Project Report

Stock market price prediction

NAME	ID
Supratim Chatterjee	2019A8PS0652H
Aayush Keval Shah	2019A7PS0137H
Navam Shrivastav	2019A7PS0092H
Jay Patel	2019A7PS0156H
Aditya Choudhari	2019AAPS0309H
Ashwin Wadatkar	2019A7PS0082H

Problem Statement

Every Stock Exchange has its own value for the Stock Index. The index is the average value derived by adding up the prices of various equities. The stock market can have a significant impact on individuals and the economy as a whole. This aids in the representation of the entire stock market as well as the forecasting of market movement over time. As a result, effectively predicting stock trends can reduce the risk of loss while increasing profit.

Paper Link - **Stock Market Prediction for Time-series Forecasting using Prophet upon ARIMA**

Model Proposed in Research Paper

Research Paper proposes the Prophet Model to forecast the stock price. Prophet is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. It works best

with time series that have strong seasonal effects and several seasons of historical data. Prophet is robust to missing data and shifts in the trend, and typically handles outliers well. This model takes only two columns as input. First column is the date, and the second column is the closing price of stock on that date. The below given results are for the basic prophet model.

Main code for Prophet Model

```
##Predicting Using Prophet
forecast=prophet_model.predict(future)
forecast.index = prophet_data['y'].index
```

```
prophet_df = pd.concat([forecast['yhat'],prophet_data['y']],axis=1,ignore_index=True)
prophet_df.columns = ['Predicted','Actual']
prophet_df
```

Plots representing the training and testing error for Prophet Model

Experiments conducted in search of Innovation

Hyper-Tuning for Prophet Model

After Hyper-parameter tuning the model is not able to capture the seasonality and sudden jump in time series in the Year 2017 onwards.

Following are the conclusions by doing the above experiment :

- Prophet is easily overfitted.
- Prophet is good at capturing the trend.
- By creating extra regressors we can maybe improve the results.

```
from sklearn.model_selection import ParameterGrid
params_grid = {'seasonality_mode':('multiplicative','additive'),
               'changepoint_prior_scale':[0.3,0.4],
               'holidays_prior_scale':[0.3,0.4],
               'n_changepoints' : [20,50]}
grid = ParameterGrid(params_grid)
cnt = 0
for p in grid:
    cnt = cnt+1

print('Total Possible Models',cnt)
```

```
strt='2017-08-31'
end='2019-12-31'
model_parameters = pd.DataFrame(columns = ['MAPE','Parameters'])
for i in grid:
    test = pd.DataFrame()
    print(i)

    train_model =Prophet(changepoint_prior_scale = i['changepoint_prior_scale'],
                         holidays_prior_scale = i['holidays_prior_scale'],
                         n_changepoints = i['n_changepoints'],
                         seasonality_mode = i['seasonality_mode'],
                         weekly_seasonality=False,
                         daily_seasonality = False,
                         yearly_seasonality = True,
                         )

    train_model.fit(prophet_train_hyper)
    train_forecast = train_model.make_future_dataframe(periods=29, freq='M',include_history = False)
    train_forecast = train_model.predict(train_forecast)
    test=train_forecast[['ds','yhat']]
    Actual = df[(df['ds']>=strt) & (df['ds']<=end)]
    MAPE = mean_absolute_percentage_error(Actual['y'],abs(test['yhat']))
    print('Mean Absolute Percentage Error(MAPE)------------------------------------------------------------',MAPE)
    model_parameters = model_parameters.append({'MAPE':MAPE,'Parameters':p},ignore_index=True)
```

