Prediction of Stock Market Security Price using Chart Patterns and Indicator Analysis

A Comprehensive Project report Submitted in Partial Fulfillment of the Requirements for Award
of
the Degree of Bachelor of Technology in Information and Communication Technology

Submitted by
Rutvik Vijay Patel
Roll No. 17BIT048

Parth Parulkumar Shah
Roll No. 17BIT051

Under the Supervision and Guidance of
Dr. Jigarkumar Shah
Assistant Professor
Department of Information and Communication Technology
Pandit Deendayal Energy University

Submitted to
Department of Information and Communication Technology
School of Technology
Pandit Deendayal Energy University (PDEU)
Gandhinagar, INDIA, 382007

Declaration

We hereby declare that the project work entitled **"Prediction of Stock Market Security Price using Chart Patterns and Indicator Analysis"** is an authentic record of our work carried out at Pandit Deendayal Energy University as a requirement of B. Tech dissertation for the award of **Bachelor of Technology in Information and Communication Technology**. We have duly acknowledged all the sources from which the ideas and extracts have been taken. The project is free from any plagiarism and has not been submitted elsewhere for any degree, diploma, or certificate.

Signature: ....
Rutvik Vijay Patel
(Roll No. 17BIT048)

Signature: ....
Parth Parulkumar Shah
(Roll No. 17BIT051)

Department of Information and Communication Technology

PDEU

Dr. Ganga Prasad Pandey
H. O. D.
I.C.T. Department

Phone: +91-79-2327 5397
e-mail: Gangaprasad.Pandey@sot.pdpu.ac.in

Certificate

This is to certify that the project entitled **"Prediction of Stock Market Security Price using Chart Patterns and Indicator Analysis"** submitted by **Rutvik Vijay Patel**, Roll No. 17BIT048 and **Parth Parulkumar Shah**, Roll No. 17BIT051 to the Department of Information and Communication Technology under School of Technology, PDEU in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Information and Communication Technology** embodies work carried out under the guidance and supervision of Dr. Jigarkumar Shah, Assistant Professor, Department of Information and Communication Technology.

Signature: ....
Dr. Ganga Prasad Pandey
(HOD, ICT Dept.,PDEU)

Certificate of Approval

The forgoing project entitled **"Prediction of Stock Market Security Price using Chart Patterns and Indicator Analysis"** submitted by **Rutvik Vijay Patel**, Roll No. 17BIT048 and **Parth Parulkumar Shah**, Roll No. 17BIT051 to the Department of Information and Communication Technology under School of Technology, PDEU is hereby approved as project work carried out and presented in a manner satisfactory to warrant its acceptance as a prerequisite of **Bachelor of Technology in Information and Communication Technology** degree for which it has been submitted. It has been understood that by this approval of the undersigned do not necessarily endorse or approve every statement made, opinion expressed, or conclusion drawn therein but approved only for the purpose for which it is being submitted.

Signature: ....
Examiner

Acknowledgement

The success and outcome of this project required a lot of guidance and assistance from many people. It is my privilege to pledge the following few lines of dedication to those who helped me directly or indirectly in our project.

We wish to express our sincere gratitude to the faculties of the Department of Information and Communication Technology, School of Technology, Pandit Deendayal Energy University for providing us with the great opportunity to work on ”Prediction of Stock Market Security Price using Chart Patterns and Indicator Analysis”, which is our attempt to design a predictive machine learning model incorporating historical data that will benefit the masses.

We are also thankful to our well-wishers for their unceasing encouragement and support to overcome all the hurdles. Furthermore, we are thankful for and fortunate enough to the faculties of the Department for all the knowledge they have imparted, which we could put to use in this project.

Most importantly, we appreciate the co-operation and benevolence of our friends for helping us with the development and our families for cheering us up throughout this journey.

Abstract

A large volume of volatile stock-market price data is generated every minute. It would not be wrong to say that the modern world is heavily affected by stock prices and related securities, making stock price prediction one of the most important and sought-after problems in the world of economics and finance. In this project, we have attempted to create and compare various approaches that would predict stock prices. For this, we have built five machine learning models for time series and chart pattern analysis. The Long Short Term Memory (LSTM), Recurrent Neural Network (RNN), and Gated Recurrent Unit (GRU) models are based directly on historical data of the stock prices, whereas the chart pattern analysis is also based on encoder-decoder architecture and Deep Convolutional Neural Networks (DCNN). Furthermore, to leverage the best qualities of each of the three (LSTM, RNN, GRU) approaches, we have combined these models to form an ensemble approach for enhanced performance, which is also based on historical data of the stock prices.

Contents

Chapter 1

Introduction

Considering today's world which is highly intertwined with financial markets, stock price prediction becomes vital to business people and common people alike. Stock markets being a complex system, it is an uphill task to build an accurate model for the prediction of stock prices as various volatile factors are affecting it, such as social media, production of the company, fundamentals, central bank interest rates, news, and country's economic aspects.

Over the years, many people have tried to devise different methods to predict stock price movements. These are mainly based on technical indicators and fundamental analysis. We have explored various machine learning-based approaches to build multiple stock price-prediction models. We have considered the trends from the historical data to create prediction models, based on the fact that history tends to repeat itself. We have built five models, which use historical data, as a part of our project. The DCNN model uses the encoder-decoder architecture for feature extraction (chart pattern recognition) predicting the stock price for the next day. LSTM, RNN, and GRU models use historical stock prices to predict future stock price trends. In addition, the ensemble approach, which is a combination of LSTM, RNN and GRU approaches, using Artificial Neural Networks (ANN) is also based on historical stock price data. We have taken into account the following parameters for each model: open, close, high, and low prices for the DCNN model, and close price for LSTM, RNN, GRU, and Ensemble models.

1.1 Problem Statement

The stock market is in the limelight on daily basis for various reasons like reaching a new high or a new low. In the stock market, the rate of investment and business opportunities for the investors could improve significantly if we can devise an efficient approach to predict the price of a stock. In this project, we have attempted to devise a model using various Machine Learning approaches which will efficiently predict stock prices.

1.2 Motivation

Stock price prediction is a classic and important problem. An efficient prediction model could help gain valuable insights into the behavior of the market over some time by identifying hidden trends. Although, increased computational power has opened various avenues which have enabled us to use Machine Learning to develop solutions for these types of problems, yet the costs of computations restrict the performance of our models as we need to consider virtually infinite variables involved in this problem.

In this report, we have introduced the five prediction models of our project which use historical stock price data for prediction. The idea is that, if the historical data of each stock (all parameter values) is known, the future stock prices are somewhat predictable.

Thus, our motivation is to design a machine learning model incorporating historical data that will benefit the masses.

1.3 Literature Review

After diligent research, we came across the work of Kannan, Sekar, Sathik, and P. Arumugam. They have exploited the automated computer programs using data mining and predictive technologies. As data mining is well-founded on the theory that the historic data holds the essential memory for predicting the future direction, their method is designed to recognize hidden patterns from the historic data. Data analysis is one way of predicting future stock prices, they have combined five methods of analyzing stocks to predict the closing stock price. These methods were Typical Price (TP), Bollinger Bands, Relative Strength Index (RSI), CMI, and Moving Average (MA). In this paper, the author got the profitable signal of significant accuracy using Bollinger Bands rather than MA, RSI, and CMI.

Secondly, Jing Tao Yao and Chew Lim Tan have used artificial neural networks for classification, prediction, and recognition. The authors have discussed a seven-step neural network prediction model building approach in this paper. They have covered pre and post-data processing/analysis skills, data sampling, training criteria, and model recommendation in this article.

