the training is complete.

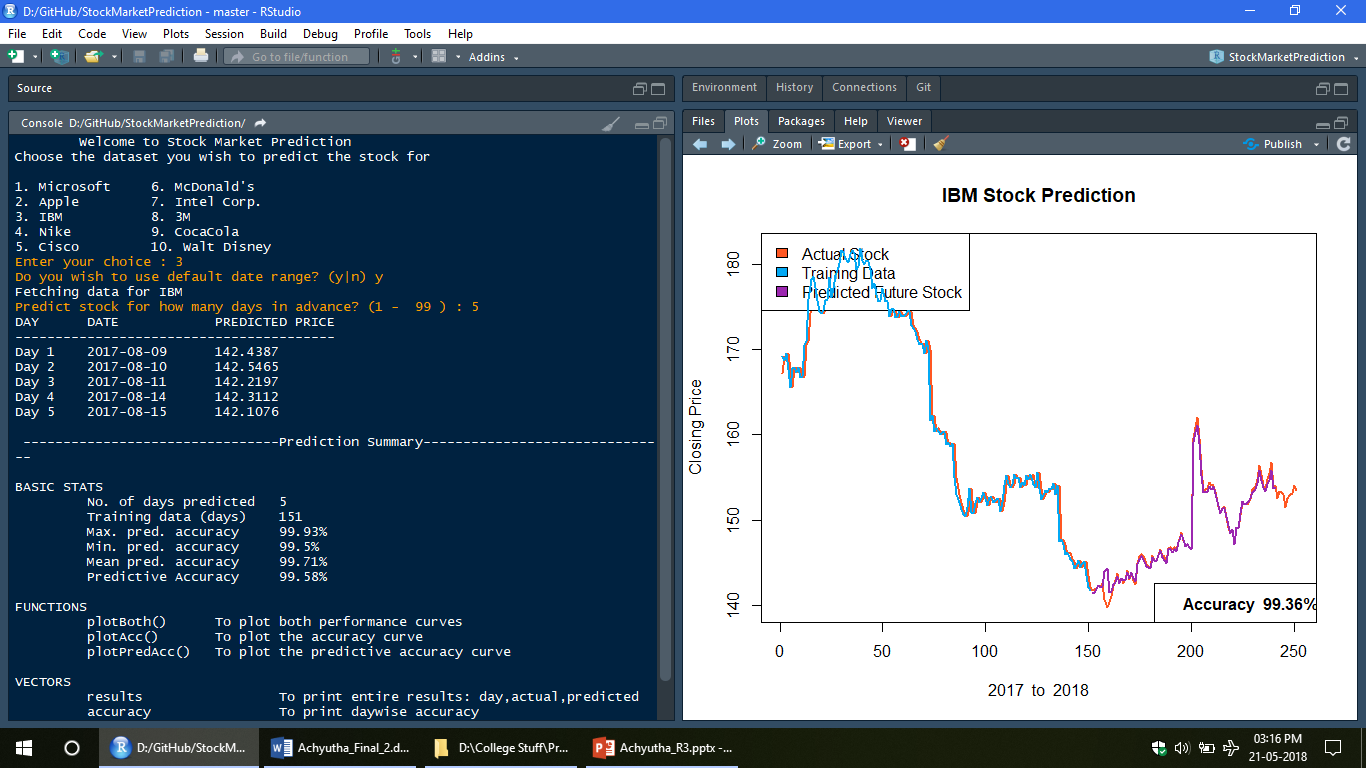


Figure 9.2 The screenshot of system

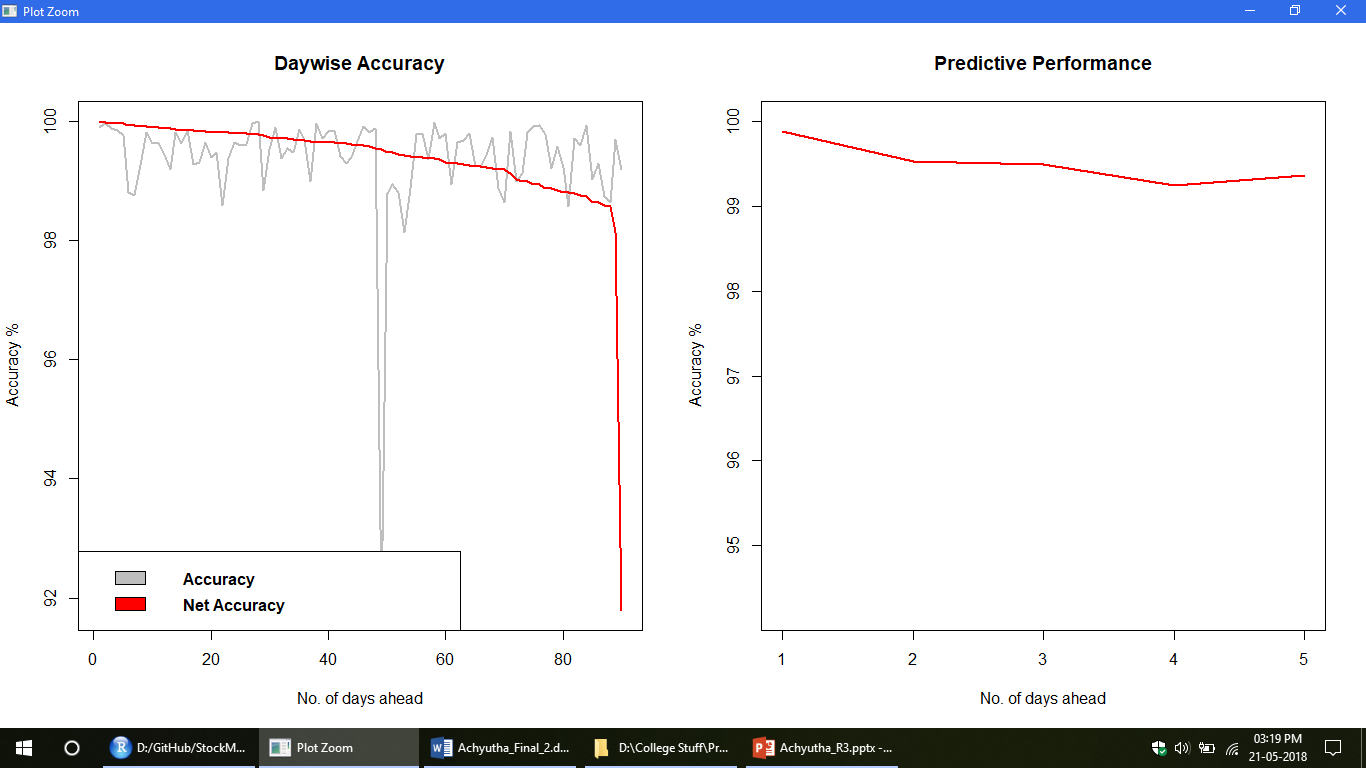


Figure 9.3 Predictive Performance of the system

**Chapter-10**

**CODE**

# Recurrent Neural Network

# Part 1 - Data Preprocessing

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the training set

dataset\_train = pd.read\_csv('Google\_Stock\_Price\_Train.csv')

training\_set = dataset\_train.iloc[:, 1:2].values

# Feature Scaling

from sklearn.preprocessing import MinMaxScaler

sc = MinMaxScaler(feature\_range = (0, 1))

training\_set\_scaled = sc.fit\_transform(training\_set)

# Creating a data structure with 60 timesteps and 1 output

X\_train = []

y\_train = []

for i in range(60, 1258):

X\_train.append(training\_set\_scaled[i-60:i, 0])

y\_train.append(training\_set\_scaled[i, 0])

X\_train, y\_train = np.array(X\_train), np.array(y\_train)

# Reshaping

X\_train = np.reshape(X\_train, (X\_train.shape[0], X\_train.shape[1], 1))

# Part 2 - Building the RNN

# Importing the Keras libraries and packages

from keras.models import Sequential

from keras.layers import Dense

from keras.layers import LSTM

from keras.layers import Dropout

# Initialising the RNN

regressor = Sequential()