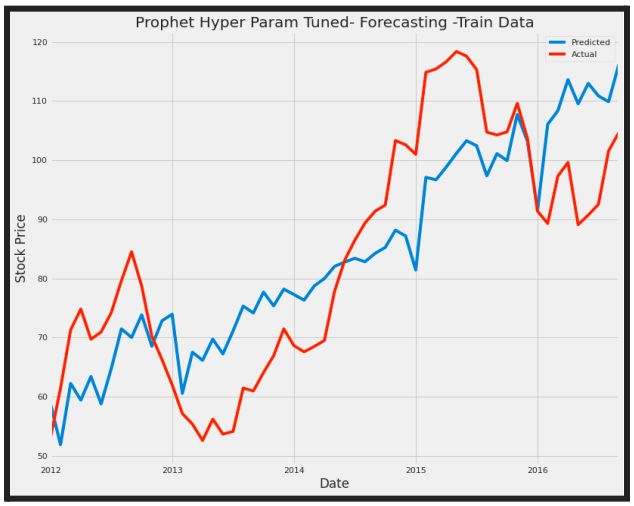
Results after tuning the hyperparameters

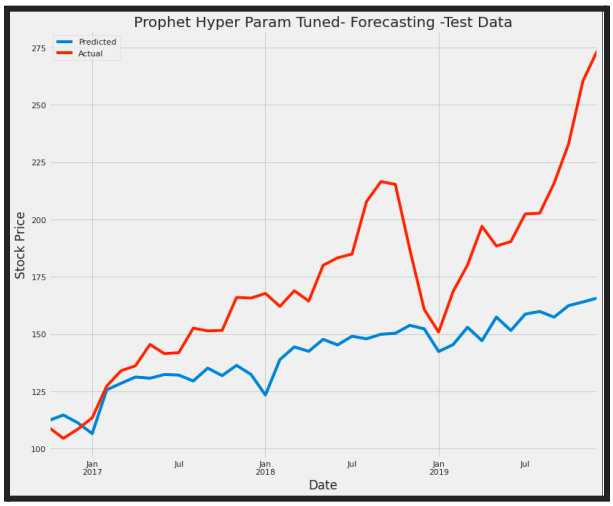
```
result_metrics(prophet_hyper_df['Actual'][:67],prophet_hyper_df['Predicted'][:67],'Hyper-Tuned Prophet Train Data')
```

```
Result Metrics for Hyper-Tuned Prophet Train Data
R2 Score :  0.784
Mean Squared Error :  129.543
Mean Absolute Error :  9.986
Mean Absolute Percentage Error 12.076
Accuracy(100-MAPE) of Model is 88.0%
```

```
result_metrics(prophet_hyper_df['Actual'][67:],prophet_hyper_df['Predicted'][67:],'Hyper-Tuned Prophet Test Data')
```

```
Result Metrics for Hyper-Tuned Prophet Test Data
R2 Score :  -1.31
Mean Squared Error :  2187.789
Mean Absolute Error :  40.501
Mean Absolute Percentage Error 20.338
Accuracy(100-MAPE) of Model is 80.0%
```





Innovation proposed

SARIMA Model

SARIMA is seasonal auto regressive integrated moving average and is an extension of ARIMA which explicitly supports univariate time series data with a seasonal component.
There are three trend elements that require configuration.
They are the same as the ARIMA model specifically :

- **p**: Trend autoregression order.
- **d**: Trend difference order.
- **q**: Trend moving average order.

There are four seasonal elements that are not part of ARIMA that must be configured
They are as follows :

