Predicting Stock Market time-series data using CNN-LSTM Neural Network model

Aadhitya A
Department of Information Technology
Madras Institute of Technology
Chennai, India
Email: aadhitya864@gmail.com

Rajapriya R
Department of Information Technology
Madras Institute of Technology
Chennai, India

Vineetha R S
Department of Information Technology
Madras Institute of Technology
Chennai, India

Anurag M Bagde
Department of Information Technology
Madras Institute of Technology
Chennai, India

***Abstract*—Stock market is often important as it represents the ownership claims on businesses. Without sufficient stocks, a company cannot perform well in finance. Predicting a stock market performance of a company is nearly hard because every time the prices of a company's stock keeps changing and not constant. Therefore, it's random to predict and pick the right company to buy or sell stocks. But if the previous performance of a company in stock market is known, then we can track the data and provide predictions (which is near accurate) to the stockholders to wisely take decisions on handling the stocks to a company. To handle this, many machine learning models have been invented but they didn't succeed due to many reasons like absence of advanced libraries, inaccuracy of model when made to train with real time data and much more. So, to track the patterns and the features of data, a CNN-LSTM Neural Network can be made. Recently, CNN is now used in Natural Language Processing (NLP) based applications, so by identifying the features from stock data and converting them into tensors, we can obtain the features and then send it to LSTM neural network to find the patterns and thereby predicting the stock market for given period of time. The accuracy of the CNN-LSTM NN model is found to be high even when allowed to train on real-time stock market data. This paper describes about the features of the CNN-LSTM model, experiments we made with the model like training with stock market datasets, performance comparison with other models and the end product we obtained at final stage.**

***Index Terms*—CNN-LSTM, Deep Learning, Prediction, Pattern Recognition**

I. Introduction

Stock Market (also called as share market) is a place where people or businessmen often look on their "shares" or ownership claims related to the company. Stock market is considered an important place in economy as a country's wealth is decided on it. The hardest task in stock market is predicting it because the data is not constant, always. In early days, many people tried to predict stocks using conventional methods but most of the time, failed. So the probability of predicting stocks correctly is very minimal. Today, Machine Learning (ML) techniques have been implemented to predict the company shares and thereby providing suggestions to stock holders to improvise the company's performance. But however, they are not accurate. This paper focuses on some of the works done in predicting stock market and a new method to follow CNN-LSTM Neural Network model approach to predict data for given time-series.

II. Related Work

Before the time of writing this paper, many have proposed and implemented various algorithms in order to predict stock market data. We did a literature survey to find some of the algorithms proposed and found some of the advantages, disadvantages present in those algorithms. Subhadra and Kalyana in [1] conducted analysis in various machine learning methods to predict stock market data and found that Random Forest performs good compared with Linear Regression and other algorithms, however the error percentage rises in the model when the input data is not smoothed in pre-processing stage. Pang, Zhou et al in [2] made comparisons with RNN and LSTM model and conducted experimental analysis, in which LSTM with Auto-Encoder module enabled (AELSTM) predicted most of the data but the implementation is done using old libraries and with real-time stock market data, thus the accuracy of the model was low. This was improved but revealed an important point when a model is made to train with real-time data. Uma and Kotrappa in [3] proposed a new method where LSTM with Log Bilinear layer on top of it. The model predicted most of the stock market data and turned out with high accuracy but it was proposed and not tested with real time data, also it was meant to predict data only during the time of COVID-19 and not beyond that. Kimoto, Yoda, Takeoka in [4] discussed a buying and selling timing prediction system based on modular neural network which converts the technical indexes and economic indexes into a space pattern to input to the neural networks. During the analysis phase, neural network model produced a higher correlation coefficient in comparison to multiple regression. The experiment did by Guresen, Kayakutlu and Daim in [5] evaluates the efficiency of dynamic artificial neural network (DAN2), multi-layer perceptron (MLP), and hybrid neural networks. The paper concludes by stating that the classic ANN model - MLP gives the most reliable results in forecasting time series while Hybrid methods failed to improve the forecast results. Nelson, Pereira and Oliveria in [6] proposed an LSTM network to predict future trends of stock prices in time steps of 15 minutes based on the price history, alongside technical analysis indicators. On average 55.9% accuracy was achieved in predicting whether the price of a particular stock may increase or not shortly in the future. Selvin, Menon, Soman et al in [7] experimented on three different deep learning models, namely CNN, RNN, and LSTM with a sliding window approach. Out of the three, CNN gave more accurate results than the other two models which is due to the reason that CNN uses only the information on the current window for predicting stock price. This allows CNN to understand the dynamic changes and patterns occurring in the current window. Conversely, RNN and LSTM use information from previous lags for predicting future instances. Hiransha, Gopalakrishnan and Soman in [8] conducted experiments to compare different Deep Learning models viz. ANN, MLP, LSTM, and RNN. ANN captured the pattern at the initial stage so did RNN but on reaching a certain time period both failed to identify the pattern. Same was the case with LSTM but CNN seemed to perform better compared to the other three networks even though several time periods showed less accuracy for the predicted values. Bansal, Hasija et al in [9] proposed an Intelligent decentralized Stock market model using the convergence of machine learning alongside DAG-based cryptocurrency. The ML model which is based on LSTM achieved an accuracy of 99.71% in prediction. The feature vector of stock for the company contained 4 parameter values i.e. 'open', 'close', 'low', and 'high' with batch size as 50 for 100 epochs.