On the other hand, Tiffany Hui-Kuang Yu and Kun-Huang Huarng exploited the capability of neural networks in handling nonlinear relationships by implementing a new fuzzy time series model to improve stock price prediction. They have predicted the Taiwan Stock index using a fuzzy relationship. Moreover, due to the availability of an abundance of data, their model can be used to predict regardless of whether out-of-sample observations appear in the in-sample observations. Their study performs out-of-sample forecasting and the results are compared with those of previous studies to demonstrate the performance of their proposed model. In addition to the research papers, we have also referred to various articles on the problem, where we came across various approaches based on Technical Analysis, traditional time series forecasting, fundamental analysis, and machine learning techniques like linear regression, polynomial regression, etc.

Furthermore, Abhinandan Gupta, Dev Kumar Chaudhary, and Tanupriya Choudhury have used Functional Link Artificial Neural Network (FLANN), originally introduced by Yoh-Han Pao and Yoshiyasu Takefuji. In this, generally, a single-layer network structure is present, having only one hidden layer for computation. The non-linearity is introduced using non-linear functional expansion. This network requires less time for computations than the multiple layer perceptron with back-propagation.

Lastly, Mehar Vijh, Deeksha Chandola, Vinay Anand Tikkiwal, and Arun Kumar have proposed the use of the Random Forest approach for stock price prediction. In this paper, Artificial Neural Networks and Random Forest techniques have been utilized for predicting the next day closing price for five companies belonging to different sectors of operation. The financial data: Open, High, Low, and Close prices of stock are used for creating new variables which are used as inputs to the models. The models are evaluated using standard strategic indicators: Root Mean Square Error (RMSE) and Mean Absolute Percentage Error (MAPE).

1.4 Plan of execution

In this project, we aim to provide the user with an estimated closing price for the upcoming day, given the long-term historical data. For this, we have explored various approaches as mentioned in the previous section.

For our project, we considered a couple of working environments like Matlab, Google Colab, and local Python IDEs. Due to the restrictive computing power of the local machines available to us, we decided to move forward with the Google Colab environment. In addition to our familiarity with Python, it also enables us to retrieve historical stock data from various online sources like Yahoo Finance in our case, through Pandas library.

Initially, we tried to get accustomed to the stock market and price movements. Further, we studied candle-stick chart patterns and technical indicators. During this, we came across a library (with a python wrapper) named TA-Lib, which provides the user with functions to identify the chart patterns. Using this, we tried to recognize chart patterns on a given date.

Nevertheless, since our main objective was to utilize machine learning approaches in our work, we decided to go ahead without using the above-mentioned Library. Since our previous project (Mini-Project) was based on Convolutional Neural Networks, we decided to consider the same approach for this project. Hence, we came up with a Deep CNN model which looks at historical stock price data and predicts the change in the stock price of the next day.

Moving forward, encouraged by our learnings from the Machine Learning course in the previous semester, we explored other machine learning-based approaches and were able to apply a few of them namely, LSTM, RNN, and GRU. Through a trial and error approach, we decided upon the different configurations of our models and the values for certain Hyper-parameters.

For optimization and enhanced performance, we have created an ANN-based ensemble model which comprises LSTM, RNN, and GRU models. By trying out various train-test split ratios of the data-set and different values of hyper-parameters to tune the model, we intend to achieve better accuracy.

Chapter 2

Experimental Methods and Results

We have built five working models using DCNN, LSTM, RNN, GRU networks, and an ensemble approach that combines LSTM, RNN, and GRU networks. Of these, the DCNN approach is based on CNN pattern extraction, whereas the other four machine learning models use direct historical data. We have used the historical stock-price data from Yahoo Finance.

2.1 Methods

2.1.1 DCNN Approach

A Deep Convolutional Neural Network is a Deep Learning algorithm that can take in an input image, and extract important features by assigning weights and biases. The architecture of a DCNN is analogous to that of the connectivity pattern of neurons in the human brain. The pre-processing required in the DCNN is much higher as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, DCNN can learn these characteristics. Similar to how humans learn bearish and bullish patterns in a candle-stick chart, our model also learns in the same way. Our model inputs images of 150x150 resolution and consists of 4 convolutional layers and 3 dense layers. Moreover, we have also used Max Pooling, Batch Normalization, and activation functions like ReLu and Softmax.

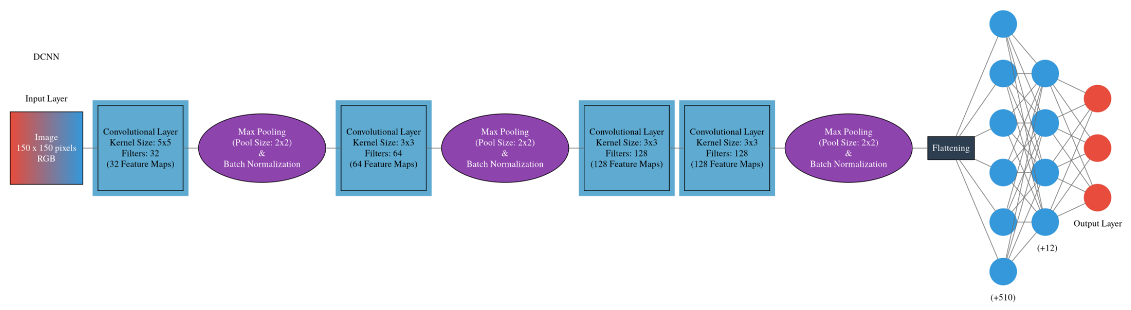

Figure 2.1: DCNN Model

2.1.2 LSTM Approach

An LSTM processes data passing on information as it propagates forward. LSTM cell and its operations are used to allow the LSTM to keep or forget information. The core concept of LSTM's is the cell state, and its various gates. The cell state act as a transport highway that transfers relative information down the sequence chain. It acts as the "memory" of the network. The cell state, in theory, can carry relevant information throughout the processing of the sequence. So even information from the earlier time steps can make its way to later time steps, reducing the effects of short-term memory. As the cell state goes on its journey, information get's added or removed to the cell state via gates. The gates are different neural networks that decide which information is allowed on the cell state. The gates can learn what information is relevant to keep or forget during training. We have considered close price data for a window of 4 days as input to our model, and the data of the upcoming 5^{th} day as the target value. The model consists of 4 LSTM layers with corresponding dropout layers, Tanh activation function, and 1 dense output layer.

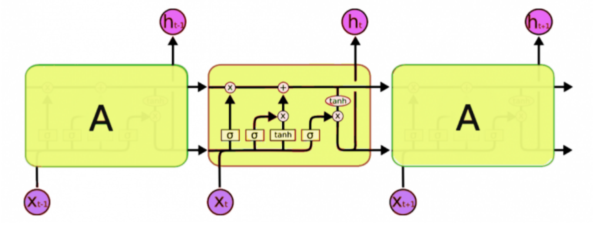

Figure 2.2: LSTM Network *(Illustration)*

2.1.3 RNN Approach

Recurrent Neural Network(RNN) is a type of Neural Network where the outputs from previous steps are fed as input to the current step. The most important feature of RNN is the hidden state, which remembers some information about the sequence. RNN has a “memory” which remembers all information about what has been calculated. It uses the same parameters for each input as it performs the same task on all the inputs or hidden layers to produce the output. This reduces the complexity of parameters, unlike other neural networks. Our RNN model considers close prices for a window of 20 days as input to our model, and the data of the upcoming 21st day as the target value. The model consists of 2 RNN layers and 1 dense output layer.