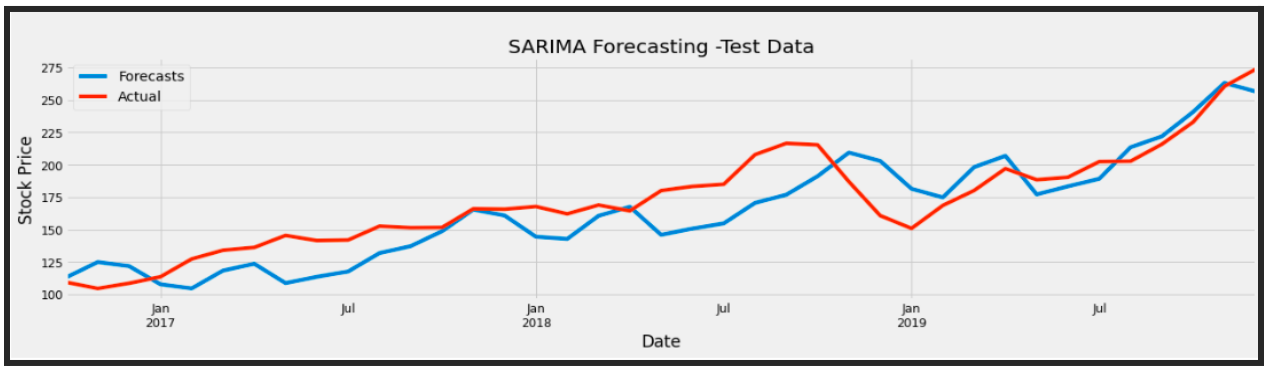
- **P**: Seasonal autoregressive order.
- **D**: Seasonal difference order.
- **Q**: Seasonal moving average order.
- **m**: The number of time steps for a single seasonal period.

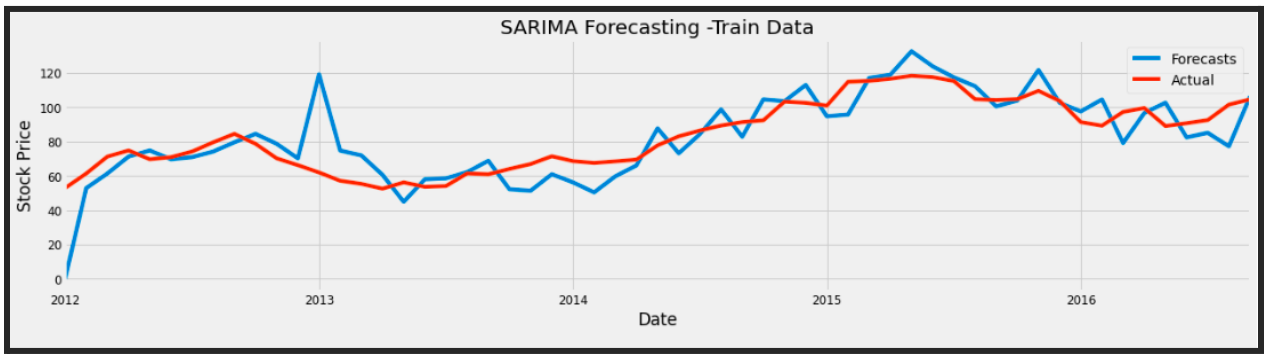
Main code for SARIMA Model

```
list_param = []
list_param_seasonal=[]
list_results_aic=[]

for param in pdq:
    for param_seasonal in seasonal_pdq:
        try:
            model = sm.tsa.statespace.SARIMAX(train,
                                              order=param,
                                              seasonal_order=param_seasonal,
                                              enforce_stationarity=False,
                                              enforce_invertibility=False)

            results = model.fit()

            print('ARIMA{}x{}12 - AIC:{}'.format(param, param_seasonal, results.aic))

            list_param.append(param)
            list_param_seasonal.append(param_seasonal)
            list_results_aic.append(results.aic)
        except:
            continue
```

```
model = sm.tsa.SARIMAX(train,order=(1,1,1),seasonal_order=(2,2,0,12))
results=model.fit()
forecasts_train = results.predict(start='2012-01-31',end='2016-09-30')
forecasts_test = results.predict(start='2016-10-31',end='2019-12-31')
```