III. Proposed Work

After we conducted the literature survey, we realised some of the key points to be taken while designing a ML model

- The model must be designed in such a way that it should parse real-time stock market data and not just sample data
- The data has to be preprocessed correctly in order to avoid errors during training and testing phase
- The accuracy of the model not only depends upon it's parameters but also the dataset as well
- The only point most of the papers didn't mention is the deployment, as it's one of the most important fact while making an ML/DL model

Considering the above key points, we focused on making the ML model. We decided to go on with CNN-LSTM Neural Network (Convolutional Neural Network and Long Term Short Memory Neural Network) approach because CNN helps in tracking the features of dataset and LSTM helps in tracking the patterns, allowing to train on them. This approach isn't the first time as some researchers already tried to implement the CNN-LSTM method but we tweaked the parameters, kernel sizes (for CNN) and layers to experiment and test it on real-time data. Since this is a regression type of problem where we had to train with time-series data, we used Mean Square Error as the standard metric rather than accuracy.

For the neural network, we first analysed with other works and then decided the architecture in order to maintain novelty. The architecture diagram for the neural network is shown in Fig. 1.

NOTE: For this project, we mainly used Python and Jupyter Notebooks via Colab and Kaggle to perform the experiment. To see the experiment we did, refer https://github.com/Circle-1/Stock-X

IV. Experimental Analysis

For the experiment, we used Python for data preprocessing and creating a Neural Network model. For deployment, it's considered as minimal because, we saved the model in HDF5 format using Keras, but in addition to that, we made a Docker Image and Helm charts for people to work on. In this section, the major parts of the project and the experiments performed are described.

A. Data Collection, Analysis and Preprocessing

Before making a model, the first step is to collect enough datasets such that the base analysis is made to study about the stock market data. So, we gathered enough datasets from Kaggle (explained in later stages) but realised that they are sample ones and we had to search for real time ones. Then, we came across several finance APIs like Yahoo finance, Alpha Vantage which helps in gathering stock data for specific time period. So, we took Alpha Vantage API and used "TIME_SERIES_DAILY (Extended)" option to obtain stock data of a company ranging since 10 years. We used "full" mode to collect enough data rather than using "compact" mode in API (which fetches only 100 columns meant for rapid usage cases) and we were able to collect the data for any company with valid API keys. Some stock data is also gathered using Google Sheets via "GOOGLEFINANCE" function. Then we stored the data in CSV format for testing phase.

Then, we did an Exploratory Data Analysis (EDA) on the dataset to know about the stock market data in depth. We also implemented Moving Average and Daily Return columns to know how a stock market works and analysed some of the features present in it. After that, we went to preprocessing phase.

In preprocessing phase, we first cleansed the data by removing NULL values from the dataset and taking the mean of data and replacing it if necessary using Pandas library. Then we took the four columns of any stock market dataset, namely "Open", "Close", "High", "Low". These are the columns which mainly involve in training the dataset especially "Close" column (shown in Fig. 2). The graphs are plotted using the matplotlib and seaborn library in Python.

