VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANA SANGAMA, BELAGAVI-590018

A Project Report
on

"Stock Market Prediction Using Machine Learning"

Submitted in partial fulfilment of the requirements for the VIII Semester, ***Bachelor of Engineering in Computer Science and Engineering***

of Visvesvaraya Technological University, Belagavi

Submitted by:

Dhanush Murali	**(1RN14CS025)**
Koushik R Kirugulige	**(1RN14CS045)**

Under the guidance of

Anjan Kumar K N

Asst Professor

Dept. of CSE

Department of Computer Science and Engineering

RNS Institute of Technology

Channasandra, Dr. Vishnuvaradana Road, Bengaluru-560 098

2017-2018

RNS Institute of Technology

Channasandra, Dr.Vishnuvaradana Road,

Bengaluru-560 098

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

Certified that the project work titled ***"Stock Market Prediction Using Machine Learning"*** has been successfully carried out by **Dhanush Murali**bearing USN 1RN14CS025, **Koushik R Kirugulige** bearing USN 1RN14CS045, bonafide students of **RNS Institute of Technology** in partial fulfilment of the requirements for the award of degree in Bachelor of Engineering in Computer Science and Engineering during academic year2017-2018. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The project report has been approved as it satisfies the academic requirements in respect of project work for the said degree.

Internal Guide:

Anjan Kumar K N
Asst Professor

Dr. M K Venkatesha
Principal

External Viva:

Name of the Examiners **Signature with Date**

1.

2.

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. We present a Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) approach to predict stock market indices.

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 .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 LSTMs and Machine Learning to constructing a stock price forecasting expert system, with the aim of improving forecasting accuracy.

ACKNOWLEDGEMENT

Any achievement, be it scholastic or otherwise does not depend solely on the individual efforts but on the guidance, encouragement and cooperation of intellectuals, elders and friends. A number of personalities, in their own capacities have helped me in carrying out this project work. I would like to take this opportunity to thank them all.

We would like to thank the management of RNS Institute of Technology for their support and encouragement towards completing this project.

We would like to thank **Dr. H N Shivashankar, Director,** for his moral support towards completing my project.

We are grateful to **Dr. M K Venkatesha, Principal,**for his support towards completing my project.

We would like to thank **Dr. G T Raju, Professorand Head of the Department, Computer Science and Engineering,** for his valuable suggestions and expert advice on the project.

We would like to thank **Mr.Anjan Kumar K N,Asst Professor** for his able guidance, encouragement and assistance in completing this project.

We also thank the project coordinators, **Mr.Devaraju B M Professor** and team, for their guidance and support in completing this project.

We are extremely thankful to my parents, classmates, and seniors for being supportive of my endeavours and providing their valuable advice.

We thank all the teaching and non-teaching staff of Department of Computer Science and Engineering RNSIT, Bangalore for their constant support and encouragement.

DhanushMurali(1RN14CS025)

Koushik R Kirugulige (1RN14CS045)

CONTENTS

LIST OF FIGURES

Figure No. **Page No.**

LIST OF TABLES

Chapter 1

INTRODUCTION

Modeling and Forecasting of the financial market has been an attractive topic to scholars and researchers from various academic fields. Financial market is an abstract concept where financial commodities such as stocks, bonds and precious metals transactions happen between buyers and sellers. In the present scenario of financial market world, especially in the stock market, forecasting the trend or the price of stocks using machine learning techniques and artificial neural networks are the most attractive issue to be investigated. As Giles et. al [1] explained, Financial forecasting is an instance of signal processing problem which is difficult because of high noise, small sample size, non-stationarity, and non-linearity. The noisy characteristic means the incomplete information gap between past stock trading price and volume with future price. Stock market is sensitive with political and macroeconomic environment. However, these two kinds of information are too complex and unstable to gather. The above information that cannot be included in features are considered as noise. The sample size of financial data is determined by real world transaction records. On one hand, a larger sample size refers a longer period of transaction records; on the other hand, large sample size increases the uncertainty of financial environment during the two sample period.

There are a lot of complicated financial indicators and also the fluctuation of the stock market is highly violent. However, as the technology is getting advanced, the opportunity to gain a steady fortune from the stock market is increased and it also helps experts to find out the most informative indicators to make a better prediction. The prediction of the market value is of great importance to help in maximizing the profit of stock option purchase while keeping the risk low. Recurrent Neural Networks (RNN) have proved one of the most powerful models for processing sequential data. Long Short-Term memory is one of the most successful RNNs architectures. LSTM introduces the memory cell, a unit of computation that replaces traditional artificial neurons in the hidden layer of the network. With these memory cells, networks are able to effectively associate memories and input remote in time, hence suit to grasp the structure of data dynamically over time with high prediction capacity.

Long short-term memory is a Recurrent Neural Network introduced by Sepp Hochreite and Jurgen Schmidhuber in 1997 [6]. LSTM is designed to forecast, predict and classify time series data even long time lags between vital events happened before. LSTMs have been applied to solve various of problems; among those, handwriting Recognition [7] and speech recognition [8] made LSTM famous. LSTM has copious advantages compared with traditional back-propagation neural networks and normal Recurrent Neural Networks. The constant error back propagation inside memory blocks enables in LSTM ability to overcome long time lags in case of problems similar to those discussed above; LSTM can handle noise, distributed representations and continuous values; LSTM requires no need for parameter fine tuning, it works well over a broad range of parameters such as learning rate, input gate bias and output gate bias [6].

Chapter 2

LITERATURE SURVEY

2.1 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].

2.2 Artificial Neural Network

Physiological Artificial Neural Networks (ANNs) or connectionist systems are computing systems vaguely inspired by the biological neural networks that constitute animal brains.Such systems "learn" (i.e. progressively improve performance on) tasks by considering examples, generally without task-specific programming. For example, in image recognition, they might learn to identify images that contain cats by analysing example images that have been manually labelled as "cat" or "no cat" and using the results to identify cats in other images. They do this without any a priori knowledge about cats, e.g., that they have fur, tails, whiskers and cat-like faces. Instead, they evolve their own set of relevant characteristics from the learning material that they process.

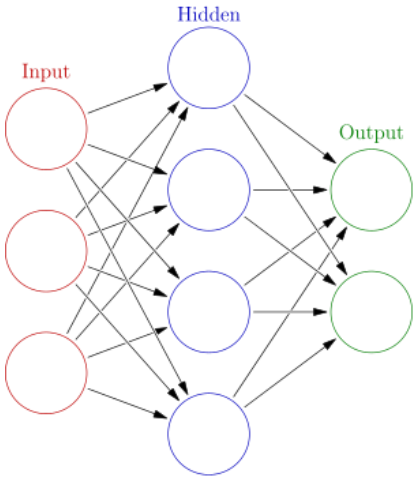

Figure2.1: Artificial Neural Network

An ANN is based on a collection of connected units or nodes called artificial neurons (a simplified version of biological neurons in an animal brain). Each connection (a simplified version of a synapse) between artificial neurons can transmit a signal from one to another. The artificial neuron that receives the signal can process it and then signal artificial neurons connected to it.

2.2.1 Recurrent Neural Network

A Recurrent Neural Network (RNN) is a class of artificial neural network where connections between units form a directed graph along a sequence. This allows it to exhibit dynamic temporal behaviour for a time sequence. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs.

Recurrent Neural Networks can have additional stored states. This storage can be directly under the control of the neural network. The storage can also be replaced by another network or graph, if that incorporates time delays or has feedback loops. Such controlled states are referred to as gated state or gated memory, and are part of Long Short-Term Memorys (LSTM) and gated recurrent units.

Both finite impulse and infinite impulse recurrent networks can have additional stored state, and the storage can be under direct control by the neural network. The storage can also be replaced by another network or graph, if that incorporates time delays or has feedback loops. Such controlled states are referred to as gated state or gated memory, and are part of Long short-term memorys (LSTM) and gated recurrent units.