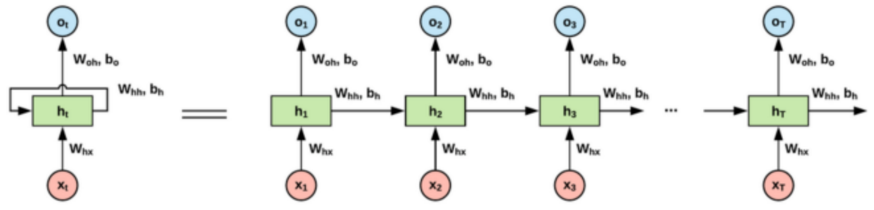

Figure 2.3: RNN Network *(Illustration)*

2.1.4 GRU Approach

GRU (Gated Recurrent Unit) aims to solve the vanishing gradient problem which comes with a standard recurrent neural network. GRU can also be considered as a variation on the LSTM because both are designed similarly and, in some cases, produce equally excellent results. Since GRUs are an improved version of the standard recurrent neural network, GRU uses a so-called, update gate and reset gate to solve the vanishing gradient problem of a standard RNN. These are two vectors that decide what information should be passed to the output. The special thing about them is that they can be trained to keep information from long ago, without washing it through time or remove information that is irrelevant to the prediction. Our GRU model considers close prices for a window of 8 days as input to our model, and the data of the upcoming 9th day as the target value. The model consists of 4 GRU layers with corresponding dropout layers, Tanh activation function, and 1 dense output layer.

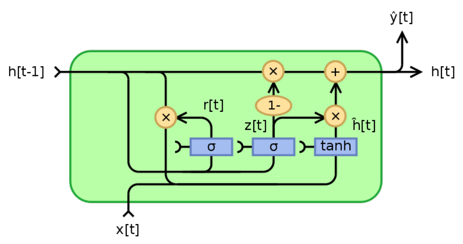

Figure 2.4: GRU Network *(Illustration)*

2.1.5 LSTM-RNN-GRU Ensemble Approach

The goal of any machine learning problem is to find a single model that will best predict our wanted outcome. Instead of making a single model and hoping it to be the best/most accurate predictor, ensemble methods take a myriad of models into account and automatically decides on the best outputs. The Ensemble approach is a machine learning technique that combines several base models to produce one optimal predictive model. In our approach, we have tried three methods to combine the results of the LSTM, RNN, and GRU approach, as listed below.

- Simple row-wise average of the predicted values, from the three models.
- Weighted average based on inverse ratios of mean squared errors of the three models.
- Ensemble, Artificial Neural Network-based approach, where a pre-trained neural network is used to decide on best prediction values, using the prediction values provided by the three models.

2.2 Algorithm

2.2.1 DCNN Algorithm

```
1: Import data of last 11 years from Yahoo Finance
2: Create candle-stick graphs for a window of 8 days with 2 days gap between each
   window
3: Plot simple moving average of 5 days window for additional features
4: Build the DCNN model
5: Train the model on the data-set
6: Test the model
```

2.2.2 LSTM Algorithm

```
1: Import data of last 11 years from Yahoo Finance
2: Build the LSTM model
3: Train the model on open, close, high, low and volume data for a window of 4
   days with $5^{th}$ day data as the target
4: Test the model on the next month values
5: Plot the predicted and actual values
```

2.2.3 RNN Algorithm

```
1: Import data of last 11 years from Yahoo Finance
2: Build the RNN model
3: Train the model on close price data of 20 days with $21^{st}$ day data as the target
4: Test the model on the next month values
5: Plot the predicted and actual values
```

2.2.4 GRU Algorithm

```
1: Import data of last 11 years from Yahoo Finance
2: Build the GRU model
3: Train the model on close price data of 8 days with $9^{th}$ day data as the target
4: Test the model on the next month values
5: Plot the predicted and actual values
```

2.2.5 Ensemble Algorithm

```
1: Import data of last 11 years from Yahoo Finance
2: Split the data into three sections, based on periods: First 7 years, next 3 years,
   and last 7 months
3: Use the data from the first section (first 7 years) to train the LSTM, RNN, and
   GRU models individually
4: Predict the values for the second section (next 3 years) and use them with the
   actual values for the same to train a simple Multi-layered ANN
5: Re-train the LSTM, RNN, and GRU models individually on the remaining data
   from the second section (next 3 years)
6: Predict stock prices for the last 7 month period, using all the 3 models
7: Combine the prediction values from LSTM, RNN, and GRU models into a Data-
   Frame
8: Use the ANN model to predict stock prices, by using the prediction values of
   the three models as input
```

2.3 Results

2.3.1 DCNN Results (general model for all tickers)

Table 2.1: Model Performance

Ticker Symbol	Accuracy
SBIN.NS	81.07
BANKBARODA.NS	75.92
AXISBANK.NS	78.39
FEDERALBNK.NS	78.39
ICICIBANK.NS	81.44
INDUSINDBK.NS	80.64
PNB.NS	76.65
HDFCBANK.NS	92.24
KOTAKBANK.NS	86.73