During preprocessing stage, we realised that CNN always considers 2-Dimensional and 3-Dimensional arrays to train and select required features. But here, the data we have is of 1-Dimensional arrays. This is one of the reasons why CNN

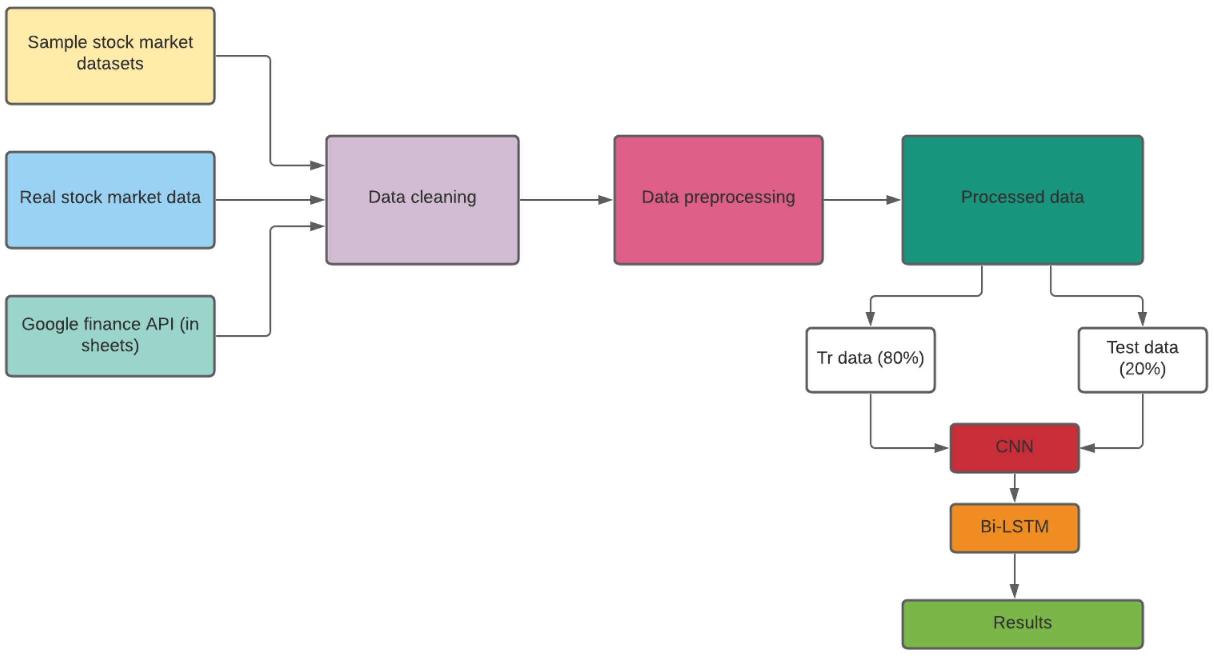
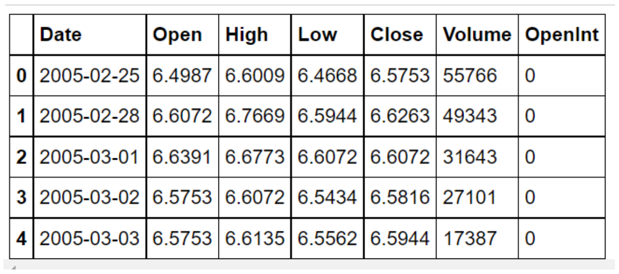


Fig. 1. Architecture for Deep Learning model



	Date	Open	High	Low	Close	Volume	OpenInt
0	2005-02-25	6.4987	6.6009	6.4668	6.5753	55766	0
1	2005-02-28	6.6072	6.7669	6.5944	6.6263	49343	0
2	2005-03-01	6.6391	6.6773	6.6072	6.6072	31643	0
3	2005-03-02	6.5753	6.6072	6.5434	6.5816	27101	0
4	2005-03-03	6.5753	6.6135	6.5562	6.5944	17387	0

Fig. 2. Example of stock market dataset

is often seen in Computer Vision (CV) based applications and not in NLP based applications.