2.3 LSTM

LSTMs were introduced by Hochreiter*et al*. in [2] which is one of the most commonly used RNN. By using a variety of gates, these networks can regulate the propagation of activations along the network which enables it to learn when to ignore a current input, when to remember the past hidden state or when to emit a non-zero input. These networks are efficient at remembering information for long or short durations of time and hence the name Long Short-Term Memory. The structure of LSTM is given in Fig. 2 There are three gates used: input gate determines how much the current input is let through, forget gate determines how much of the previous state is let through and output gate determines how much the current node influences the external network

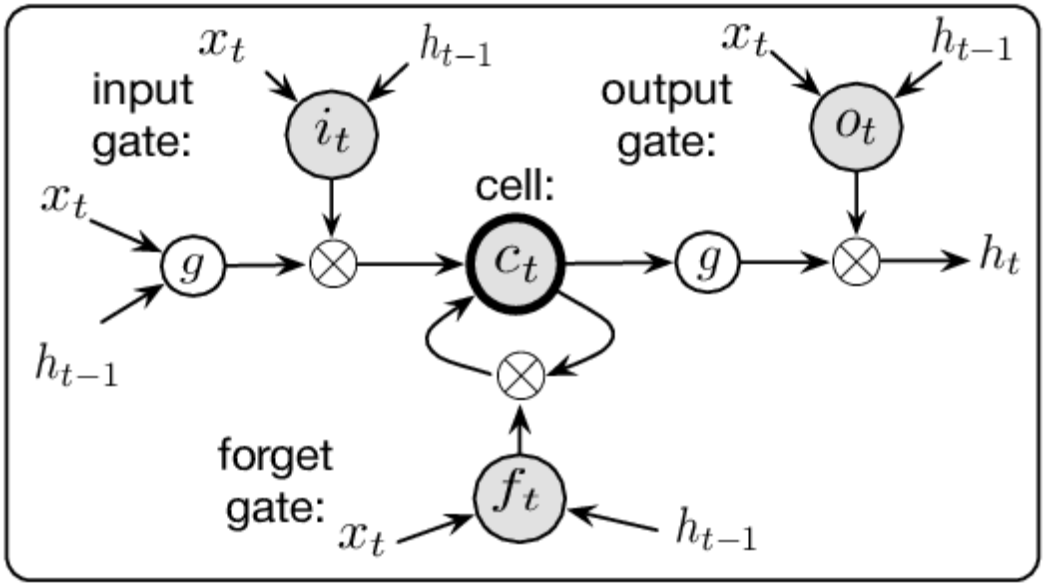

Figure2.2: LSTM Cell

LSTM equations:

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

Chapter 3

PROBLEM STATEMENT

3.1 Introduction

The Stock market process is full of uncertainty and it's affected by many factors. There are two types of analysis possible for prediction, fundamental and technical. The goal is to develop a system that uses technical analysis to satisfactorily predict the future price of stocks.

3.2 Existing Systems

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].

3.3 Fundamental Analysis

Fundamental Analysts are concerned with the company that underlies the stock itself. They evaluate a company's past performance as well as the credibility of its accounts. Many performance ratios are created that aid the fundamental analyst with assessing the validity of a stock, such as the P/E ratio.

Fundamental analysis is built on the belief that human society needs capital to make progress and if a company operates well, it should be rewarded with additional capital and result in a surge in stock price. Fundamental analysis is widely used by fund managers as it is the most reasonable, objective and made from publicly available information like financial statement analysis.

3.4 Technical Analysis

Technical analysts or chartists are not concerned with any of the company's fundamentals. They seek to determine the future price of a stock based solely on the (potential) trends of the past price (a form of time series analysis). Numerous patterns are employed such as the head and shoulders or cup and saucer. Alongside the patterns, techniques are used such as the exponential moving average (EMA). Candle stick patterns, believed to have been first developed by Japanese rice merchants, are nowadays widely used by technical analysts.

Chapter 4

REQUIREMENT ANALYSIS AND ELICITATION

4.1 Functional Requirements

This section describes the functional requirements of the system for those requirements which are expressed in the natural language style:

- Take historical prices of a particular stock as input for training
- Train a model with the provided data to recognise the individual
- Take the latest data and feed it as input for the machine and check its prediction accuracy

4.2 Non-Functional Requirements

A non-functional requirement (NFR) is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviours. They are contrasted with functional requirements that define specific behaviour or functions.

- **Accuracy**: The system must return accurate results, where the correct class is present in the top 5 results 99.5% of the time.
- **Performance**: Retraining a model with new information must be completed in under a minute, and predicting the identity of an individual given an image of their ear should not take more than 10 seconds.
- **Reusability**: The code must be modular and well-documented in order to facilitate future changes and improvements.
- **Portability**: It should be easy to implement in any system.

4.3 Hardware Requirements

Hardware requirements are very minimal and the program can be run on most of the systems. A system with the following features will be more than sufficient to run the system comfortably:

Processor: Intel Quad core and above.

Processor Speed: 1.60GHz and above.

RAM: 4GB and above.

Storage Space: 250 GB and above.

Internet Connection: Required to auto-download dataset.

4.4 Software Requirements

Operating System:

- Ubuntu 16.04 LTS

Language installed: Python 3.5

Python packages:

- TensorFlowversion 1.7.0
- Keras version 2.1.5
- Pandas version 0.23.0
- NumPy version 1.14.2

4.5 Feasibility Study

The feasibility of the project is analysed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are:

- Economical feasibility
- Technical feasibility
- Social feasibility

4.5.1 Economical Feasibility

The proposed system makes use of only freely available open source software, tools, and libraries. Thus, there is no additional cost that needs to be borne by the user in order to utilise, modify, or update the system as long as the hardware requirements are met.

4.5.2 Technical Feasibility

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

Current systems are well equipped to handle the current model. When scaled up or redirected for higher degree of processing, the available processing power and memory of a home computer might not be sufficient. Use of servers (preferably cloud computing) can reduce cost of system upgrading.

4.5.3 Social Feasibility

The aspect of this study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptanceby the users solely depends on the methods that are employed to educate the user about the system and to make them familiar with it. Their level of confidence must be raised so that they are also able to make some constructive criticism, which is welcomed, as they are the final user of the system.

