Using Machine Learning in Stock Market

Rutvik 1810110202

Imran 1810110083

Prof. Madan Gopal

Acknowledgement

This report is a culmination of our learnings over the course EED-363-Applications Of Machine Learning. We would like to express our gratitude to Prof Madan Gopal for conducting the course, sparking our interest in the subject, and supporting us throughout the development of this report. We would also like to express our gratitude to the institution of Shiv Nadar University for providing us with a multitude of resources and a platform for us to be able to engage in meaningful research. The course has been a great learning experience and we look forward to developing our skills further in the field.

Abstract:

Stock market forecasting is a challenging research activity and in this paper, we perform technical analysis on historical stock market data to predict future prices using machine learning and deep learning techniques. We aim to perform and compare different machine learning algorithms and techniques and their accuracy in successfully predicting stock prices. We analyse the stock time series data by implementing them in Python code and discussing them theoretically.

Table of Contents:

Introduction:

Stock Market Prediction is the process of trying to predict the future value of a company stock or any other financial instruments traded on an exchange, based on their current information and trends. Such predictions could yield significant profits when successful and over time, many analysis techniques have been used to create stock market forecasts. This forecasting has posed as a challenging research activity that is now expanding with the availability of new data sources, markets, financial instruments, and algorithms. There is a lot of research being conducted and data mining done on stock market data in order to predict stock market prices with higher accuracies.

Stock Market analysis methodologies fall broadly into two categories — Fundamental Analysis and Technical Analysis.

- Fundamental Analysis consists of examining a stock's economic and financial factors in order to measure a security's intrinsic value. It analyses the company's future profitability on the basis of macroeconomic factors such as its current business environment and financial performance, the state of the economy etc.
- Technical analysis consists of using charts and statistical figures to predict trends and future market behaviour.

Our focus will be on technical analysis. Technical analysis of stocks and trends have been performed for hundreds of years since the 17th century. In the modern day world, however, technical analysis owes heavily to Charles Dow, William P. Hamilton, Robert Rhea, Edson Gould and many others who proposed revolutionary theories to perform technical analysis on stock trends. They reflected a new point of view that the trends in a market can be measured in chart patterns of highs and lows and statistical indicators. The fundamental principle that technical analysis is based on is that all available information that may have an effect on the market is reflected in the market price. As a consequence, as they are already priced into a given security, there is no need to look at economic, fundamental or other developments.

This report is an empirical study on various open source models available on Stock Market prediction. We have chosen the models in accordance with the sequential nature of the data.

Raw data was acquired from yahoo (yahoo api). Mainly ICICIBANK (2018-2019) is used for stock market prediction. Our beginning studies showed poor results with 1 year worth of data so we doubled it to 2 years in this report.

Non linear models (especially Neural network) ones have been run 5 times and then their accuracy metrics are reported.

RMSE metric will be reported for each model and our goal of this study is to minimise this metric.

Above data is for ICICI BANK

Data Description And Pre-Processing:

The data being used for analysis has been taken from Yahoo and represents the stock prices for ICICI Bank, Gail and ONGC from 1th January 2018 to 31st December 2019. The data represents a series of data points indexed by time or a time series. The 2 years data is divided into train and validation sets for each of the implementations. The data is complete and as such no features are missing or inaccurate. The data represents typical stock market descriptors -

OPEN: Represents the price at which trading starts for a particular day
CLOSE: Represents the price at the end of a trading day
HIGH: The maximum price of a stock on a particular trading day
LOW: The minimum price of a stock on a trading day
LAST: The last traded price of a stock for a day
TOTAL TRADE QUANTITY: Represents the trade volume and is the total number of stocks of a company bought or sold on a day
TURNOVER: The turnover of the company on the particular day represented in lacs.

It should be kept in mind that stock trading floors are closed on weekends and national holidays and the same is reflected in the dataset as well since some dates are missing from the same.

For a stock the profit/loss made is specified by the closing price of the same. Closing price hence is the factor that will be used for prediction purposes and will as such act as the target variable. Total trade quantity is also a very important metric for a decrease in the same along with an increase in price shows a declining interest in a stock while a sizable price change on a large volume is also an indicator of a major change in the company's outlook

To keep the models scalable and reusable over other stocks, only the "*symbol*" variable needs to be changed according to yahoo symbols. Even start and end dates can be changed to increase dataset size.