So, in order for CNN model to parse the dataset, we made a function where the 1-D arrays are made to convert to [100,1] tensors (precisely, a vector). Tensors are a type of data structures that describe a mulilinear relationship between set of objects in a vector space. So, for converting 1-D array to tensor, every 100 rows are taken and from that the mean of the values are calculated and made to store in a separate column. This process is done for entire dataset. In our case, we did this on the ”Close” column as it’s the main column where we would decide the prediction of the stock data.

After this step, we would obtain tensors for CNN side of model to train. Then, we split 80% for training and 20% for testing. Finally, we reshaped the data and sent it to the training phase.

B. Training Phase

After the dataset is processed, the NN model has to be made. In our case, it’s the CNN-LSTM Neural Network model.

For our model, we considered to divide the model into two parts, CNN and LSTM.

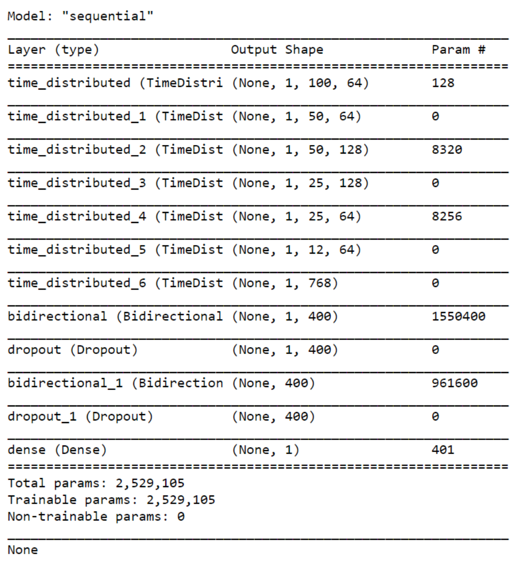
1) CNN: For the CNN section of model, we followed a custom way instead of ascending kind of way in the size of layers. So, we made 3 layers of neuron size 64,128,64 with kernel size=3 along with MaxPooling layers in between. Finally we added a Flatten layer at end of CNN section to convert the tensors back to 1-D array.

All CNN layers are added with TimeDistributed function in order to train every temporal slice of input, as we’re approaching a Time-Series problem in this case. Then, the processed data is sent to LSTM layers.

2) LSTM: For the LSTM section, we made 2 Bi-LSTM layers to detect the features and train them forward & backward. For each layer, the neuron size is 100. Additionally, dropout layers are added in between with value of 0.25 and 0.5 in drop some features for stability.

Last, we added a dense layer with linear activation function and at the final layer we used ”adam” optimizer (”sgd” also worked in this case but we found ”adam” optimizer to be accurate after analysis), Mean Squared Error (mse) as the loss function and ”mse” and ”mae” as metrics.

The architecture for the model is shown in Fig. 3. After the model is trained, we made it to plot the graph for the loss values (both training and validation) and at first it proved to be less and later it varied accordingly (shown in Fig. 4 and 5.). But the loss and mse varies when the model is saved and made