2.3.2 LSTM Results

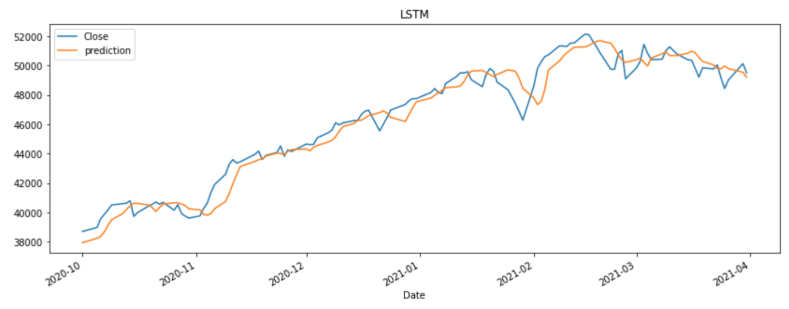

Figure 2.5: LSTM Performance

2.3.3 RNN Results

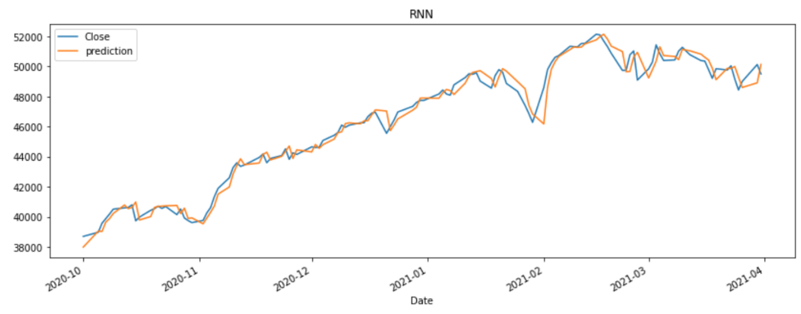

Figure 2.6: RNN Performance

2.3.4 GRU Results

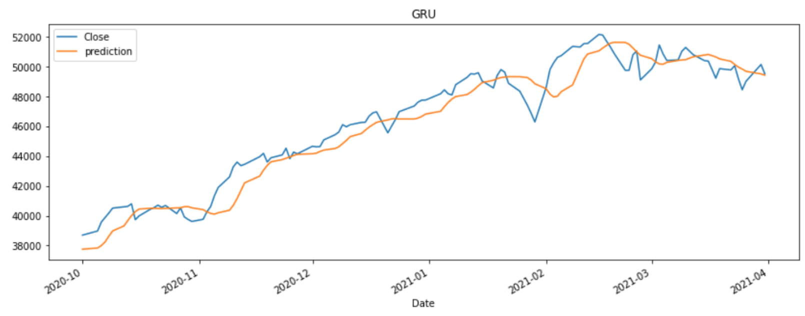

Figure 2.7: GRU Performance

2.3.5 Ensemble Approach

Simple Average

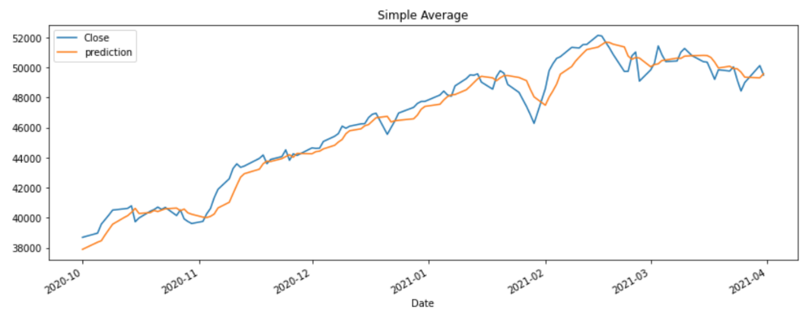

Figure 2.8: Simple Average

Weighted Average

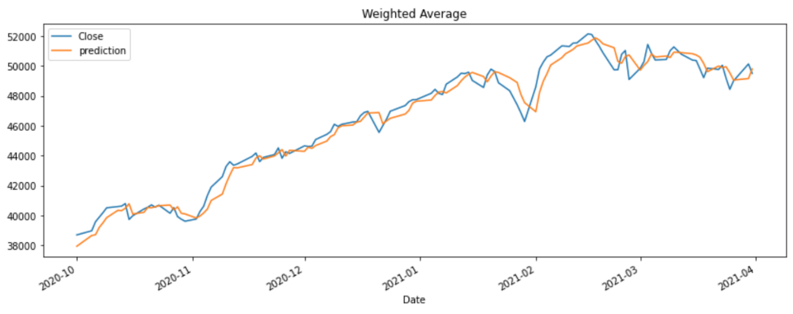

Figure 2.9: Weighted Average

Ensemble Results

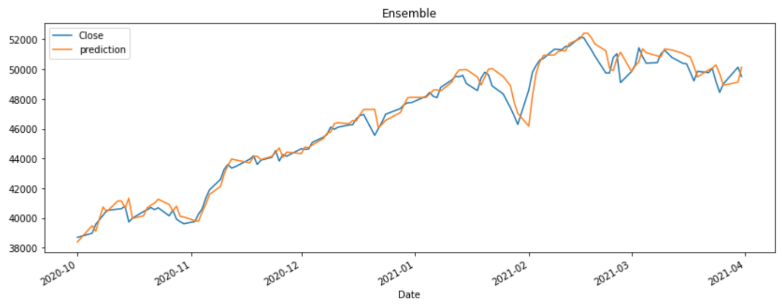

Figure 2.10: Ensemble Performance

Chapter 3

Discussion, Conclusion and Future Possibilities

In this project, we have tried to build and compare the performance of various machine learning models for stock price prediction. Currently, the GRU, LSTM, RNN, and Ensemble models are tested on the historical data of Sensex, whereas the DCNN model used several stock price data of which performed considerably better on the HDFC Bank data.

3.0.1 Performance Comparison of the Models

- DCNN Model:
  - We have considered the Banking Sector stocks as they have higher volatility (beta value) as compared to many other sectors and also give a good idea about the economy's health.
  - According to the performance table, Bank of Baroda stock has the lowest prediction accuracy (75.92) and HDFC Bank stock has the highest prediction accuracy (92.24).
  - Here, the prediction accuracy depends on the trends of the stock. For instance, the Bank of Baroda stock trends vary significantly in both directions, whereas stocks like those of HDFC Bank majorly follow a single trend in a particular direction. Hence, the difference in the prediction accuracy.

- LSTM vs RNN vs GRU vs Ensemble Models:

Table 3.1: Model Performances

Model Name	Sum of MSEs (Norm. data)	Avg. MSE (%)	Peak Error (%)
LSTM	0.00043386	16.8328	31.0494
RNN	0.00301170	14.4353	33.0459
GRU	0.00022649	20.6766	37.9117
Simple Average	0.00080520	16.0973	29.6835
Weighted Average	0.00030862	18.5452	34.3259
Ensemble (ANN)	0.00899516	18.9642	57.4312

  - Here, the RNN model has the lowest error whereas GRU has the highest error.
  - Here, because of the volatility of the stocks, the feature of long-term memory in LSTM and GRU may prove to be a disadvantage, whereas the RNN model having a short-term memory component may give better results.
  - The common scenario across all the models is the lag of one or more days in the predicted stock prices.
  - The visualization of Ensemble (ANN) displays better synchronization of price trends than the weighted average approach, although it is the reverse case when considering the sum of MSEs.
  - The RNN and Ensemble models followed the trends more accurately than others, except major deviations (caused by unforeseen circumstances) in the prices.

3.0.2 Future Possibilities

- The issue of lag in the prediction of stock prices has been around for some time now. The removal of such a lag is itself an area of research being pursued by many at present.

- ARIMA (Auto-Regressive Integrated Moving Average) is another strategy being widely used for stock price prediction. It is a class of models that ‘explains’ a given time series based on its past values, that is, its lags and the lagged forecast errors, so that equation can be used to forecast future values.

- Q-Learning which uses Reinforcement Learning and Sentiment Analysis from social media which may predict stock movement more accurately using historical data. It uses a Partially Observable Markov Decision Process for deciding on 3 actions which are hold, sell or buy. Using social media comments and tweets analysis regarding a stock along with the reinforcement model, another technique can be developed to predict stock prices with more accuracy.

- FLANN and Cascaded FLANN (CFLANN) are other machine learning approaches that are being considered for time-series prediction.

- In addition to historical time-series data, company fundamentals can also be incorporated for the task of stock price prediction. CANSLIM, which was developed by William J. O'Neil, is a system for selecting growth stocks using a combination of fundamental and technical analysis techniques.

3.0.3 Conclusion

In our project, we have primarily focused on technical analysis using machine learning approaches. Our project shows that there is some validity in predicting stock trends using historical data. This is not to say that this algorithm would make anyone rich, but it may be useful for trading analysis. These algorithms could perhaps be used as a buying or selling signal or it could be used to give confidence to a trader's prediction of stock prices.

Based on the outcomes of our project, we suggest that the RNN model performs significantly better as compared to the GRU, LSTM, and Ensemble models. Further, we can use this algorithm for developing various automated trading algorithms like a trading bot.

Since there are various other machine learning approaches as well as statistical approaches, we cannot state with confidence, which model has the best performance with regards to accurate prediction of the stock price for the next day.

Lastly, we need to keep in mind that there can be a considerable swing in prices due to unforeseen events like natural disasters, wars, financial crisis or any small incidents which occur time-to-time, and no prediction model or approach can be full-proof.