# Adding the first LSTM layer and some Dropout regularisation

regressor.add(LSTM(units = 50, return\_sequences = True, input\_shape = (X\_train.shape[1], 1)))

regressor.add(Dropout(0.2))

# Adding a second LSTM layer and some Dropout regularisation

regressor.add(LSTM(units = 50, return\_sequences = True))

regressor.add(Dropout(0.2))

# Adding a third LSTM layer and some Dropout regularisation

regressor.add(LSTM(units = 50, return\_sequences = True))

regressor.add(Dropout(0.2))

# Adding a fourth LSTM layer and some Dropout regularisation

regressor.add(LSTM(units = 50))

regressor.add(Dropout(0.2))

# Adding the output layer

regressor.add(Dense(units = 1))

# Compiling the RNN

regressor.compile(optimizer = 'adam', loss = 'mean\_squared\_error')

# Fitting the RNN to the Training set

regressor.fit(X\_train, y\_train, epochs = 100, batch\_size = 32)

# Part 3 - Making the predictions and visualising the results

# Getting the real stock price of 2017

dataset\_test = pd.read\_csv('Google\_Stock\_Price\_Test.csv')

real\_stock\_price = dataset\_test.iloc[:, 1:2].values

# Getting the predicted stock price of 2017

dataset\_total = pd.concat((dataset\_train['Open'], dataset\_test['Open']), axis = 0)

inputs = dataset\_total[len(dataset\_total) - len(dataset\_test) - 60:].values

inputs = inputs.reshape(-1,1)

inputs = sc.transform(inputs)