Results for SARIMA Model

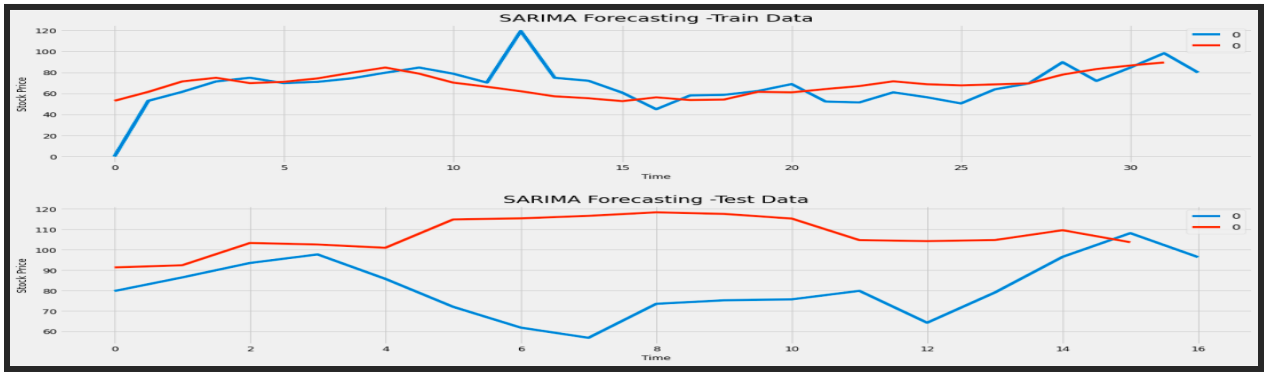


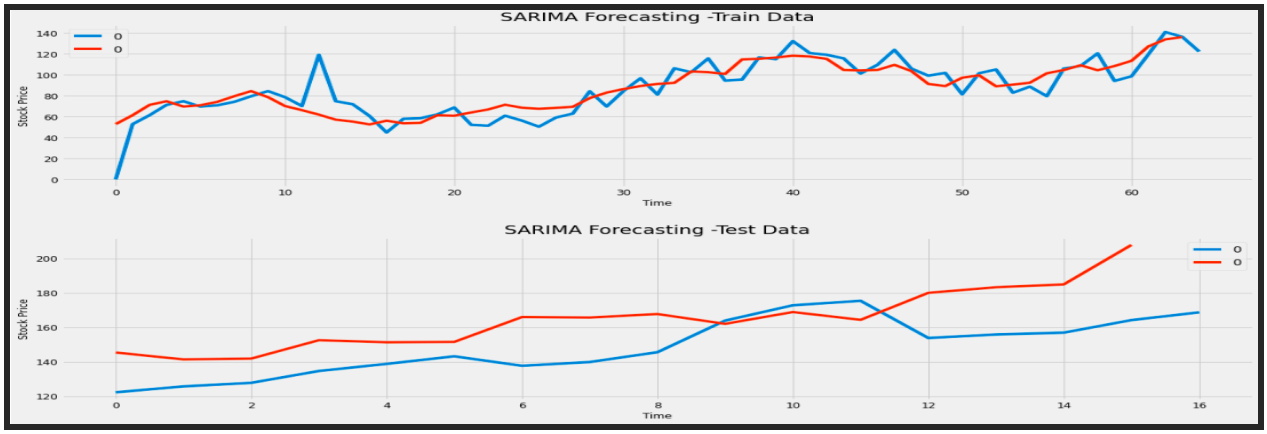


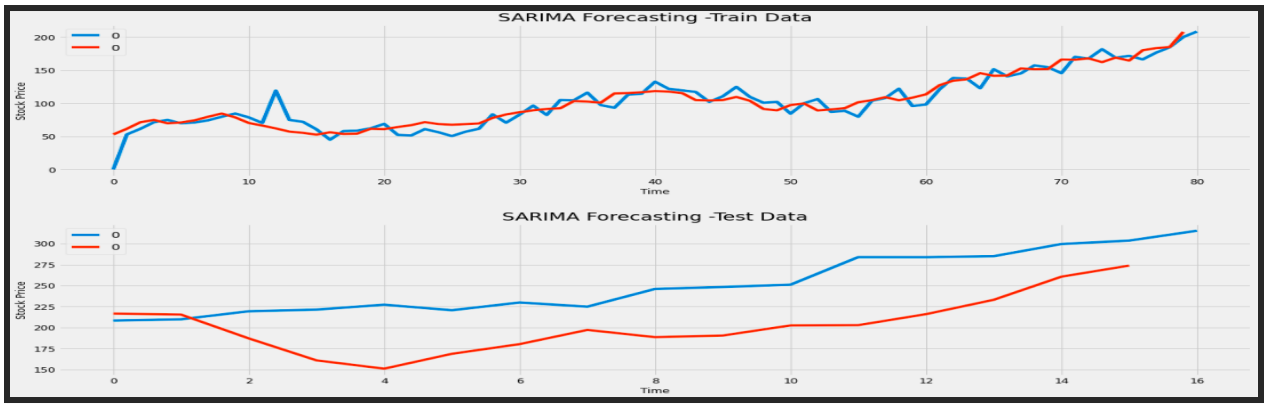

```
Result Metrics for SARIMA-Train Data
R2 Score :  0.491
Mean Squared Error :  199.567
Mean Absolute Error :  9.797
Mean Absolute Percentage Error 13.39
Accuracy(100-MAPE) of Model is 87.0%
None
----
Result Metrics for SARIMA-Test Data
R2 Score :  0.704
Mean Squared Error :  441.121
Mean Absolute Error :  17.557
Mean Absolute Percentage Error 10.65
Accuracy(100-MAPE) of Model is 89.0%
```

Training and Testing data using different types of split in databases like TimeSeries Split, Stratified Shuffle Split, Shuffle Split.

Plots representing the training and testing error for SARIMA Model







Results

SARIMA Vs Facebook's Prophet

- Advantages of Prophet include a very easy to implement, fast, and less statistical know-how model . In Seasonal ARIMA we had to follow a lot of tests and processes to generate predictions.
- Seasonal ARIMA is better at capturing the seasonality part .
- Prophet has an overfitting problem .
- Overall Both models are robust .
- Prophet is better at dealing with outliers.
- We have found Seasonal ARIMA is much better at our prediction problem. More confidence when predicting with Seasonal-ARIMA since it is backed by Mathematical and Statistical tests.
- Accuracy of SARIMA is 89% and 80% for Prophet(Both on Out of Sample Data) .