Appendix

3.1 DCNN Code

```
!pip install mplfinance
from pandas_datareader import data
from numpy import genfromtxt
import matplotlib.pyplot as plt
import mplfinance.original_flavor as mpl_finance
import numpy as np
import uuid

modes_list = ['next__day', 'next_3_days', 'next_5_days']
mode = modes_list[0]

#to create folders in "test" folder based on ticker_list

import os
ticker_list = ["SBIN.NS", "BANKBARODA.NS", "AXISBANK.NS", "
    ↪ FEDERALBNK.NS", "ICICIBANK.NS", "INDUSINDBK.NS", "PNB.NS", "
    ↪ HDFCBANK.NS", "KOTAKBANK.NS"]
for t in ticker_list:
  os.mkdir('data2/' + mode + '/test/' + t)
  os.mkdir('data2/' + mode + '/test/' + t + '/buy')
  os.mkdir('data2/' + mode + '/test/' + t + '/sell')
  os.mkdir('data2/' + mode + '/test/' + t + '/no_change')

ticker_list = ["SBIN.NS", "BANKBARODA.NS", "AXISBANK.NS", "
    ↪ FEDERALBNK.NS", "ICICIBANK.NS", "INDUSINDBK.NS", "PNB.NS", "
    ↪ HDFCBANK.NS", "KOTAKBANK.NS"]
print("CREATING␣TEST␣DATA␣:")
for t in ticker_list:
  print("␣␣␣GET␣DATA:␣TICKER␣=␣", t)
  df = data.DataReader(t, start='2010-1-1', end='2021-04-21',
      ↪ data_source='yahoo')

  buy_dir = 'data2/' + mode + '/test/' + t + '/buy/'
  sell_dir = 'data2/' + mode + '/test/' + t + '/sell/'
  no_change_dir = 'data2/' + mode + '/test/' + t + '/no_change/'
  change_thres = 0.025
```

```
def convolve_sma(array, period):
    return np.convolve(array, np.ones((period,))/period, mode='
       ↪ valid')

def graphwerk(start, finish):
    open, high, low, close, volume, date = [], [], [], [], [], []
    for _ in range(finish-start):
        open.append(df['Open'][start])
        high.append(df['High'][start])
        low.append(df['Low'][start])
        close.append(df['Close'][start])
        volume.append(df['Volume'][start])
        date.append(df.index[start])
        start = start + 1

    if mode == modes_list[0]:
      close_next = float(df['Close'][finish])

    elif mode == modes_list[1]:
      next_3_avg = []
      for i in range(3):
        next_3_avg.append(float(df['Close'][finish + i]))
      next_3_avg = np.average(next_3_avg)
      close_next = next_3_avg

    elif mode == modes_list[2]:
      next_5_avg = []
      for i in range(5):
        next_5_avg.append(float(df['Close'][finish + i]))
      next_5_avg = np.average(next_5_avg)
      close_next = next_5_avg

    sma = convolve_sma(close, 5)
    smb = list(sma)
    diff = sma[-1] - sma[-2]

    for x in range(len(close)-len(smb)):
        smb.append(smb[-1]+diff)
    fig = plt.figure(num=1, figsize=(3, 3), dpi=50, facecolor='w',
       ↪ edgecolor='k')
    dx = fig.add_subplot(111)
    mpl_finance.candlestick2_ochl(dx,open, close, high, low, width
       ↪ =1.5, colorup='g', colordown='r', alpha=0.5)

    plt.autoscale()
    plt.plot(smb, color="blue", linewidth=10, alpha=0.5)
    plt.axis('off')
    comp_ratio = close_next / close[-1]

    if close[-1] > close_next and abs((close[-1]-close_next)/close
       ↪ [-1]) > change_thres :
           plt.savefig(sell_dir + "diff=" + str(round(100*abs((
              ↪ close[-1]-close_next)/close[-1]),3)) + "___" +
              ↪ str(uuid.uuid4()) +'.jpg', bbox_inches='tight')

    elif abs((close[-1]-close_next)/close[-1]) > change_thres:
           plt.savefig(buy_dir + "diff=" + str(round(100*abs((close
              ↪ [-1]-close_next)/close[-1]),3)) + "___" + str(
              ↪ uuid.uuid4()) +'.jpg', bbox_inches='tight')
    else:
           plt.savefig(no_change_dir + "diff=" + str(round(100*abs
              ↪ ((close[-1]-close_next)/close[-1]),3)) + "___" +
              ↪ str(uuid.uuid4())+'.jpg', bbox_inches='tight')

    open.clear()
    close.clear()
    volume.clear()
    high.clear()
    low.clear()
    plt.cla()
    plt.clf()

iter_count = int(len(df)/4)
iter = 0

for x in range(len(df) - 4):
  if iter + 28 >= len(df):
    break
  graphwerk(iter, iter + 8)
```

```
    iter = iter + 2

#merge all folders from "test" and create common "buy", "sell" and
   ↪ "no_change" folders
from distutils.dir_util import copy_tree
import os
print("CREATING␣TRAIN␣DATA␣:")
for t in ticker_list:
  for sclass in ['buy', 'sell', 'no_change']:
    copy_tree(str(os.getcwd()) + '/data2/' + mode + '/test/' + t + '/
       ↪ ' + sclass + '/', str(os.getcwd()) + '/data2/' + mode + '/
       ↪ train/' + sclass + '/')

"""# Training"""

import os
import sys
from tensorflow import keras
import tensorflow as tf
from tensorflow.keras import optimizers, initializers
from tensorflow.keras.layers import Dropout, Flatten, Dense,
   ↪ Activation, BatchNormalization
from tensorflow.keras.layers import Convolution2D, MaxPooling2D,
   ↪ Conv2D
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ModelCheckpoint

img_width, img_height = 150, 150
datagen_train_validation = ImageDataGenerator(
                                   rescale=1./255,
                                   horizontal_flip = False,
                                   validation_split = 0.2)

train_generator = datagen_train_validation.flow_from_directory(
                                  'data2/' + mode + '/train/',
                                  target_size = (img_width, img_height),
                                  seed = 5,
                                  class_mode = 'categorical',
                                  subset='training')
```

```
validation_generator = datagen_train_validation.flow_from_directory(
                              'data2/' + mode + '/train/',
                              target_size = (img_width, img_height),
                              seed = 5,
                              class_mode = 'categorical',
                              subset = 'validation')

model = Sequential()
model.add(Conv2D(filters = 32, kernel_size = (5, 5), activation='
    ↪ relu', padding = 'same', input_shape = (150, 150 , 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())

model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu',
    ↪ padding = 'same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())

model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu',
    ↪  padding = 'same'))
model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu',
    ↪  padding = 'same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(3, activation='softmax'))

model.summary()
model.compile(loss='categorical_crossentropy',
                    optimizer='adam',
                    metrics=['accuracy', tf.keras.metrics.
                        ↪ TruePositives(name="TP"), tf.keras.metrics
                        ↪ .TrueNegatives(name="TN"), tf.keras.
                        ↪ metrics.FalsePositives(name="FP"), tf.
                        ↪ keras.metrics.FalseNegatives(name="FN")])

model = keras.models.load_model(str(os.getcwd()) + '/models/' + mode
    ↪ )
```

```
history = model.fit(
    x = train_generator,
    steps_per_epoch=16,
    epochs = 8,
    shuffle=False,
    validation_data=validation_generator
    )


model.save(str(os.getcwd()) + '/models/' + mode + '__banks_26Apr_3')

"""# Testing on Individual Tickers"""

modes_list = ['next__day', 'next_3_days', 'next_5_days']
mode = modes_list[0]
model = keras.models.load_model(str(os.getcwd()) + '/models/' + mode
    ↪  + '__bank_8')

ticker_list = ["SBIN.NS", "BANKBARODA.NS", "AXISBANK.NS", "
    ↪ FEDERALBNK.NS", "ICICIBANK.NS", "INDUSINDBK.NS", "PNB.NS", "
    ↪ HDFCBANK.NS", "KOTAKBANK.NS"]

filenames_dict, predict_dict = {}, {}

for t in ticker_list:
  print("Testing␣for␣:␣", t, "␣␣mode␣:␣", mode )
  img_width, img_height = 150, 150
  datagen_test = ImageDataGenerator(
                              rescale=1./255,
                              horizontal_flip = False,
                              validation_split = False)

  test_generator = datagen_test.flow_from_directory(
                                'data2/' + mode + '/test/' + t,
                                target_size = (img_width, img_height
                                   ↪ ),
                                seed = 5,
                                shuffle = False,
                                class_mode = 'categorical',
                                subset=None)
  filenames = test_generator.filenames
  predict = model.evaluate(x=test_generator)
  filenames_dict[t] = filenames
  predict_dict[t] = predict
```