We have dropped the columns Open, High, Low and Adj Close as they require a dataset with higher precision to be of use. Implying dataset needs to have at least a 15min time frame for each day over a year. As such data is paid we have not included that in our study.

Data being of Sequential nature, each day is fed like a feature and corresponding Closing Price is used for either training or calculating accuracy metrics accordingly. Data is broken into 70% train and 30% test.
For a detailed implementation please refer to the hypothesis models code submitted along with this report

Previous report submission on this topic had 1-year (2019) ICICIBANK data under trial. We have decided to use 2-years (2018-2019) for better understanding in this final report.

Data Extraction using Yahoo API:

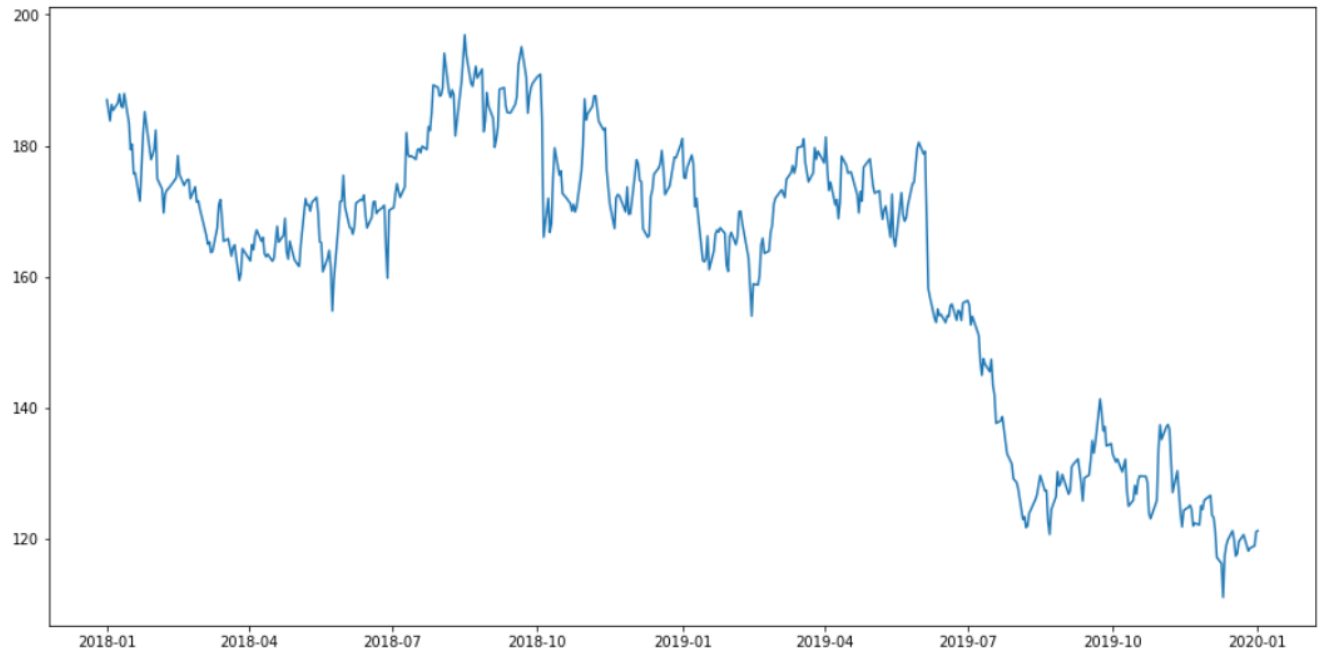
```
# Define which online source to use
data_source = 'yahoo'

# define start and end dates
start_date = '2018-01-01'
end_date = '2019-12-31'
#symbol = 'ICICIBANK.NS'
symbol = 'GAIL.NS'
#symbol = 'ONGC.NS'
# Use pandas_datareader.data.DataReader to load the desired data list(companies_dict.values()) used for python 3 compatibility
web.DataReader(symbol, data_source, start_date, end_date).to_csv(symbol+'.csv')
df = pd.read_csv(symbol+'.csv')
df = pd.DataFrame(df,columns=['Date','Open','High','Low','Close']).round(decimals=2)
df.head()
```