Chapter 5

SYSTEM DESIGN

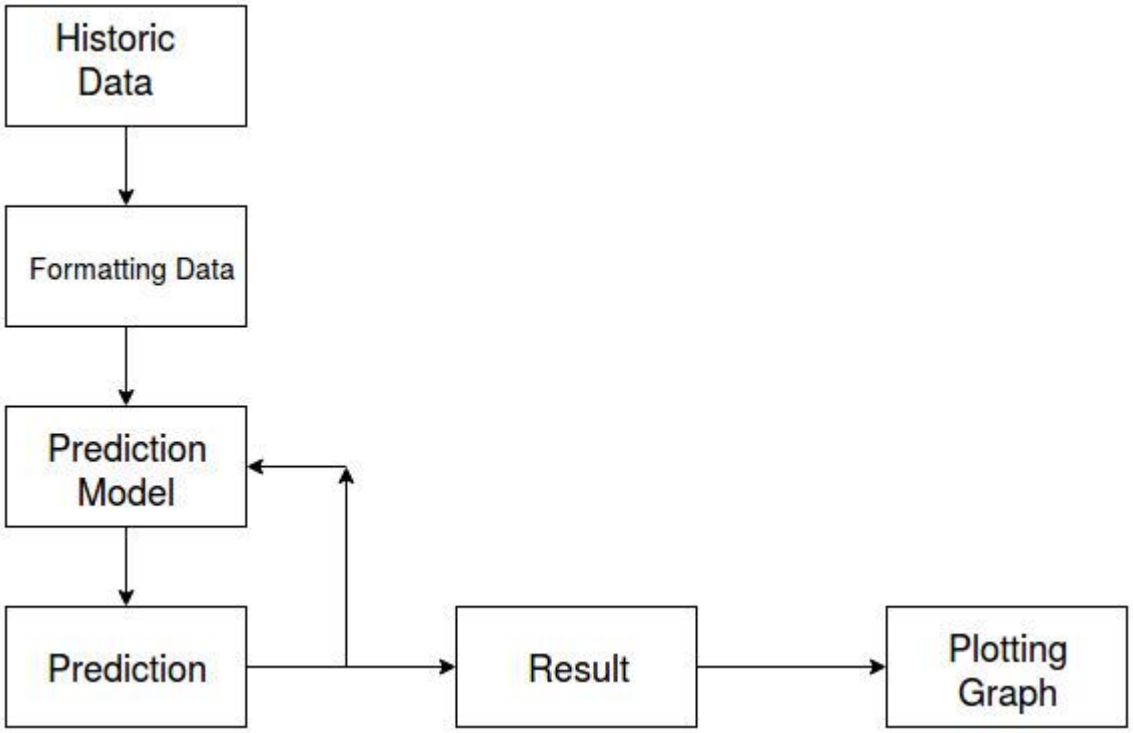

Figure5.1: System Model

The following chapter outlines the general system design of the proposed module. outlines the system design. The system takes in historic data as input. The data is then cleaned for errors like not a number values and fed to the prediction model and the obtained results are plotted in a graph.

5.1 System Architecture

5.1.1 Training Phase

The 90% of the data obtained is used for training which is fed to the neural network Architecture shown below

```
model.add(LSTM(128,input_shape=(1,1),return_sequences=True))
model.add(Dropout(d))
```

```
model.add(LSTM(64,input_shape=(1,1),return_sequences=True))
model.add(Dropout(d))
model.add(LSTM(32,input_shape=(1,1),return_sequences=False))
model.add(Dropout(d))
```

5.1.2 Testing Phase

The Rest of the data (10%) which was unused is now used for testing. This data is fed to the machine and the prediction is done by the model and its accuracy is compared to the actual data which is never seen by the machine before.

Chapter 6

DETAILED DESIGN

6.1 Neural Network Model

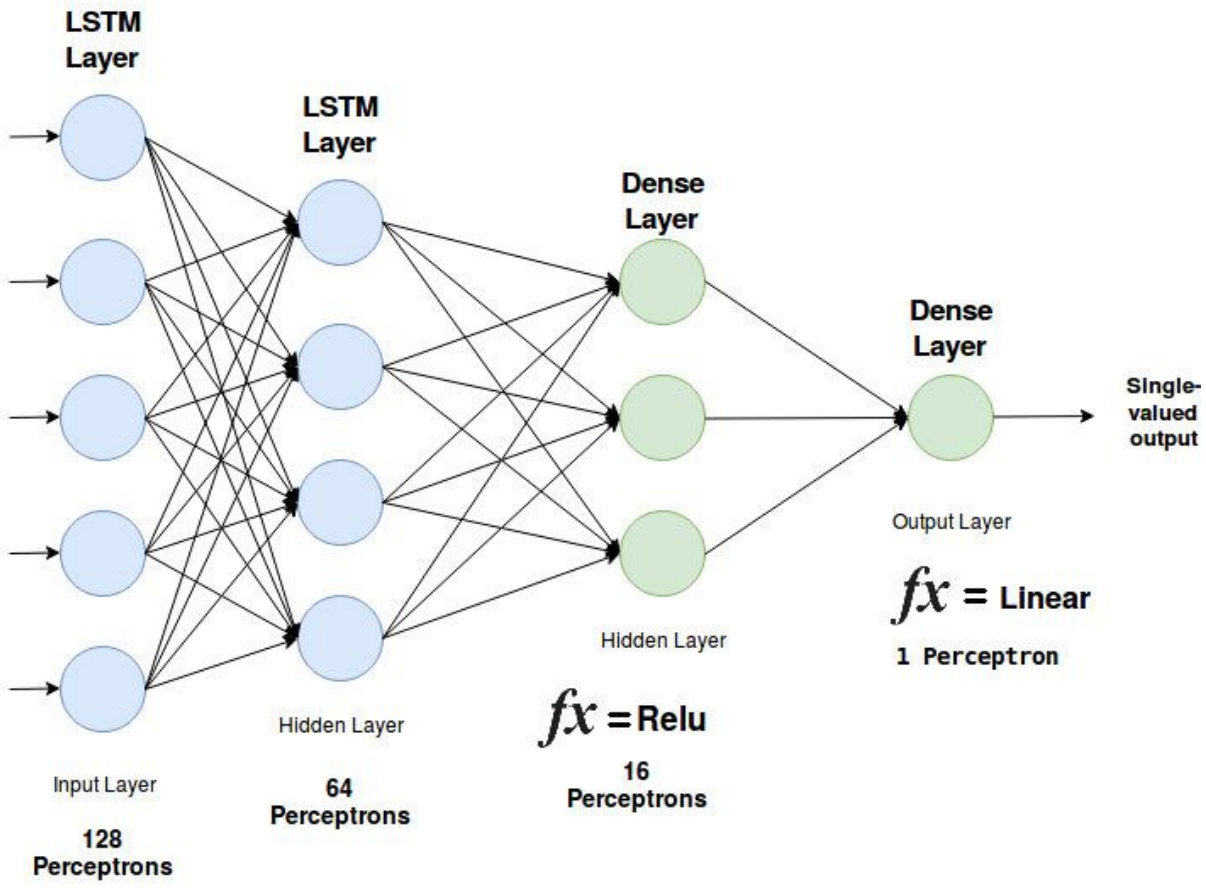

Figure6.1: Model Used For Prediction

The first layer has 128 perceptrons of type LSTM which passes the data to first hidden LSTM layer of 64 perceptrons which passes to another hidden layer which is a 16 perceptron Dense layer with ReLU as the activation function and finally sent to a 1 perceptron Dense layer with linear activation which gives single valued output.

6.2Flowchart

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.

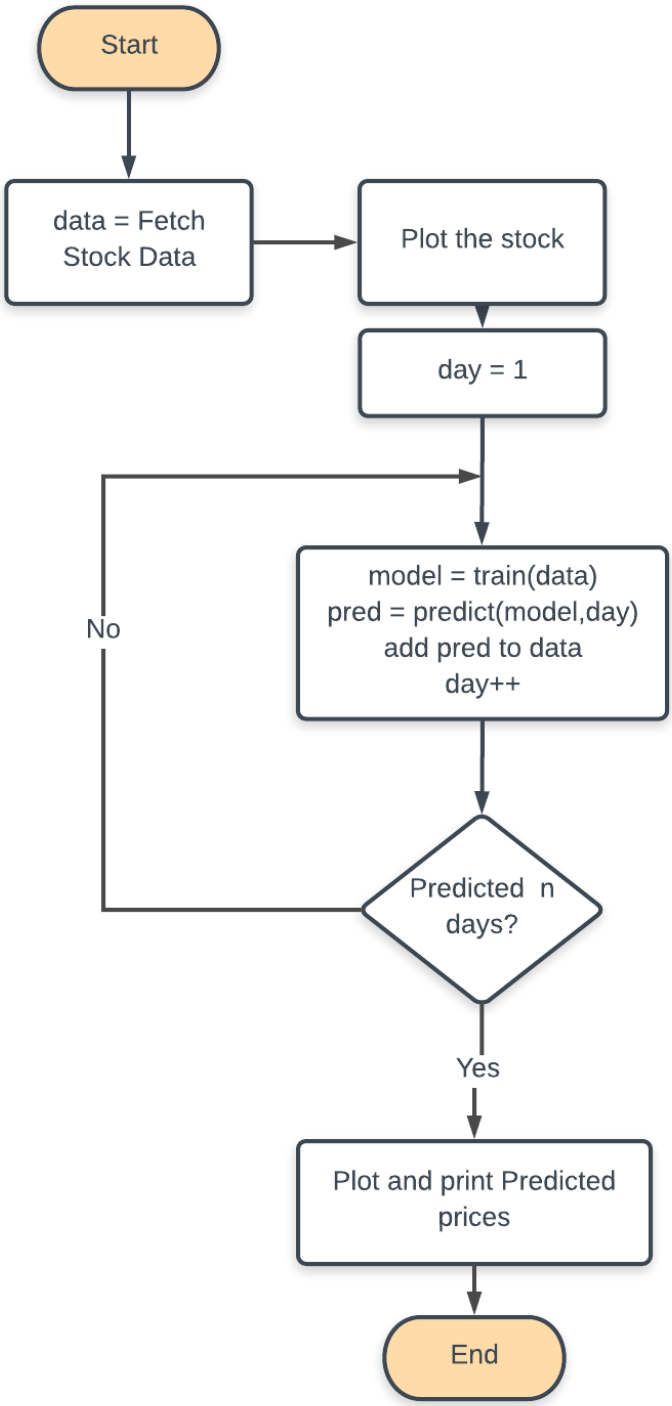

Figure6.2: Flowchart OfThe Proposed System

6.3 Libraries/APIs used

The proposed system utilises several Python libraries and APIs. Keras, Keras-vis, and numpy.