X\_test = []

for i in range(60, 80):

X\_test.append(inputs[i-60:i, 0])

X\_test = np.array(X\_test)

X\_test = np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1))

predicted\_stock\_price = regressor.predict(X\_test)

predicted\_stock\_price = sc.inverse\_transform(predicted\_stock\_price)

# Visualising the results

plt.plot(real\_stock\_price, color = 'red', label = 'Real Google Stock Price')

plt.plot(predicted\_stock\_price, color = 'blue', label = 'Predicted Google Stock Price')

plt.title('Google Stock Price Prediction')

plt.xlabel('Time')

plt.ylabel('Google Stock Price')

plt.legend()

plt.show()

**CHAPTER 11**

**FUTURE ENHANCEMENT**

**10.1 Limitations**

The proposed system has a number of limitations that have been observed such as its computational speed, especially with respect to sliding-window validation as the computational cost increases with the number of forward day predictions.

* LSSVR is computationally slower when used along with sliding-window method.
* Each time the window is moved across, the LSSVR must learn for the new dataset
* The LSSVR must be run each time the window slides and thus the window size affects the performance of the system.
* LSSVR uses many parameters that must be handled and taken care of.
* The system is best suited for short term prediction and error rate increases with number of days of prediction

.

**10.2 Future Enhancement**

* The proposed model does not predict well for sudden changes in the trend of stock data.
* This occurs due to external factors and real-world changes affecting the stock market.
* We can overcome this by implementing Sentiment Analysis and Neural Networks to enhance the proposed model.
* We can modify the same system to an online-learning system that adapts in real-time.

**CHAPTER 12**

**CONCLUSION**

Decision to buy or sell a stock is very complicated since many factors can affect stock price. This work presents a novel approach, based on LSSVR and Machine Learning to constructing a stock price forecasting expert system, with the aim of improving forecasting accuracy.

Thus, as we can see in our proposed method, we train the data using existing stock dataset that is available. We use this data to predict and forecast the stock price of n-days into the future.

The average performance of the model decreases with increase in number of days, due to unpredictable changes in trend as noted in the literature’s limitations.

The current system can update its training set as each day passes so as to detect newer trends and behave like an online-learning system that predicts stock in real-time. The intelligent time series prediction system that uses sliding-window metaheuristic optimization is a graphical user interface that can be run as a stand-alone application. The system makes the prediction of stock market values simpler, involving fewer computations, than that using the other method that was mentioned above.

**REFERENCES**

**[1] Jui-Sheng Chou and Thi-Kha Nguyen,** Forward Forecast of Stock Price Using Sliding-window Metaheuristic-optimized Machine Learning Regression, IEEE Transactions on Industrial Informatics, 2018, DOI 10.1109/TII.2018.2794389

**[2] M. Göçken, M. Özçalıcı, A. Boru, and A. T. Dosdoğru**, “Integrating metaheuristics and Artificial Neural Networks for improved stock price prediction,” Expert Systems with Applications, vol. 44, pp. 320-331, 2016.

**[3]** **Y. Bao, Y. Lu, and J. Zhang**, "Forecasting Stock Price by SVMs Regression," Artificial Intelligence: Methodology, Systems, and Applications, C. Bussler and D. Fensel, eds., pp. 295-303, Berlin, Heidelberg: Springer Berlin Heidelberg, 2004

**[4]** **K. Duan, S. S. Keerthi, and A. N. Poo**, “Evaluation of simple performance measures for tuning SVM hyperparameters,” Neurocomputing, vol. 51, pp. 41-59, 2003.

**[5]** **T. Xiong, Y. Bao, and Z. Hu**, “Multiple-output support vector regression with a firefly algorithm for interval-valued stock price index forecasting,” Knowledge-Based Systems, vol. 55, pp. 87-100, 2014.

**[6]** **D. Saini, A. Saxena, and R. C. Bansal**, "Electricity price forecasting by linear regression and SVM." pp. 1-7.

**[7]** **V. N. Vapnik,** The nature of statistical learning theory: Springer-Verlag New York, Inc., 1995

**[8]** **J. A. K. Suykens**, "Nonlinear modelling and support vector machines." pp. 287-294.

**[9] W. Hao, and S. Yu**, "Support Vector Regression for Financial Time Series Forecasting," Knowledge Enterprise: Intelligent Strategies in Product Design, Manufacturing, and Management, K. Wang, G. L. Kovacs, M. Wozny and M. Fang, eds., pp. 825-830, Boston, MA: Springer US, 2006

**[10]** **X.-S. Yang**, Nature-Inspired Metaheuristic Algorithms: Luniver Press, 2008.