3.2 LSTM Code

```
import pandas as pd
import numpy as np
import pandas_datareader as pdr
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, SimpleRNN
import os

os.environ['PYTHONHASHSEED'] = '0'
np.random.seed(0)
tf.random.set_seed(0)

all_data = pdr.DataReader('^BSESN', start='2010-01-01', end='
    ↪ 2021-03-31', data_source='yahoo')

"""# Train-Test Split"""

print("There␣are␣"+ str(all_data[:'2017-01']["Close"].shape[0]) + "␣
    ↪ observations␣in␣the␣training␣data␣1")
print("There␣are␣"+ str(all_data['2017-02':'2020-09']["Close"].shape
    ↪ [0]) + "␣observations␣in␣the␣training␣data␣2")
print("There␣are␣"+ str(all_data['2020-10':]["Close"].shape[0]) + "␣
    ↪ observations␣in␣the␣test␣data")

def ts_train_test_no_split(all_data,time_steps,for_periods,do_norm=
    ↪ False):
    # create training and test set
    ts_train = (all_data[:'2020-09'])['Close'].values.reshape(-1,1)
    ts_test = (all_data['2020-10':])['Close'].values.reshape(-1,1)
    ts_train_len = len(ts_train)
    ts_test_len = len(ts_test)
```

```
# scale the data
if do_norm == True:
  from sklearn.preprocessing import MinMaxScaler
  sc = MinMaxScaler(feature_range=(0,1))
  ts_train_scaled = sc.fit_transform(ts_train)
else:
  ts_train_scaled = ts_train.copy()
ts_train_scaled.reshape(-1,1)
# create training data of s samples and t time steps
X_train = []
y_train = []
for i in range(time_steps,ts_train_len-1):
    X_train.append(ts_train_scaled[i-time_steps:i,0])
    y_train.append(ts_train_scaled[i:i+for_periods,0])
X_train, y_train = np.array(X_train), np.array(y_train)

# Reshaping X_train for efficient modelling
X_train = np.reshape(X_train, (X_train.shape[0],X_train.shape
    ↪ [1],1))

inputs = all_data["Close"].values
inputs = inputs[len(inputs)-len(ts_test) - time_steps:] #
    ↪ selecting test data
inputs = inputs.reshape(-1,1)
if do_norm == True:
  sc = MinMaxScaler(feature_range=(0,1))
  inputs = sc.fit_transform(inputs)
else:
  sc = None

# Preparing X_test
X_test = []
for i in range(time_steps,ts_test_len+time_steps-for_periods):
    X_test.append(inputs[i-time_steps:i,0])

X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0],X_test.shape[1],1))

return X_train, y_train , X_test, sc
```

```python
def ts_train_test(all_data,time_steps,for_periods,do_norm=False):
    # create training and test set
    ts_train_1 = (all_data[:'2017-01'])['Close'].values.reshape(-1,1)
    ts_train_2 = (all_data['2017-02':'2020-09'])['Close'].values.
       ↪ reshape(-1,1)
    ts_test_1 = ts_train_2.copy()
    ts_test_2 = (all_data['2020-10':])['Close'].values.reshape(-1,1)
    ts_train_1_len = len(ts_train_1)
    ts_train_2_len = len(ts_train_2)
    ts_test_1_len = len(ts_train_2)
    ts_test_2_len = len(ts_test_2)

    # scale the data
    if do_norm == True:
      from sklearn.preprocessing import MinMaxScaler
      sc = MinMaxScaler(feature_range=(0,1))
      ts_train_scaled_1 = sc.fit_transform(ts_train_1)
    else:
      ts_train_scaled_1 = ts_train_1.copy()
    ts_train_scaled_1.reshape(-1,1)

    if do_norm == True:
      from sklearn.preprocessing import MinMaxScaler
      sc = MinMaxScaler(feature_range=(0,1))
      ts_train_scaled_2 = sc.fit_transform(ts_train_2)
    else:
      ts_train_scaled_2 = ts_train_2.copy()
    ts_train_scaled_2.reshape(-1,1)

    # create training data of s samples and t time steps
    X_train_1 = []
    y_train_1 = []
    for i in range(time_steps,ts_train_1_len-1):
        X_train_1.append(ts_train_scaled_1[i-time_steps:i,0])
        y_train_1.append(ts_train_scaled_1[i:i+for_periods,0])
    X_train_1, y_train_1 = np.array(X_train_1), np.array(y_train_1)

    X_train_2 = []
    y_train_2 = []
    for i in range(time_steps,ts_train_2_len-1):
        X_train_2.append(ts_train_scaled_2[i-time_steps:i,0])
    y_train_2.append(ts_train_scaled_2[i:i+for_periods,0])
X_train_2, y_train_2 = np.array(X_train_2), np.array(y_train_2)

# Reshaping X_train for efficient modelling
X_train_1 = np.reshape(X_train_1, (X_train_1.shape[0],X_train_1.
    ↪ shape[1],1))
X_train_2 = np.reshape(X_train_2, (X_train_2.shape[0],X_train_2.
    ↪ shape[1],1))

inputs_ts_test_1 = all_data["Close"].values
inputs_ts_test_1 = inputs_ts_test_1[len(inputs_ts_test_1) -
    ↪ ts_train_2_len - len(ts_test_2) - time_steps:]
inputs_ts_test_1 = inputs_ts_test_1.reshape(-1,1)
if do_norm == True:
  sc_ts_test_1 = MinMaxScaler(feature_range=(0,1))
  inputs_ts_test_1 = sc_ts_test_1.fit_transform(inputs_ts_test_1)
else:
  sc_ts_test_1 = None

# Preparing X_test
X_test_1 = []
for i in range(time_steps,ts_test_1_len + time_steps -
    ↪ for_periods):
    X_test_1.append(inputs_ts_test_1[i-time_steps:i,0])

X_test_1 = np.array(X_test_1)
X_test_1 = np.reshape(X_test_1, (X_test_1.shape[0],X_test_1.shape
    ↪ [1],1))


inputs_ts_test_2 = all_data["Close"].values
inputs_ts_test_2 = inputs_ts_test_2[len(inputs_ts_test_2) - len(
    ↪ ts_test_2) - time_steps:]
inputs_ts_test_2 = inputs_ts_test_2.reshape(-1,1)
if do_norm == True:
  sc_ts_test_2 = MinMaxScaler(feature_range=(0,1))
  inputs_ts_test_2 = sc_ts_test_2.fit_transform(inputs_ts_test_2)
else:
  sc_ts_test_2 = None

# Preparing X_test
```

```
    X_test_2 = []
    for i in range(time_steps, ts_test_2_len + time_steps -
        ↪ for_periods):
        X_test_2.append(inputs_ts_test_2[i-time_steps:i,0])

    X_test_2 = np.array(X_test_2)
    X_test_2 = np.reshape(X_test_2, (X_test_2.shape[0], X_test_2.
        ↪ shape[1], 1))


    return X_train_1, y_train_1, X_test_1, X_train_2, y_train_2,
        ↪ X_test_2, sc_ts_test_1, sc_ts_test_2


"""#Plot Function"""

def actual_pred_plot(preds, is_test_1, title=None):
    actual_pred = pd.DataFrame(columns = ['Close', 'prediction'])
    if is_test_1 == True:
      actual_pred['Close'] = all_data.loc['2017-02':'2020-09','Close'
          ↪ ][0:len(preds)]
    elif is_test_1 == False:
      actual_pred['Close'] = all_data.loc['2020-10':,'Close'][0:len(
          ↪ preds)]
    actual_pred['prediction'] = preds[:,0]

    from sklearn.preprocessing import MinMaxScaler
    sc1 = MinMaxScaler(feature_range=(0,1))
    scaled_actual_pred = sc1.fit_transform(actual_pred.values)

    from tensorflow.keras.metrics import MeanSquaredError
    m = MeanSquaredError()
    m.update_state(scaled_actual_pred[0], scaled_actual_pred[1])

    diff = np.subtract(scaled_actual_pred[1:,0], scaled_actual_pred
        ↪ [1:,1])
    std_dev_mse = np.std((diff / scaled_actual_pred[1:,0])*100)
    avg_error_perc = np.sum(np.abs(diff) / scaled_actual_pred[1:,0])
        ↪ *100/len(diff)
    return (scaled_actual_pred, m.result().numpy(), std_dev_mse,
       ↪ avg_error_perc, actual_pred.plot(title=title, figsize
       ↪ =(14,5)))

def actual_pred_plot_no_split(preds, title=None):
    actual_pred = pd.DataFrame(columns = ['Close', 'prediction'])
    actual_pred['Close'] = all_data.loc['2020-10':,'Close'][0:len(
       ↪ preds)]
    actual_pred['prediction'] = preds[:,0]

    from sklearn.preprocessing import MinMaxScaler
    sc1 = MinMaxScaler(feature_range=(0,1))
    scaled_actual_pred = sc1.fit_transform(actual_pred.values)

    from tensorflow.keras.metrics import MeanSquaredError
    m = MeanSquaredError()
    m.update_state(scaled_actual_pred[0], scaled_actual_pred[1])

    diff = np.subtract(scaled_actual_pred[1:,0], scaled_actual_pred
       ↪ [1:,1])
    std_dev_mse = np.std((diff / scaled_actual_pred[1:,0])*100)
    avg_error_perc = np.sum(np.abs(diff) / scaled_actual_pred[1:,0])
       ↪ *100/len(diff)

    return (actual_pred, m.result().numpy(), std_dev_mse,
       ↪ avg_error_perc, actual_pred.plot(title=title, figsize
       ↪ =(14,5)))

"""# Normalized LSTM"""

def LSTM_model_regularization(X_train, y_train, X_test, sc,
    ↪ no_of_outputs, regressor=False):
  from keras.models import Sequential
  from keras.layers import Dense
  from keras.layers import LSTM
  from keras.layers import Dropout

  if regressor == False:
    regressor = Sequential()
    regressor.add(LSTM(units = 50, return_sequences = True,
       ↪ input_shape = (X_train.shape[1], 1)))
    regressor.add(Dropout(0.2))
    regressor.add(LSTM(units = 50, return_sequences = True))
    regressor.add(Dropout(0.2))
    regressor.add(LSTM(units = 50, return_sequences = True))
    regressor.add(Dropout(0.2))
    regressor.add(LSTM(units = 50))
    regressor.add(Dropout(0.2))
    regressor.add(Dense(units = no_of_outputs))
    regressor.compile(optimizer = 'adam', loss = 'mean_squared_error'
       ↪ )

  regressor.fit(X_train, y_train, epochs = 50, batch_size = 32,
     ↪ verbose=0)
  LSTM_predictions = regressor.predict(X_test)
  LSTM_predictions = sc.inverse_transform(LSTM_predictions)
  return regressor, LSTM_predictions

no_of_inputs, no_of_outputs = 4, 1
X_train, y_train, X_test, sc = ts_train_test_no_split(all_data,
   ↪ no_of_inputs, no_of_outputs,True)
LSTM_model, LSTM_predictions = LSTM_model_regularization(X_train,
   ↪ y_train, X_test, sc, 1)

actual_pred_lstm, error_lstm, lstm_std_mse, lstm_avg_error_perc, _ =
   ↪  actual_pred_plot_no_split(LSTM_predictions, 'LSTM')
print("LSTM␣Error␣=␣", "{:.12f}".format(error_lstm),"␣␣LSTM␣STD␣MSE␣
   ↪ =␣", lstm_std_mse,"␣␣LSTM␣AVG␣Error␣=␣", lstm_avg_error_perc)
```