Sl. No.	Name of Library/API	Description
1.	**Keras**	A high-level neural networks API, written in Python on top of TensorFlow[10]
5.	**Numpy**	A Python library for high-level mathematical functions on multi-dimensional arrays
6.	**Keras-Vis**	A visualisation library for Keras models

Table 6.1: Libraries/APIs Used

Chapter 7

IMPLEMENTATION

The code is written in python3.5 and to run the code we have to use certain options and give a few command line parameters such as name of the stock, number of epochs, batch size etc.

7.1 Configuration

Run the code file in python3.x environment alongwith the following command line parameters .

Sl. No.	Options	Actions
1.	-h –help	show this help message and exit
2.	n STOCK_NAME, --name=STOCK_NAME	Name of the stock
3.	-e EPOCH, --epoch=EPOCH	Number of epochs for training the model
4.	-b BATCH_SIZE	Batch Size for training model
5.	-r, --reload_data	Reloads training data and model
6.	-t, --retrain_model	Retrains the model
7.	-s TEST_SIZE, --test_size=TEST_SIZE	Number of days to test the model with
8.	-f ,FUTURE_DAYS	Number of days to predict
9.	-a, --sample_case	Show Sample Results(Demo)
10.	-x TRAIN_SIZE, --train_size=TRAIN_SIZE	Adjust training set size

Table 7.1: Command Line options

If no command line parameters are given then the previously stored models of a few standard and pre-trainedstocks are displayed along with predicted values and accuracy graph.