Main findings and Accomplishments

- We found out that for non linear data like stock prices, models like linear regression, support vector machine etc are less accurate than models like Prophet, SARIMA, Random Forest etc.
- Overcoming the overfitting problem increases the accuracy to a great extent.
- By changing the model from Prophet to SARIMA in search of innovation, we were able to improve the accuracy of prediction.
- We were able to predict the stock prices with accuracy as high as 90% using the SARIMA model.
- Hyper tuning the model also resulted in improvement of accuracy.
- We got an average accuracy of 86% on testing on a dataset of multiple stocks.

Github Link - https://github.com/pjay20301/ML-Project

Datasets were taken from - https://finance.yahoo.com/

Name	Task Done
Jay Patel - 2019A7PS0156H	Implemented the research paper using Prophet Model
Supratim Chatterjee - 2019A8PS0652H	Implemented the innovated model using SARIMA
Navam Shrivastav - 2019A7PS0092H	Implemented the research paper using Prophet Model
Aditya Choudhari - 2019AAPS0309H	Implemented the innovated model using SARIMA
Aayush Shah - 2019A7PS0137H	Implemented the innovated model using SARIMA
Ashwin Wadatkar - 2019A7PS0082H	Implemented the research paper using Prophet Model

Literature Review

Name : Supratim Chatterjee
Paper : **Prediction in Stock Price Using of Python and Machine Learning | IEEE Conference Publication | IEEE Xplore**