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
time_distributed (TimeDistri (None, 1, 100, 64)        128       
_________________________________________________________________
time_distributed_1 (TimeDist (None, 1, 50, 64)         0         
_________________________________________________________________
time_distributed_2 (TimeDist (None, 1, 50, 128)        8320      
_________________________________________________________________
time_distributed_3 (TimeDist (None, 1, 25, 128)        0         
_________________________________________________________________
time_distributed_4 (TimeDist (None, 1, 25, 64)         8256      
_________________________________________________________________
time_distributed_5 (TimeDist (None, 1, 12, 64)         0         
_________________________________________________________________
time_distributed_6 (TimeDist (None, 1, 768)            0         
_________________________________________________________________
bidirectional (Bidirectional (None, 1, 400)            1550400   
_________________________________________________________________
dropout (Dropout)            (None, 1, 400)            0         
_________________________________________________________________
bidirectional_1 (Bidirection (None, 400)               961600    
_________________________________________________________________
dropout_1 (Dropout)          (None, 400)               0         
_________________________________________________________________
dense (Dense)                (None, 1)                 401       
=================================================================
Total params: 2,529,105
Trainable params: 2,529,105
Non-trainable params: 0
_________________________________________________________________
None
```

Fig. 3. The summary of the proposed model

to train again with saved parameters (this is fully explained in the testing phase). Then, we managed to refine the test dataset back to arrays using the reshape() function, then made the model to predict the dataset. The graph is plotted and the results proved to be great. The model was able to predict most of the stock data with given time series and it's shown in Fig. 6.

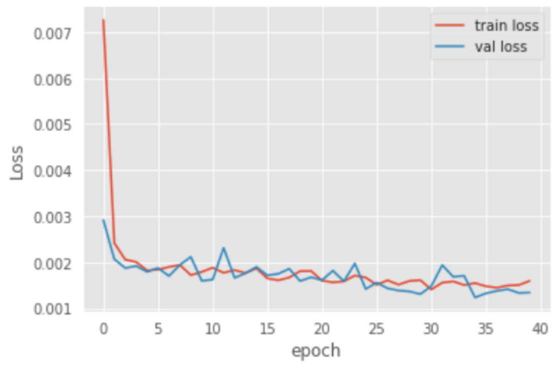


Fig. 4. The loss graph obtained during training

TABLE I
ACCURACY SCORES

Scores	
Name	***Score***
MSE	0.035
MAE	0.075
Variance	0.935370
R2 Score	0.9353
Max Error	0.174930

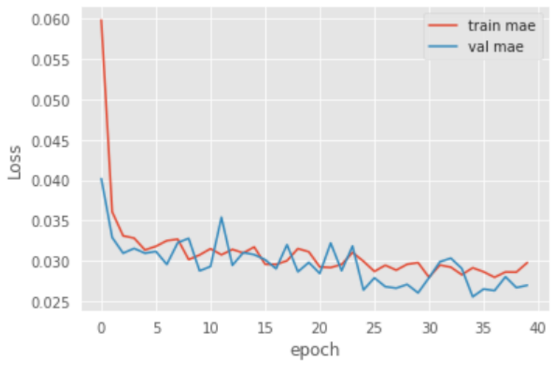


Fig. 5. MAE obtained during training

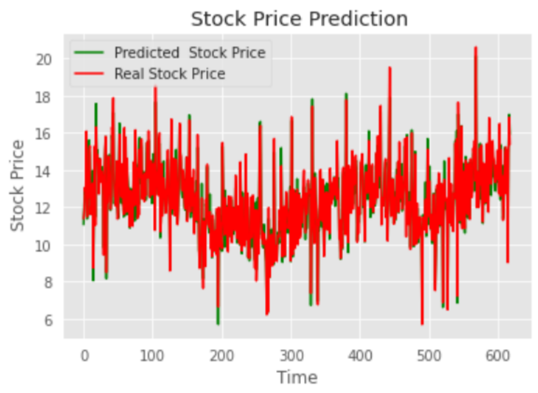


Fig. 6. The prediction graph for sample NSE/NASDAQ data

C. Testing Phase

After the model has been trained and the values are noted, we saved the model in HDF5 format using Keras API in TensorFlow library. Then, we loaded the HDF5 file and tried training the model again but this time with the different dataset, we were able to train the model but the loss value varies accordingly. For example, if the loss is 0.055 during training phase, the loss increases to 0.153 (estimated, not accurate). It is also found that this happens depending on the dataset we use, for NIFTY sample dataset the error didn't occur whereas in the NASDAQ dataset it occured while loading up the saved model.

We tested and experimented the model with different datasets from Kaggle consisting sample data of mixed content from different stock markets [10], NIFTY-50 [11], NASDAQ and NYSE [12] to find how the model copes up with different stock market (shown in Fig. 6 and Fig. 7). We also did with real time dataset by fetching data from Alpha Vantage API (at first step) and tested the model. It was able to predict most of the stocks as shown in the figures and in table II.

D. Deployment

This section is considered as minimal one because we used Kaggle notebooks in order to create and deploy the models but we realised some developers who don't work with Kaggle and

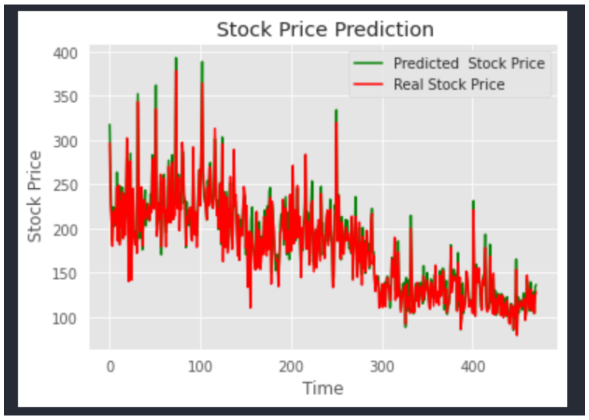