	Date	Open	High	Low	Close
0	2018-01-01	188.55	189.28	186.32	186.99
1	2018-01-02	189.00	189.00	184.99	185.38
2	2018-01-03	186.30	186.54	183.19	183.79
3	2018-01-04	185.25	186.90	183.64	186.26
4	2018-01-05	187.13	187.13	184.54	185.40

Data Visualisation:

```
#creating a separate dataset
disdata = pd.DataFrame(df,columns=['Date','Open','High','Low','Close'])

#setting index as date
disdata['Date'] = pd.to_datetime(disdata.Date,format='%Y-%m-%d')
disdata.index = disdata['Date']

#plot
plt.figure(figsize=(16,8))
plt.plot(disdata['Close'], label='Close Price history')
```

```
[<matplotlib.lines.Line2D at 0x7f1eb2aae850>]
```



Above data is for GAIL

Brief Guide to run the models:

Each hypothesis model has two files available: .ipynb and .py

It is advised to run the .ipynb file on Google Colab for all the models to avoid installing many libraries.

If .py files used run:

pip install "insert necessary library"

before running the .py files themselves.

Please use .ipynb files as the models were developed on Google Colab and do not require any prior installation.

Note: Open sources models are imported via sklearn library

Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.

Hypothesis Models:

Here the theory of the models implemented is discussed and later we will discuss each model's implication.

Linear Regression:

Linear Regression is a machine learning algorithm based on **supervised learning**. It performs a **regression task**. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables, they are considering and the number of independent variables being used.Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input) and y(output). Hence, the name is Linear Regression.

With simple linear regression we want to model our data as follows:

$$y = B0 + B1 * x$$

This is a line where y is the output variable we want to predict, x is the input variable we know and B0 and B1 are coefficients that we need to estimate that move the line around.

Technically, B0 is called the intercept because it determines where the line intercepts the y-axis. In machine learning we can call this the bias, because it is added to offset all predictions that we make. The B1 term is called the slope because it defines the slope of the line or how x translates into a y value before we add our bias.

The goal is to find the best estimates for the coefficients to minimize the errors in predicting y from x.The representation is a linear equation that combines a specific set of input values (x) the solution to which is the predicted output for that set of input values (y). As such, both the input values (x) and the output value are numeric.

It is common to talk about the complexity of a regression model like linear regression. This refers to the number of coefficients used in the model.

When a coefficient becomes zero, it effectively removes the influence of the input variable on the model and therefore from the prediction made from the model (0 * x = 0). This becomes relevant if you look at regularization methods that change the learning algorithm to reduce the complexity of regression models by putting pressure on the absolute size of the coefficients, driving some to zero

In higher dimensions when we have more than one input (x), the line is called a plane or a hyper-plane. The representation therefore is the form of the equation and the specific values used for the coefficients (e.g. B0 and B1 in the above example)..

Given the representation is a linear equation, making predictions is as simple as solving the equation for a specific set of inputs.

Let's make this concrete with an example. Imagine we are predicting weight (y) from height (x). Our linear regression model representation for this problem would be:

$$y = B0 + B1 * x1$$

or

$$\text{weight} = B0 + B1 * \text{height}$$

Where B0 is the bias coefficient and B1 is the coefficient for the height column. We use a learning technique to find a good set of coefficient values. Once found, we can plug in different height values to predict the weight.

For example, lets use B0 = 0.1 and B1 = 0.5. Let's plug them in and calculate the weight (in kilograms) for a person with the height of 182 centimeters.

$$weight = 0.1 + 0.5 * 182$$

$$weight = 91.1$$

You can see that the above equation could be plotted as a line in two-dimensions. The B0 is our starting point regardless of what height we have. We can run through a bunch of heights from 100 to 250 centimeters and plug them to the equation and get weight values, creating our line. Similarly we predict the values for the stocks.

```
from sklearn.model_selection import train_test_split
train, test = train_test_split(data, test_size=0.30,shuffle=False)
from sklearn.linear_model import LinearRegression

x_train = np.array(train.index).reshape(-1, 1)
y_train = train['Close']

# Create LinearRegression Object
model = LinearRegression()
# Fit linear model using the train data set
model.fit(x_train, y_train)
```