The paper discusses an ML procedure known as LSTM i.e. Long short-term memory to forecast stock prices and find instabilities in stock values. It makes predictions based on the last 60 days closing price of the stock. Earlier work included utilizing Support vector machines(SVMs), ANN, and Rf methods for stock price prediction. Most of these models were hybrid with ANN being combined with other architectures for predictions. The authors collected the stock values of the organization ‘Alphabet Inc.’ from 1st January 2012 to May 10th, 2021 from X Finance-Yahoo. The dataset comprised of 2261 rows and 6 columns. The autonomous train information collection and the subordinate informational index were converted to NumPy clusters. The information is then

restructured into three dimensions, the number of time steps, the number of features, and the number of tests. The model had two LSTM layers with fifty neurons and two dense layers, one with twenty-five neurons and the other with one neuron. Adam optimizer is used for the model. It achieved the highest level of performance with respect to the root Least Squared error.
The model achieved a root mean square error of 57.966. The predicted price of the stock was 2281.20 USD.

Name : Ashwin Avinash Wadatkar
Paper : **Hybrid Quantum Network for classification of finance and MNIST data | IEEE Conference Publication | IEEE Xplore**

This paper discusses an approach, where a quantumclassical network is applied to classify non-trivial datasets (finance and MNIST data). These (hybrid) quantum networks are being considered to tackle the issue of the classical machine learning overfitting of data set problems. Quantum machine learning (QML) emerged as an approach to combine classical machine learning ideas (e.g. deep neural nets) with the peculiarities of quantum computations - superposition and entanglement.

Quantum classifiers are algorithms suitable for binary or multiclass classification tasks. These algorithms work suitably for balanced data sets/targets. A common approach in QML for NISQ is named “variational (or parameterized) quantum classifier” (VQC or PQC), the VQC ansatz.

Shortcomings : The number of qubits needed to encode the classical data are proportional to the number of features, which are infeasible with the scope of the quantum computers of the NISQ era. A solution to this problem was to encode the classical data using a classical neural network, which is used to effectively scale down the number of features in the training data to the parameters used in the quantum network.

In the last classical layer, a softmax activation function is used to feed the results of the network into the loss function. The quantum network is built with all qubits in |0> state, after which the transformation R_y is applied (rotation gates). After this encoding part, variational part Uˆ is applied. There are further two approaches : one in which the variational part is applied n_{layers} times, and another where data re-uploading is carried out as well. In the hybrid learning approach, only the last layer is substituted with a quantum network. Generally it is expected that entanglement of qubits will improve the quality of QML.

Name : Aditya Choudhari
Paper : **Predicting Chinese Stock Market Price Trend Using Machine Learning Approach**

The stock market is dynamic, noisy and hard to predict. In this paper, we explored four machine learning models using technical indicators as input features to predict the price trend 30 days later. The experimental dataset is Shanghai Stock Exchange(SSE) 50 index stocks. The raw data includes open price, close price, high price, low price and trading volume for the day. We split the first 80% of data as a training set, the remaining 20% data as a test set.

Four machine learning models were tested for accuracy. These were SVM, Artificial Neural Network, Random Forest, and Naive Bayesian Classifier.

The SVM model had hyper-parameters included and a penalty rate for misclassification. Both of these parameters have significant effects on the performance of the model that need to be optimized. The kernel function used in the SVM model is the radial basis function.
We employ five-fold cross-validation to seek optimal parameters of the SVM model. The whole training dataset is divided into five parts, among which four parts are used for training and the remaining fifth part for testing. The procedure is repeated five times so that each part could be used for testing. The performance of different parameters settings is evaluated based on the prediction accuracy which is the percentage of the correct classified data point over the total number of data points. Once the optimal parameters are chosen, they are used to classify the testing data point

The result demonstrates the superiority of ANN over the other three models. The ANN had a testing accuracy of 53.88 %. However, the SVM achieves the best training accuracy of 69.76 %.

Name : Aayush Keval Shah
Paper : **Stock Market Prediction for Time-series Forecasting using Prophet upon ARIMA**

Traditionally, stock prediction models fail to continuously predict proper trends. For time series data, the models predict based on the market data and historical trends but most don't take user selection of type of investment and the risk which the user is willing to take. In this paper, the proposed model will project the returns based on the user's selection of types of investments and the risk he/she is willing to take.