3.3 RNN Code

```
def rnn_model(X_train, y_train, X_test, sc, no_of_outputs, rnn_model
   ↪ =False):
    if rnn_model == False:
      rnn_model = Sequential()
      rnn_model.add(SimpleRNN(32, return_sequences=True))
      rnn_model.add(SimpleRNN(32))
      rnn_model.add(Dense(no_of_outputs))
      rnn_model.compile(optimizer='adam', loss='mean_squared_error')
    rnn_model.fit(X_train, y_train, epochs=100, batch_size=50,
        ↪ verbose=0)
    rnn_predictions = rnn_model.predict(X_test)
    from sklearn.preprocessing import MinMaxScaler
    rnn_predictions = sc.inverse_transform(rnn_predictions)

    return rnn_model, rnn_predictions

no_of_inputs, no_of_outputs = 20, 1
X_train, y_train, X_test, sc = ts_train_test_no_split(all_data,
    ↪ no_of_inputs, no_of_outputs,True)
rnn_model, rnn_predictions = rnn_model(X_train, y_train, X_test, sc,
    ↪  no_of_outputs)

actual_pred_rnn, error_rnn, rnn_std_mse, rnn_avg_error_perc, _ =
    ↪ actual_pred_plot_no_split(rnn_predictions, 'RNN')
print("RNN␣Error␣␣=␣", "{:.12f}".format(error_rnn), "␣␣RNN␣STD␣MSE␣␣
    ↪ =␣", rnn_std_mse, "␣␣RNN␣AVG␣Error␣=␣␣", rnn_avg_error_perc)
```

3.4 GRU Code

```
def GRU_model_regularization(X_train, y_train, X_test, sc,
    ↪ no_of_outputs, GRU_model=False):
    from keras.models import Sequential
    from keras.layers import Dense, SimpleRNN, GRU
    from keras.optimizers import SGD
    from keras.layers import Dropout

    if GRU_model == False:
      GRU_model = Sequential()
      GRU_model.add(GRU(50, return_sequences=True, input_shape=(
          ↪ X_train.shape[1],1), activation='tanh'))
      GRU_model.add(Dropout(0.2))
      GRU_model.add(GRU(50, return_sequences=True, activation='tanh')
          ↪ )
      GRU_model.add(Dropout(0.2))
      GRU_model.add(GRU(50, return_sequences=True, activation='tanh')
          ↪ )
      GRU_model.add(Dropout(0.2))
      GRU_model.add(GRU(50, activation='tanh'))
      GRU_model.add(Dropout(0.2))
      GRU_model.add(Dense(no_of_outputs))
      GRU_model.compile(optimizer=SGD(lr=0.01, decay=1e-7, momentum
         ↪ =0.9, nesterov=False),loss='mean_squared_error')

    GRU_model.fit(X_train,y_train,epochs=50,batch_size=150, verbose
       ↪ =0)
    GRU_predictions = GRU_model.predict(X_test)
    GRU_predictions = sc.inverse_transform(GRU_predictions)
    return GRU_model, GRU_predictions

no_of_inputs, no_of_outputs = 8, 1
X_train, y_train, X_test, sc = ts_train_test_no_split(all_data,
   ↪ no_of_inputs, no_of_outputs,True)
GRU_model, GRU_predictions = GRU_model_regularization(X_train,
   ↪ y_train, X_test, sc, 1)

actual_pred_gru, error_gru, gru_std_mse, gru_avg_error_perc, _ =
   ↪ actual_pred_plot_no_split(GRU_predictions, 'GRU')
print("GRU␣Error␣␣=␣", "{:.12f}".format(error_gru), "␣␣GRU␣STD␣MSE␣␣
   ↪ =␣", gru_std_mse, "␣␣GRU␣AVG␣Error␣=␣␣", gru_avg_error_perc)
```