7.2 Code

```
#STOCK MARKET PREDICTION USING HISTORIC DATA

importnumpy as np
importmatplotlib.pyplot as plt
import pandas as pd
from pandas
importdatetime
import math, time
importitertools

fromkeras
import optimizers
importdatetime
from operator
importitemgetter
from math
importsqrt
fromkeras.models
import Sequential
fromkeras.layers.core

import Dense, Dropout, Activation, Reshape, Flatten
fromkeras.layers.convolutional
import Conv2D
fromkeras.layers.convolutional
import Conv3D
fromkeras.layers.recurrent
import LSTM
fromkeras.layers.convolutional_recurrent
import ConvLSTM2D
fromkeras.layers.advanced_activations
```

```
importLeakyReLU, PReLU, ELU, ThresholdedReLU
fromkeras
import * from keras.callbacks
import * from keras.layers.normalization
importBatchNormalization
importmatplotlib.pyplot as graph
importcsv
import requests
import operator
import sys

importnumpy as np
importos
fromsklearn
import metrics
fromdatetime
import date
fromdatetime
importdatetime
importshutil
fromdatetime
importtimedelta

fromoptparse
importOptionParser, SUPPRESS_HELP
fromkeras.models
importmodel_from_json

# Directory to be used for storing stock data
master_directory = "./stocks"

#Used to scan the command - line arguments for options and values

defgenerate_opt_parser():
```

```
usage_format = "usage: %prog [options] arg1 arg2"
parser = OptionParser(usage = usage_format)
parser.add_option("-n",
        "--name",
        action = "store",
        type = "string",
        dest = "stock_name",
        default = "EMPTY",
        help = "Name of the stock")
parser.add_option("-e",
        "--epoch",
        action = "store",
        type = "int",
        dest = "epoch",
        default = 300,
        help = "Number of epochs for training the model")
parser.add_option("-b",
        "--batch_size",
        action = "store",
        type = "int",
        dest = "batch_size",
        default = 16,
        help = "Batch Size for training model")
parser.add_option("-r",
        "--reload_data",
        action = "store_true",
        dest = "reload_data",
        default = False,
        help = "Reloads training data and model")
parser.add_option("-t",
         "--retrain_model",
        action = "store_true",
        dest = "retrain_model",
        default = False,
```

```
            help = "Retrains the model")
    parser.add_option("-s",
            "--test_size",
            action = "store",
            type = "int",
            dest = "test_size",
            default = 30,
            help = "Number of days to test the model with")
    parser.add_option("-f",
             "--future_days",
            action = "store",
            type = "int",
            dest = "future_days",
            default = 7,
            help = "Number of days to predict")
    parser.add_option("-a",
            "--sample_case",
            action = "store_true",
            dest = "sample_mode",
            default = False,
            help = "Show Sample Results(Demo)")
    parser.add_option("-x",
            "--train_size",
            action = "store",
            type = "int",
            dest = "train_size",
            default = 365,
            help = "Adjust training set size")

    (options, args) = parser.parse_args()

    # If no stock has been given in the commandline, then
    default to DEMO mode
```

```
        ifoptions.stock_name == "EMPTY":
                options.sample_mode = True

        # If the model has to be retrained
        ifoptions.retrain_model == True:
                model_loc_json = master_directory + "/" + str(options.stock_name) +
                "/model/" + str(options.stock_name) + ".json"
                model_loc_h5 = master_directory + "/" + str(options.stock_name) + "/model/"
                + str(options.stock_name) + ".h5"
                if (os.path.exists(model_loc_json)):
                        os.remove(model_loc_json)
                if (os.path.exists(model_loc_h5)):
                        os.remove(model_loc_h5)

        # If the data of the stock has to be downloaded again
        ifoptions.reload_data == True:
                name = master_directory + "/" + str(options.stock_name)
        ifos.path.exists(name):
                shutil.rmtree(name)

        return (options, args)

(options, args) = generate_opt_parser()
test_size = options.test_size
future_num = options.future_days
train_size = options.train_size

# Used to get the next market - working day after 'date'

defget_next_date(date):
        today = datetime.strptime(date, "%Y-%m-%d")
        today += timedelta(1)

        iftoday.isoweekday() in set((6, 7)):
```

```
            today = today + timedelta(8 - today.isoweekday())
        returnstr(today.strftime("%Y-%m-%d"))

# Used to create the directory structure for the stock
# If the Directory already exists, no change is made

defconstruct_path(stock_name):
        name = master_directory + "/" + str(stock_name)
        ifos.path.exists(name) and os.path.exists(name + "/data") and os.path.exists(name +
        "/output_image") and os.path.exists(name + "/model"):
            return
        ifos.path.exists(name):
            shutil.rmtree(name)
        os.mkdir(name)
        os.mkdir(name + "/data")
        os.mkdir(name + "/output_image")
        os.mkdir(name + "/model")

# Fetches stock data from the internet# Places the data in the relevant folder.If the file already
exists, it is NOT redownloaded

defpopulate_data(stock_name):
        name = str(stock_name)
        path_data = master_directory + "/" + name + "/data"
        file_path = path_data + "/" + name + "_details.csv"#
        URL from which to fetch stock details
        CSV_URL='https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&
        symbol=%s&outputsize=full&apikey=4QTXZKSRGCINTK64&datatype=csv' %
        (name)
        ifos.path.exists(file_path):
            return
        withrequests.Session() as s:
            download = s.get(CSV_URL)
```

```
        decoded_content = download.content.decode('utf-8')
        cr = csv.reader(decoded_content.splitlines(), delimiter = ',')
        # sort the stock details based on date
        cr = sorted(cr, key = operator.itemgetter(0))

        my_list = list(cr)
        f1 = open(file_path, "w")# writing all stock details to the file
        for i in range(0, len(my_list) - 1):
                ifmy_list[i][3] != '0.0000':
                        f1.write(my_list[i][0] + "," + my_list[i][1] + "," + my_list[i][2] + "," +
                        my_list[i][3] + "," + my_list[i][4] + "," + my_list[i][5] + "\n")
        f1.close()

# The directory of the stock data, the hstoric data are all populated

def preprocess(stock_name):
        name = str(stock_name)
        ifos.path.exists(master_directory) == False:
                os.mkdir("stocks")
        construct_path(stock_name)
        populate_data(stock_name)

# Thisfunction returns the required column of data# The column names are hashed to their
respective index values

defget_data(stock_name, column):
        column_id = {
           "Date": 0,
           "Open": 1,
           "High": 2,
           "Low": 3,
           "Close": 4,
           "Volume": 5
         }
```

```
    name = str(stock_name)
    path_data = master_directory + "/" + name + "/data/" + name + "_details.csv"
    table = pd.read_csv(path_data, header = None)
    return table[column_id[column]]


# Function to generate and compile an instance of the defined model

def build():
    d = 0.2
    model = Sequential()
    # adding input LSTM layer with 128 perceptrons
    model.add(LSTM(128, input_shape = (1, 1), return_sequences = True))
    # adding hidden LSTM layer with 64 perceptrons
    model.add(LSTM(64, input_shape = (1, 1), return_sequences = False))
    # adding hidden dense layer with 16 perceptrons
    model.add(Dense(16, kernel_initializer = 'uniform', activation = 'relu'))
    # adding output layer with single perceptron
    model.add(Dense(1, kernel_initializer = 'uniform', activation = 'linear'))
    # compiling the model
    model.compile(loss = 'mse', optimizer = 'adam', metrics = ['acc', 'mse'])
    # returning the compiled model
    return model

# Thisfunction returns the model of a stock that has been trained on historic data
# If the model already exists, then it is not retrained.
#The trained model is also stored before being returned

defget_model(stock, train_x, train_y):
    stock_name = str(stock)
    model_loc_json = master_directory + "/" + stock_name + "/model/" + stock_name +
    ".json"
```

```
        model_loc_h5 = master_directory + "/" + stock_name + "/model/" + stock_name +
        ".h5"
#check if path exists
        ifos.path.exists(model_loc_json) and os.path.exists(model_loc_h5):
        #read the existing model
        json_file = open(model_loc_json, 'r')
        loaded_model_json = json_file.read()

        json_file.close()
        model = model_from_json(loaded_model_json)
        model.load_weights(model_loc_h5)
        print("Loaded an existing model for " + stock_name)
        return model
        # model does not exist so it has to be trained
        model = build()
        # data on which the model must be trained
        train_x = train_x.reshape(train_x.shape[0], 1, 1)
        print("Training " + str(stock))
        model.fit(train_x, train_y, batch_size = options.batch_size, epochs = options.epoch,
        validation_split = 0.1, verbose = 1)
        # Saving the trained modelfor future use
        model_json = model.to_json()
        with open(model_loc_json, "w") as json_file:
                json_file.write(model_json)
        model.save_weights(model_loc_h5)
        return model

# Thisfunction returns the predicted values based on the model and set of values sent

defpredict_model_val(model, predict_x):
        predict_x = predict_x.reshape(predict_x.shape[0], 1, 1)
        result = model.predict(predict_x)
        result = np.array(result)
        return result
```

```
# Thisfunction is used to select the last 'num' number of entries

def downsize(data, num):
        data = data[data.shape[0] - num: data.shape[0]]
        return data




defget_arr(a):
        p = []
        for i in range(0, a.shape[0]):
                p.append(a.item(i))
        return p

# Returns the accuracy array

defget_accuracy(actual, pred):
        accuracy = np.array([])
        actual = np.array(actual)
        pred = np.array(pred)
        for i in range(0, pred.shape[0]):
                diff = abs(pred.item(i) - actual.item(i))
        error_p = (diff / actual.item(i)) * 100
        acc = 100 - error_p
        accuracy = np.append(accuracy, acc)
        return accuracy

# Thisfunction partitions the data - set into training and test data set
# The model is trained on the training data
# The model is tested and the result is pushed to a graph
# The accuracy of each data point is calculated and pushed to the 'Accuracy' graph

defget_output(stock_name, color_graph):
        preprocess(stock_name)
```

```
        data = get_data(stock_name, "Close")
        dates = get_data(stock_name, "Date")
        test_size = int(0.1 * data.shape[0])
        graph_image_loc = master_directory + "/" + stock_name + "/output_image/" +
        stock_name + ".png"
        train_x = np.array(data[0: data.shape[0] - test_size])
        train_y = np.array(data[1: data.shape[0] - test_size + 1])
        test = np.array(data[data.shape[0] - test_size: data.shape[0] - 1])

        predict_x = np.array(test)
        # Creating set for prediction
        predict_test_y = np.array(data[data.shape[0] - test_size + 1: data.shape[0]])
        predict_date_y = np.array(dates[dates.shape[0] - test_size + 1: dates.shape[0]])
        model = get_model(stock_name, train_y, train_x)
        predict_y = predict_model_val(model, predict_x)
        accuracy = np.array([])
        avg_acc = []
        points = []
        acc_window = 10
        accuracy = get_accuracy(predict_test_y, predict_y)
        # Getting the points to plot on the graph
        for i in range(0, int(predict_y.shape[0] / acc_window)):
                limit = i * acc_window + acc_window
        if limit >predict_y.shape[0]:
                limit = predict_y.shape[0]
        sum = 0
        for j in range(i * acc_window, limit):
                sum = sum + accuracy.item(i)
        sum = sum / acc_window;
        avg_acc.append(sum)
        points.append((i + 1) * acc_window)

# Print the Stock actual Vs Expected price graph and save it
        graph.figure(stock_name)
```

```
        graph.plot(predict_y, label = "prediction " + stock_name)
        graph.plot(predict_test_y, label = "actual " + stock_name)
        graph.legend(loc = 'upper right')

# Print the Stock accuracy graph
        graph.savefig(graph_image_loc)
        graph.figure("Accuracy")
        graph.plot(points, avg_acc, label = "accuracy " + stock_name, color = color_graph)
        x1, x2, y1, y2 = graph.axis()
        graph.axis((x1, x2, 0, 100))
        graph.legend(loc = 'upper right')

ifoptions.sample_mode == False: #For execution of a single stock
        stock_name = str(options.stock_name)
        get_output(stock_name, "blue")

else :
#DEMO mode shows summary of all the stocks in the array 'stocks'
        stocks = ["SBIN.NS", "AAPL", "RELIANCE.NS", "MSFT", "TATAMOTORS.NS",
        "BHARTIARTL.NS", "GOOGL"]
        colors = ["blue", "green", "red", "cyan", "magenta", "yellow", "black"]

for i in range(0, len(stocks)):
        get_output(stocks[i], colors[i])

# Plot and save accuracy graph

graph.figure("Accuracy")
graph.savefig("./stocks/accuracy.png")
graph.show()
```

Chapter 8

RESULT ANALYSIS

The following chapter outlines the testing methodologies utilised while building this project, as well as the achieved results and an analysis of the same. The purpose of testing is to discover errors. Testing provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product.

8.1 Testing Design

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

- White Box Testing
- Black Box Testing

8.1.1 White Box Testing

This testing is also called as glass box testing. In this 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.