Fig. 7. The prediction graph for sample NIFTY data

TABLE II
ACCURACY WITH DATASETS

Dataset and MSE Scores	
Dataset	***MSE Score***
NIFTY (SBIN - sample)	0.001
NASDAQ (ACTG - sample)	0.1565
NASDAQ (AAOI - sample)	0.0016
NYSE (IBM - real)	0.0027
BSE (RELIANCE - real)	0.0145

prefer local environments and tools like Conda. So, we made a Docker Image for them in order to work with Jupyterlab (instead of Jupyter notebook) with the GitHub repository as the folder. Initially, it was hard because by default, Docker Images are huge in size because of the OS images like Ubuntu. We got the image size as 250MB at first (without installing any Python libraries) and 750MB when uncompressed. We also tried to use pre-defined images for Jupyter notebooks where the libraries are already installed and ready, but we didn't use them as they are of huge size (approximately 3GB). So we added some libraries and made a docker image with Conda environment (via miniconda) on top of it, still we reached 1GB of data due to the size of libraries and scripts bundled so it's considered to be used only when a user doesn't want to use Kaggle as a working environment.

For deploying the image to Docker Hub, we used GitHub actions which is a CI tool from GitHub to write a CI action in which an image is published every time a change or release is made, so that users need not build image everytime locally. Docker Inc. already made a GitHub action [13] to perform the above functions, so we used and modified it to build and push Docker images. Similarly, we also made Helm charts if a person needs to run the Docker Image in a Kubernetes cluster. Initially we faced difficulties but after trial and error, we were able to expose the service to the network and run our Docker Image in Kubernetes. The Helm charts are deployed on the Artifact Hub (https://artifacthub.io) and anyone can download the templates from the repository.

V. End Product

After several works and tests, we were able to obtain a great working CNN-LSTM Neural Network model which can able to predict most of the stocks when the relevant stock market data is provided. In order to check how our model performed, we compared with other models which are obtained during the literature survey and found the results in the table III

The model experimented from [3] is proposed one and haven't been tested rigorously. The model from [9] performed well and stated that an average of 0.0003 to 0.0004 due to the fact that the data is processed and decentralized. In our case, we obtained an MSE of 0.001 maximum and 0.002 in average but varied accordingly with different datasets we experimented.

TABLE III
COMPARISON WITH OTHER MODELS

Models		
Model name	***Reference***	***Accuracy***
CNN-LSTM (Our model)		MSE = 0.035 (avg)
Random Forest	[1]	EVS = -0.400594
AELSTM	[2]	53% (Avg)
LBL-LSTM (Proposed)	[3]	0.017 (Train), 0.026 (Test)
LSTM (Decentralized)	[9]	MSE = 0.0003
IKN-ConvLSTM	[14]	98.307%
k-NN Regression	[15]	90%
Rider-MBO-based Deep-ConvLSTM	[16]	MSE = 7.2487, RMSE = 2.6923
SVM & Logistic Regression	[17]	87% to 90%
SVM with RBF Kernel	[18]	89%
LSTM	[19]	MSE = 3.5 to 3.7

VI. CONCLUSION

Designing a customised CNN-LSTM is bit tricky at first, because there are many algorithms already present to predict the stock data, however not fully optimized. We trained and tested the model with different kinds of datasets like NYSE, NASDAQ and NIFTY and found that accuracy of the model varies accordingly. For NIFTY [11], the model proved to be good and predicted at most 99% of stocks even during the testing stage. But for other datasets like NYSE [12], the accuracy varied during the testing phase and not during training. We believe it may be due to the noise present in the dataset during the testing phase and compiling it again. Also, we obtained MSE of value 0.001 to 0.05 (approx) during training and 0.002 to 0.1 (approx) for validation with different datasets. Overall, the paper presents on how CNN can be combined with LSTM to obtain the features from the tensors processed from dataset and detect patterns based on the features obtained. The paper also presents one of the ways in which stock market can be predicted with great accuracy.