3.5 Ensemble Code

```
combined_predictions_ = pd.DataFrame(columns = ['rnn_prediction', '
   ↪ gru_prediction', 'lstm_prediction', 'weighted_avg'])
combined_predictions_['rnn_prediction'] = rnn_predictions.reshape
   ↪ (-1)
combined_predictions_['gru_prediction'] = GRU_predictions.reshape
   ↪ (-1,1)
combined_predictions_['lstm_prediction'] = LSTM_predictions.reshape
   ↪ (-1,1)

actual_pred_avg, error_avg, std_mse, avg_error_perc, _ =
   ↪ actual_pred_plot_no_split(combined_predictions_[['
   ↪ rnn_prediction', 'gru_prediction', 'lstm_prediction']].values.
   ↪ mean(axis=1).reshape(-1,1), 'Simple␣Average')
print(error_avg, std_mse, avg_error_perc)
```

```
sum_of_sq_errors_inv = (1/error_rnn) + (1/error_gru) + (1/error_lstm
    ↪ )
combined_predictions_['weighted_avg'] = ((combined_predictions_['
    ↪ rnn_prediction'].values)*((1/error_rnn)/sum_of_sq_errors_inv))
    ↪ \
                                    + ((combined_predictions_['
                                        ↪ gru_prediction'].values)
                                        ↪ *((1/error_gru)/
                                        ↪ sum_of_sq_errors_inv))\
                                    + ((combined_predictions_['
                                        ↪ lstm_prediction'].values)
                                        ↪ *((1/error_lstm)/
                                        ↪ sum_of_sq_errors_inv))

actual_pred_weighted_avg, error_weighted_avg, std_mse,
    ↪ avg_error_perc, _ = actual_pred_plot_no_split(
    ↪ combined_predictions_['weighted_avg'].values.reshape(-1,1), '
    ↪ Weighted␣Average')
print(error_weighted_avg, std_mse, avg_error_perc)

#train the 3 models using 2010-2017 data and predict values for
    ↪ 2017-2020 using appropriate test data
no_of_inputs_rnn, no_of_outputs_rnn = 20, 1
X_train_1_rnn, y_train_1_rnn, X_test_1_rnn, X_train_2_rnn,
    ↪ y_train_2_rnn, X_test_2_rnn, sc_ts_test_1_rnn,
    ↪ sc_ts_test_2_rnn = ts_train_test(all_data, no_of_inputs_rnn,
    ↪ no_of_outputs_rnn,True)
rnn_model_1, rnn_predictions_1 = rnn_model(X_train_1_rnn,
    ↪ y_train_1_rnn, X_test_1_rnn, sc_ts_test_1_rnn,
    ↪ no_of_outputs_rnn)

no_of_inputs_gru, no_of_outputs_gru = 8, 1
X_train_1_gru, y_train_1_gru, X_test_1_gru, X_train_2_gru,
    ↪ y_train_2_gru, X_test_2_gru, sc_ts_test_1_gru,
    ↪ sc_ts_test_2_gru = ts_train_test(all_data, no_of_inputs_gru,
    ↪ no_of_outputs_gru,True)
GRU_model_1, GRU_predictions_1 = GRU_model_regularization(
    ↪ X_train_1_gru, y_train_1_gru, X_test_1_gru, sc_ts_test_1_gru,
    ↪ no_of_outputs_gru)
```

```
no_of_inputs_lstm, no_of_outputs_lstm = 4, 1
X_train_1_lstm, y_train_1_lstm, X_test_1_lstm, X_train_2_lstm,
   ↪ y_train_2_lstm, X_test_2_lstm, sc_ts_test_1_lstm,
   ↪ sc_ts_test_2_lstm = ts_train_test(all_data, no_of_inputs_lstm,
   ↪  no_of_outputs_lstm,True)
LSTM_model_1, LSTM_predictions_1 = LSTM_model_regularization(
   ↪ X_train_1_lstm, y_train_1_lstm, X_test_1_lstm,
   ↪ sc_ts_test_1_lstm, no_of_outputs_lstm)

combined_predictions_1 = pd.DataFrame(columns = ['rnn_prediction', '
   ↪ gru_prediction', 'lstm_prediction'])
combined_predictions_1['rnn_prediction'] = rnn_predictions_1.reshape
   ↪ (-1)
combined_predictions_1['gru_prediction'] = GRU_predictions_1.reshape
   ↪ (-1,1)
combined_predictions_1['lstm_prediction'] = LSTM_predictions_1.
   ↪ reshape(-1,1)

actual_pred_avg, error_avg, std_mse, avg_error_perc, _ =
   ↪ actual_pred_plot(combined_predictions_1.values.mean(axis=1).
   ↪ reshape(-1,1), True)

combined_ann = Sequential()
combined_ann.add(Dense(16, input_dim=3, activation='relu')) #3
   ↪ inputs are predicted values from LSTM, RNN and GRU models
combined_ann.add(Dense(16, activation='relu'))
combined_ann.add(Dense(32, activation='relu'))
combined_ann.add(Dense(1, activation='linear'))
combined_ann.compile(loss='mean_squared_error', optimizer='adam')
combined_ann.summary()

#train ANN based on x=predicted values from 2017-2020 y=actual
   ↪ values for 2017-2020
combined_ann.fit(x=combined_predictions_1, y=all_data['2017-02':'
   ↪ 2020-09']['Close'][0:len(combined_predictions_1)].values,
   ↪ epochs=500, verbose=0)

#train the previously trained model (on 2010-2017 data), again
   ↪ using 2017-2020 data and predict values for 2020-2021
# *use the previously trained model and fit more data on it
```

```
rnn_model_2, rnn_predictions_2 = rnn_model(X_train_2_rnn,
    ↪ y_train_2_rnn, X_test_2_rnn, sc_ts_test_2_rnn,
    ↪ no_of_outputs_rnn, rnn_model=rnn_model_1)
GRU_model_2, GRU_predictions_2 = GRU_model_regularization(
    ↪ X_train_2_gru, y_train_2_gru, X_test_2_gru, sc_ts_test_2_gru,
    ↪ no_of_outputs_gru, GRU_model_1)
LSTM_model_2, LSTM_predictions_2 = LSTM_model_regularization(
    ↪ X_train_2_lstm, y_train_2_lstm, X_test_2_lstm,
    ↪ sc_ts_test_2_lstm, no_of_outputs_lstm, LSTM_model_1)

combined_predictions_2 = pd.DataFrame(columns = ['rnn_prediction', '
    ↪ gru_prediction', 'lstm_prediction','ann_prediction'])
combined_predictions_2['rnn_prediction'] = rnn_predictions_2.reshape
    ↪ (-1)
combined_predictions_2['gru_prediction'] = GRU_predictions_2.reshape
    ↪ (-1,1)
combined_predictions_2['lstm_prediction'] = LSTM_predictions_2.
    ↪ reshape(-1,1)

#use predictions for 2020-2021, from the 3 models and input it in
    ↪ the ANN model (trained on 2017-2020 prediction errors)
#and use the output as final ensemble predicion output
combined_predictions_2['ann_prediction'] = combined_ann.predict(
    ↪ combined_predictions_2[['rnn_prediction', 'gru_prediction', '
    ↪ lstm_prediction']].values)

actual_pred_avg, error_, std_mse, avg_error_perc, _ =
    ↪ actual_pred_plot(combined_predictions_2['ann_prediction'].
    ↪ values.reshape(-1,1), False, title='Ensemble')
print(error_, std_mse, avg_error_perc)
```

Bibliography

[1] Kannan, Sekar, Sathik, and P. Arumugam, ”Financial Stock Market Forecast using Data Mining Techniques”, *Proceedings of the International Multi-Conference of Engineers and Computer Scientists Vol 1, IMECS*, 2010.

[2] Yao, Jingtao and Tan, Chew Lim, ”A case study on using neural networks to perform technical forecasting of forex”, *Neurocomputing Vol 34*, pp. 79-98, 2000.

[3] Tiffany Hui-Kuangyu and Kun-Huang Huarng, ”A Neural network-based fuzzy time series model to improve forecasting”, *Elsevier*, pp. 3366-3372, 2010.

[4] Abhinandan Gupta, Dev Kumar Chaudhary and Tanupriya Choudhury, ”Stock Prediction using Functional Link Artificial Neural Network (FLANN)”, *International Conference on Computational Intelligence and Networks, 2017*

[5] Mehar Vijh, Deeksha Chandola, Vinay Anand Tikkiwal and Arun Kumar, ”Stock Closing Price Prediction using Machine Learning Techniques”, *International Conference on Computational Intelligence and Data Science (ICCIDS)*, Procedia Computer Science 167 pp. 599–606, 2020.

[6] R. Sathya, Prateek Kulkarni, Momin Nawaf Khalil and Shishir Chandra Nigam, ”Stock Price Prediction using Reinforcement Learning and Feature Extraction”, *International Journal of Recent Technology and Engineering (IJRTE)*, ISSN: 2277-3878, Volume-8 Issue-6, March 2020