8.1.2 Black Box Testing

In this 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
- Equivalence partitioning
- Boundary value analysis
- Comparison testing

8.2 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.

8.3 Levels of Testing

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

- Unit Test
- Integration test
- Functional Test

8.3.1 Unit Test

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.

8.3.2 Integration Test

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.

8.3.3 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.

8.4 Results

Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations. It is an ambiguous assumption that if we have high accuracy then our model is best. Yes, accuracy is a great measure but only when we have symmetric datasets where values of false positive and false negatives are almost same. Therefore, other parameters must also be considered while evaluating performance of a model.

8.4.1 Prediction on Certain stocks

Here are some of the predictions made by the machine on giving historical data as the input to the neural networks.

8.4.1.1 GOOGLE Stock

The historical data used corresponds to only the Closing prices of the Google stock (NASDAQ: GOOGL). The data is divided in the ratio of 90:10 with respect to the training and testing sets. The given graph is a representation of the actual Vs the predicted value of the stock for the testing set.

The accuracy of the prediction is calculated as the percentage of the difference in value of the predicted price against the actual price.

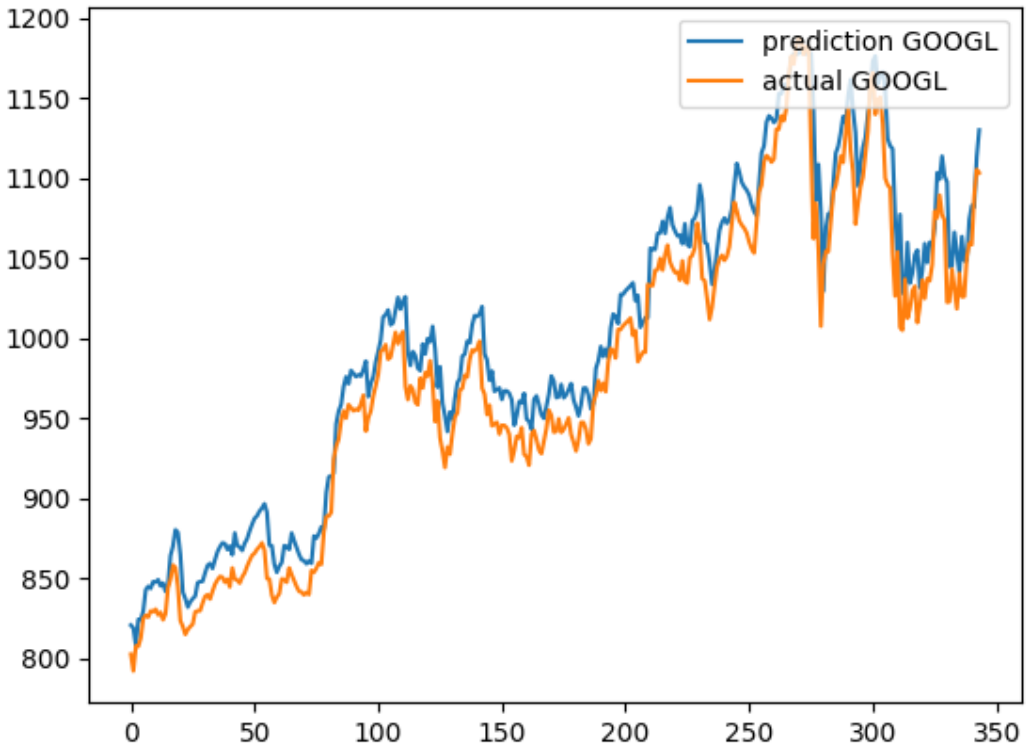

Figure 8.1 GOOGL stock

8.4.1.2 APPLE Stock

The historical data used corresponds to only the Closing prices of the apple stock (NASDAQ: AAPL). The data is divided in the ratio of 90:10 with respect to the training and testing sets. The given graph is a representation of the actual Vs the predicted value of the stock for the testing set.

The accuracy of the prediction is calculated as the percentage of the difference in value of the predicted price against the actual price.

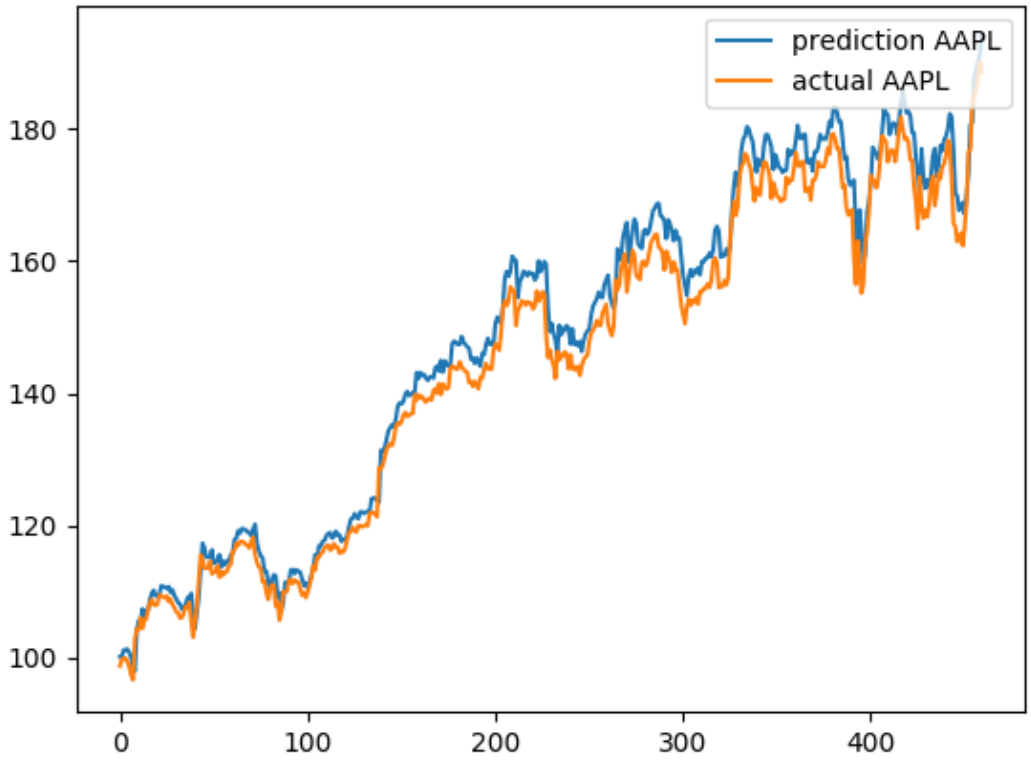

Figure 8.2 AAPL stock

8.4.1.3 MICROSOFT Stock

The historical data used corresponds to only the Closing prices of the Microsoft stock (NASDAQ: MSFT). The data is divided in the ratio of 90:10 with respect to the training and testing sets. The given graph is a representation of the actual Vs the predicted value of the stock for the testing set.

The accuracy of the prediction is calculated as the percentage of the difference in value of the predicted price against the actual price.