REFERENCES

[1] Kompella, S., & Chakravarthy Chilukuri, K. C. C. (2020). Stock Market Prediction Using Machine Learning Methods. International Journal of Computer Engineering and Technology, 10(3), 2019.

[2] Pang, X., Zhou, Y., Wang, P., Lin, W., & Chang, V. (2020). An innovative neural network approach for stock market prediction. The Journal of Supercomputing, 76(3), 2098-2118.

[3] Gurav, U., & Kotrappa, D. S. (2020). Impact of COVID-19 on stock market performance using efficient and predictive LBL-LSTM based mathematical model. International Journal on Emerging Technologies11 (4), 108-115.

[4] Kimoto, T., Asakawa, K., Yoda, M., & Takeoka, M. (1990, June). Stock market prediction system with modular neural networks. In 1990 IJCNN international joint conference on neural networks (pp. 1-6). IEEE.

[5] Guresen, E., Kayakutlu, G., & Daim, T. U. (2011). Using artificial neural network models in stock market index prediction. Expert Systems with Applications, 38(8), 10389-10397.

[6] Nelson, D. M., Pereira, A. C., & de Oliveira, R. A. (2017, May). Stock market's price movement prediction with LSTM neural networks. In 2017 International joint conference on neural networks (IJCNN) (pp. 1419-1426). IEEE.

[7] Selvin, S., Vinayakumar, R., Gopalakrishnan, E. A., Menon, V. K., & Soman, K. P. (2017, September). Stock price prediction using LSTM, RNN and CNN-sliding window model. In 2017 international conference on advances in computing, communications and informatics (icacci) (pp. 1643-1647). IEEE.

[8] Hiransha, M., Gopalakrishnan, E. A., Menon, V. K., & Soman, K. P. (2018). NSE stock market prediction using deep-learning models. Procedia computer science, 132, 1351-1362.

[9] Bansal, G., Hasija, V., Chamola, V., Kumar, N., & Guizani, M. (2019, December). Smart stock exchange market: a secure predictive decentralized model. In 2019 IEEE Global Communications Conference (GLOBECOM) (pp. 1-6). IEEE.

[10] Kaggle, Huge Stock Market Dataset (online). Link: https://www.kaggle.com/borismarjanovic/price-volume-data-for-all-us-stocks-etfs

[11] Kaggle, NIFTY-50 Stock Market Data (2000-2021) (online). Link: https://www.kaggle.com/rohanrao/nifty50-stock-market-data

[12] Kaggle, Stock Market Data (NASDAQ, NYSE, S&P500). Link: https://www.kaggle.com/paultimothymooney/stock-market-data

[13] GitHub, Build and Push Docker images (GitHub Action), Link: https://github.com/marketplace/actions/build-and-push-docker-images

[14] Nti, I. K., Adekoya, A. F., & Weyori, B. A. (2021). A novel multi-source information-fusion predictive framework based on deep neural networks for accuracy enhancement in stock market prediction. Journal of Big Data, 8(1), 1-28.

[15] Ananthi, M., & Vijayakumar, K. (2021). Stock market analysis using candlestick regression and market trend prediction (CKRM). Journal of Ambient Intelligence and Humanized Computing, 12(5), 4819-4826.

[16] Kelotra, A., & Pandey, P. (2020). Stock market prediction using optimized deep-convlstm model. Big Data, 8(1), 5-24.

[17] Parray, I. R., Khurana, S. S., Kumar, M., & Altalbe, A. A. (2020). Time series data analysis of stock price movement using machine learning techniques. Soft Computing, 24(21), 16509-16517.

[18] Deepak, R. S., Uday, S. I., & Malathi, D. (2017). Machine learning approach in stock market prediction. International Journal of Pure and Applied Mathematics, 115(8), 71-77.

[19] Bhattacharjee, I., & Bhattacharja, P. (2019, December). Stock Price Prediction: A Comparative Study between Traditional Statistical Approach and Machine Learning Approach. In 2019 4th International Conference on Electrical Information and Communication Technology (EICT) (pp. 1-6). IEEE.