We will use the PyStan and fbprophet libraries of python to use Facebook's Prophet model to forecast time series data. We can forecast time series data mainly based on additive models using Prophet unlike non-linear trends that in general works with daily, weekly, and yearly as well for seasonality, plus holidays. It gives best results for time series which have several season(s) of historical data and strong seasonal effects. The major important requirement that Prophet had was pystan. PyStan provides an interface for statistican modelling and high-performance statistical computation.

All the required packages like PyStan and fbprophet are installed and imported. The dataset containing historical data of 37 years is imported directly from Zen Securities Pvt Ltd. The data is loaded into a pandas dataframe and is converted to a stationary data. The training set contains stock market prices from 1981–03–31 to 1991–03–31 while the test set contains prices from 1991–04–01 to 2001–03–31. Modelling is done by instantiating the Prophet to the model and fit with the data frame. The future data is forecasted and analysed for results.

The result is given in the form of a graph for 10 traditional sectors for which the seasonality and trends components were determined. People who don't have vast prior knowledge can also use this model to predict future trends as it will automatically find seasonal related trends.

Name : Jay Patel
Paper : **Predicting the direction of stock market prices using random forest**

The paper describes the prediction of stock market prices as a classification problem. The model uses a random forest algorithm to forecast the stock price. The algorithm belongs to a class of machine learning algorithms called ensemble learning. The indicators used for prediction are Relative Strength Index (RSI), stochastic Oscillator, Moving Average Convergence Divergence (MACD), Price Rate of Change (PROC), etc. The methodology has five steps, Data collection, exponential smoothing, feature extraction, ensemble learning, and stock market prediction. If the classifier returns a value of +1 then the stock market price tends to increase, and if it returns -1 then the price is more likely to decrease.

Because of the low bias and high variance of the decision trees, they tend to overfit the training sets when they grow deep. The random forest algorithm resolves this problem by training multiple decision trees on different subspaces of the feature space at the cost of increased bias. Because of this, none of the trees are able to see the complete dataset. Each node is associated with an attribute and splitting is based on that attribute. Impurity features like Gini impurity and Shannon entropy are used for recursively splitting the data.

We observe that as the trading period and the number of trees increases, the out-of-bag (OOB) error decreases, and the accuracy of the model increases. We achieve an accuracy of approximately 86%, 90%, and 92% for the trading period of 1 month, 2 months, and 3 months respectively. This model shows more accuracy than logistic regression, gaussian discriminant analysis, quadratic discriminant analysis, and support vector machine. This paper identifies that this problem is not linearly separable and hence algorithms like SVM should not work.

Name : Navam Shrivastav
Paper : **Prediction of Stock Market Using Recurrent Neural Network**

The paper focuses on prediction of stock market prices using Recurrent Neural Networks (RNN). In order to avoid the problem of exploding and vanishing gradients in RNN, Long Short Term Memory i.e. LSTM was used as the core of the architecture in this paper. The data set used was the stock prices of several companies from 1999 to 2021 and this was taken from Stock Bangladesh website. There are 6 features in the data set which are "Date", "Open", "Highest Price", "Lowest Price", "Close Price" and "Volume" and has around 4000 rows of data in it.

The data set was normalized as a part of the pre-processing step and was split as 80% training set and 20% testing set. Among the training set, there was again a split with 10% of it as the validation set and walk-forward-cross-validation was used. While training the model the main difficulty was to set the number of epochs for the model which would work for all the companies in the data set. The hyperparameter number of epochs was tuned to 100 which gave good results for all the companies.

After training, an accuracy of 88% was obtained for the whole data set. For the company BEXIMCO, an accuracy of more than 90% was achieved and this was special as getting this much accuracy is very rare in stock market analysis. Though these results were very good, predicting the stock market is a very difficult task. The authors suggested some potential improvements by having an even larger data set as Deep Learning models tend to have a large number of parameters which need an even more number of data points for better training. They also proposed to change the model and try some other models like ResNet or Attention models for better results.