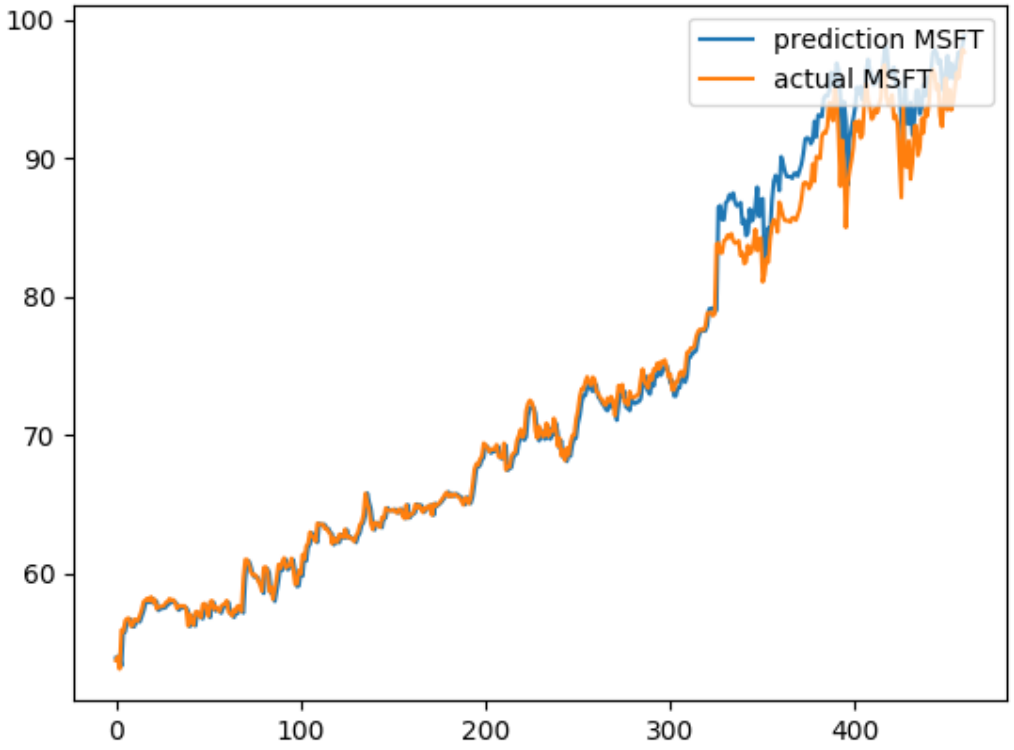

Figure 8.3 MSFT stock

8.4.1.3 SBI Stock

The historical data used corresponds to only the Closing prices of the Sbi stock (NSE: SBIN). The data is divided in the ratio of 90:10 with respect to the training and testing sets. The given graph is a representation of the actual Vs the predicted value of the stock for the testing set.

The accuracy of the prediction is calculated as the percentage of the difference in value of the predicted price against the actual price.

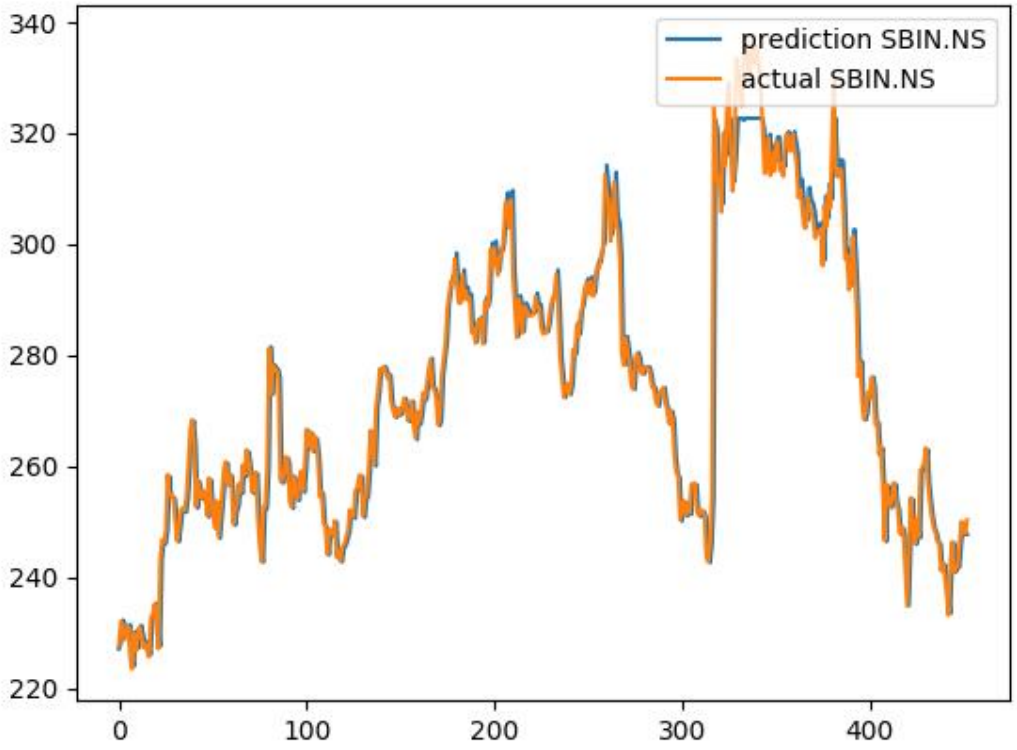

Figure 8.4 SBIN stock

8.4.1.5 TATA MOTORS Stock

The historical data used corresponds to only the Closing prices of the Tata motors stock (NSE: TATAMOTORS). The data is divided in the ratio of 90:10 with respect to the training and testing sets. The given graph is a representation of the actual Vs the predicted value of the stock for the testing set.

The accuracy of the prediction is calculated as the percentage of the difference in value of the predicted price against the actual price.

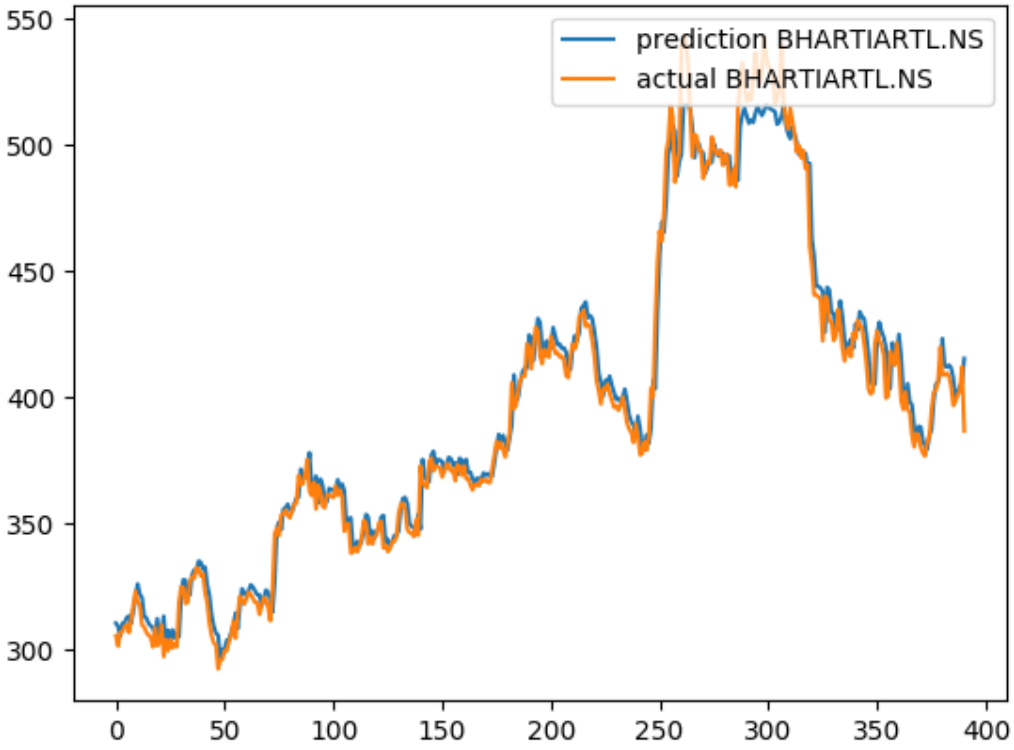

Figure 8.5 TATAMOTORS stock

8.4.1.6 AIRTEL Stock

The historical data used corresponds to only the Closing prices of the Airtel stock (NSE: BHARTIARTL). The data is divided in the ratio of 90:10 with respect to the training and testing sets. The given graph is a representation of the actual Vs the predicted value of the stock for the testing set.

The accuracy of the prediction is calculated as the percentage of the difference in value of the predicted price against the actual price.

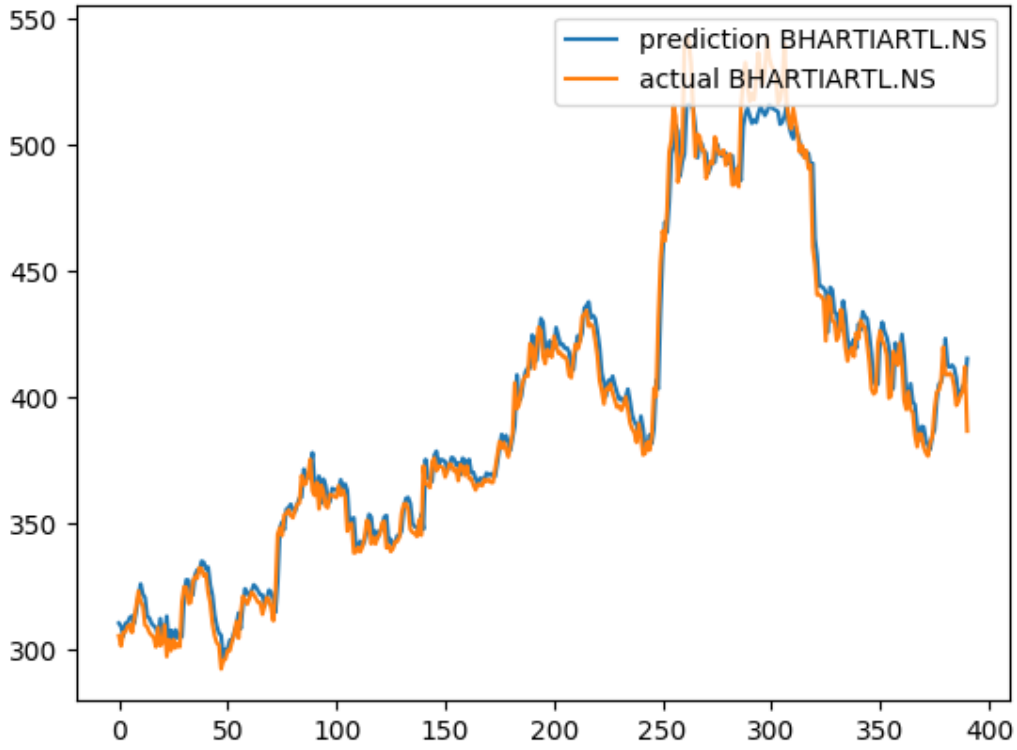

Figure 8.6 BHARATIARTL stock

8.4.2 ACCURACY

The accuracy of the prediction is calculated as the percentage of the difference in value of the predicted price against the actual price.

The Following graph shows the accuracy of stock prediction at various data points of the test set. The test set was 10% of the total dataset. The following stocks are considered for testing purposes – State Bank Of India (NSE: SBIN), Apple (NASDAQ: AAPL), Reliance (NSE:RELIANCE), Microsoft (NASDAQ:MSFT) Tata Motors (NSE: TATAMOTORS), Bharati Airtel (NSE:BHARTIARTL) and Google (NASDAQ: GOOGL).

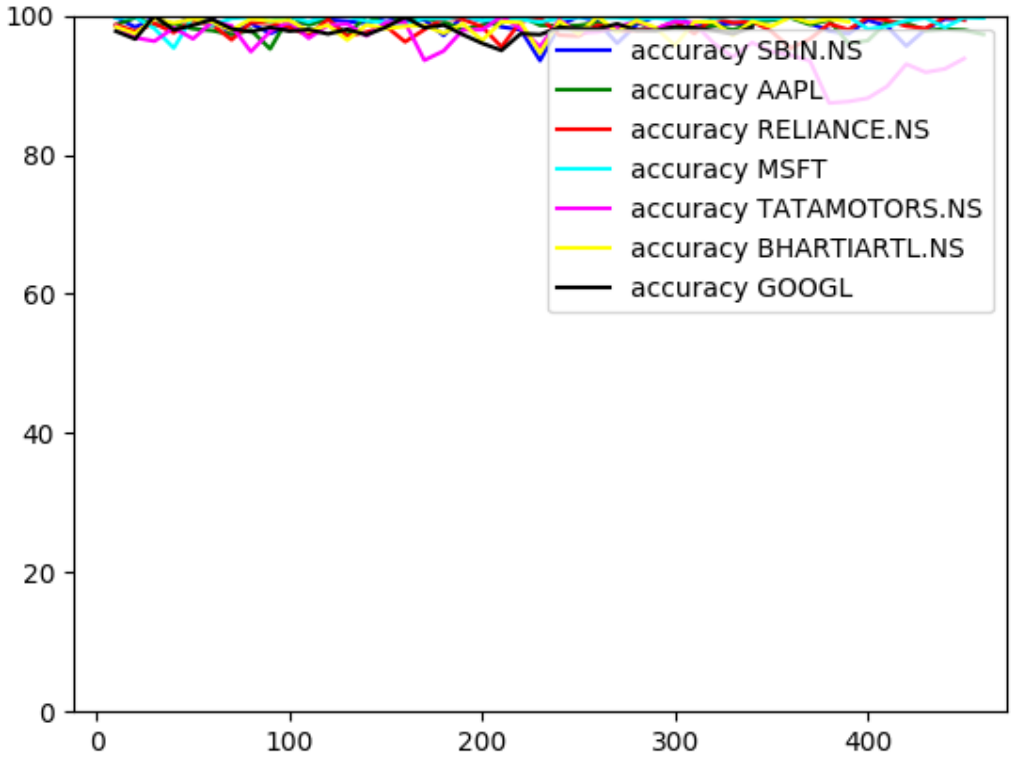

Figure 8.7 Accuracy graph

Chapter 9

CONCLUSION AND FUTURE WORK

9.1 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 LSTMs 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 future price of the stock. 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.

9.2 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.

GLOSSARY

Neural Network: Artificial neural networks (ANNs) or connectionist systems are computing systems vaguely inspired by the biological neural networks that constitute animal brains. Such systems learn tasks by considering examples, generally without task-specific programming.

Machine learning: Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed.

Logistic Regression: Logistic regression is a statistical method for analysing a dataset in which there are one or more independent variables that determine an outcome.

ReLU: In the context of artificial neural networks, the rectifier is an activation function defined as the positive part of its argument .

REFERENCES

1. Anjan Kumar KN Probabilistic classification techniques to perform geographical labeling of web object https://link.springer.com/article/10.1007/s10586-018-1822-y?wt_mc=Internal.Event.1.SEM.ArticleAuthorOnlineFirst
2. 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.
3. G. Batres-Estrada, “Deep learning for multivariate financial time series,”ser. Technical Report, Stockholm, May 2015.
4. 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
5. V. Devadoss and T. A. A. Ligori, “Forecasting of stock prices using multi-layer perceptron,” Int J Comput Algorithm, vol. 2, pp. 440–449, 2013.
6. V. K. Menon, N. C. Vasireddy, S. A. Jami, V. T. N. Pedamallu, V. Sureshkumar, and K. Soman, “Bulk price forecasting using spark over nse data set,” in International Conference on Data Mining and Big Data. Springer, 2016, pp. 137–146.
7. G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, Time series analysis: forecasting and control. John Wiley & Sons, 2015.
8. "LogisticRegression,"[Online].Available:https://en.wikipedia.org/wiki/Logistic_regression [Accessed May 2018].
9. F. a. o. Chollet, "Keras," 2015. [Online]. Available: https://keras.io. [Accessed May 2018].
10. S. Hochreiter and J. Schmidhuber, “Long short-term memory,” Neural computation, vol. 9, no. 8, pp. 1735–1780, 1997.
11. X. Ding, Y. Zhang, T. Liu, and J. Duan, “Deep learning for event-driven stock prediction.” in IJCAI, 2015, pp. 2327–2333.
12. Mart´ınAbadi, AshishAgarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, RajatMonga, Sherry Moore, Derek Murray, Chris ´ Olah, Mike Schuster, Martin Wattenberg, Martin Wicke, Yuan Yu, and XiaoqiangZheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.