Lasso:

The "LASSO" stands for Least Absolute Shrinkage and Selection Operator. Lasso regression is a regularization technique. It is used over regression methods for a more accurate prediction.This model is like linear regression, but it uses a technique "shrinkage" ,where data values are shrunk towards a central point as the mean. The lasso procedure encourages simple, sparse models (i.e. models with fewer parameters). This particular type of regression is well-suited for models showing high levels of multicollinearity or when you want to automate certain parts of model selection, like variable selection/parameter

elimination. The lasso regression allows you to shrink or regularize these coefficients to avoid overfitting and make them work better on different datasets. Lasso regression penalizes less important features of your dataset and makes their respective coefficients zero, thereby eliminating them. Thus it provides you with the benefit of feature selection and simple model creation.

So, if the dataset has high dimensionality and high correlation, lasso regression can be used.

Observing this numerically,

Residual Sum of Squares + λ * (Sum of the absolute value of the magnitude of coefficients)

$$\sum_{i=1}^{n}(y_i - \sum_{j} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

Where,

- λ denotes the amount of shrinkage.
- λ = 0 implies all features are considered and it is equivalent to the linear regression where only the residual sum of squares is considered to build a predictive model
- λ = ∞ implies no feature is considered i.e, as λ closes to infinity it eliminates more and more features
- The bias increases with increase in λ
- variance increases with decrease in λ

We have implemented it in the following way:

```
from sklearn.model_selection import train_test_split
train, test = train_test_split(data, test_size=0.30,shuffle=False)
from sklearn.linear_model import Lasso

x_train = np.array(train.index).reshape(-1, 1)
y_train = train['Close']

# Create LinearRegression Object
model = Lasso()
# Fit linear model using the train data set
model.fit(x_train, y_train)
```

KNN(K-nearest neighbors)

The k-nearest neighbour algorithm is a simple non-parametric method used for classification and regression. It belongs to the family of instance-based, competitive learning and lazy learning algorithms. When we need a prediction for a particular unseen test data value, the k-NN algorithm searches through the training dataset for the k most similar instances. A majority vote is then carried out among the prediction attributes of these k selected records and returned as the prediction for the unseen instance. Euclidean distance is one of the commonly used similarity metrics to take a decision for the data. In case of classification, the most prevalent class is returned, But in case of k-NN regression, the average of the prediction attributes of the k nearest neighbours are returned.

The kNN is powerful since it does not assume anything about the data except the fact that the distance measure can be consistently calculated between any 2 instances. Hence, we see that it does not assume a functional form and as a result, it is called non-linear.

It is possible to map a stock prediction problem into a similarity based classification. The test data and the previous years' stock price data is mapped into a set of vectors where

each vector represents N dimension for each stock feature. The closing price acts as the target variable.

In the Python code implementation for predicting stock prices using the k-NN algorithm, we make use of the sklearn.neighbors.KNeighborsRegressor() function that from the sklearn library that performs regression based on the k-nearest neighbours.The target is predicted by local interpolation of the targets associated of the nearest neighbors in the training set.

We have implemented it in the following way:

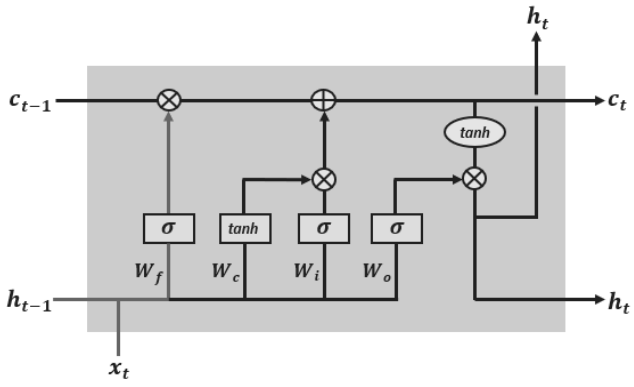
```
from sklearn.model_selection import train_test_split
train, test = train_test_split(data, test_size=0.30,shuffle=False)

x_train = np.array(train.index).reshape(-1, 1)
y_train = train['Close']

x_test = np.array(test.index).reshape(-1, 1)
y_test = test['Close']
```

```
#scaling data
x_train_scaled = scaler.fit_transform(x_train)
x_train = pd.DataFrame(x_train_scaled)
x_test_scaled = scaler.fit_transform(x_test)
x_test = pd.DataFrame(x_test_scaled)

#using gridsearch to find the best parameter
params = {'n_neighbors':[2,3,4,5,6,7,8,9]}
knn = neighbors.KNeighborsRegressor()
model = GridSearchCV(knn, params, cv=5)

#fit the model and make predictions
model.fit(x_train,y_train)
preds_train = model.predict(x_train)
preds_test = model.predict(x_test)
```

LSTM:

Long Short Term Memory is a kind of recurrent neural network. In RNN output from the last step is fed as input in the current step. LSTM was designed by Hochreiter & Schmidhuber. It tackled the problem of long-term dependencies of RNN in which the RNN cannot predict the word stored in the long term memory but can give more accurate predictions from the recent information. As the gap length increases RNN does not give efficient performance. LSTM can by default retain the information for a long period of time. It is used for processing, predicting and classifying on the basis of time series data.

An LSTM cell is shown below:



The LSTM has 4 components

- Memory cell
- Forget gate
- Input gate
- Output gate

Memory cell

The purpose of this cell is to remember/forget data from the previous cell. Remembering/forgetting is done based on the context of the input. Below is the memory cell

There are 2 operations that can be performed in this cell, pointwise multiplication and addition. For example, let's assume that the previous cell's output is a vector [1 2 3 4 5] and we get a vector [1 1 1 0 0] from the forget gate. Upon pointwise multiplication we get the vector [1 2 3 0 0] . Here the last 2 elements of the vector are 0 which implies that in this cell, the data corresponding to the 4th and 5th element of the array has been forgotten.

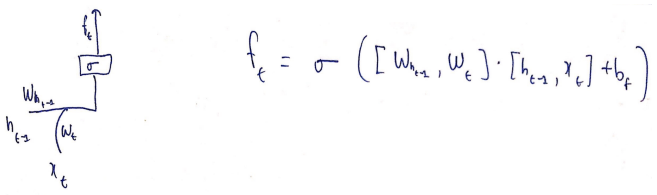
So if the context of the input (**xt**) is same as the incoming cell state (**Ct-1**), a vector of all 1's would be passed from the forget gate and no data will be forgotten and if the context of the input (**xt**) is completely different from the previous cell, then a vector of all 0's would be passed and the data would be forgotten.

At the +, new information is added or existing information is updated based on what we get from the input gate.

For example, in stocks, if the next sequence of stock prices are starkly different from the ones before it, then slowly in the consecutive layers the information regarding these prices will be forgotten and prediction will be made without a lot of influence by these previous values.

Forget Gate

The forget gate generates a number between 0 and 1 for every number in the cell state **Ct-1.** 0 implies, completely forget it and 1 implies keep this.



Weights are initialized on either side of the input of the cell and output of the previous cell. The number between 0 and 1's are generated by the sigmoid function.

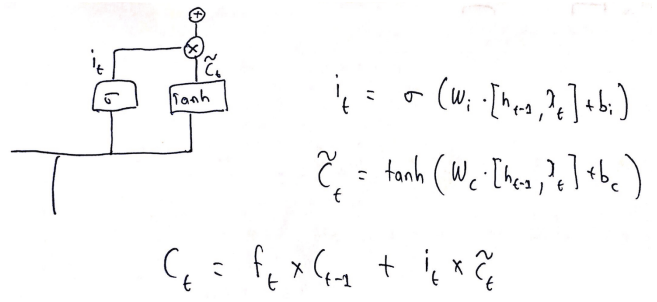
Example - Predicting next word based on the previous word. The previous cell state can include gender so that correct pronouns can be used. If the incoming input is a new subject, we'd want to forget the gender of the old subject and replace the old gender with the new gender.

Input Layer

This is where we update/add new data to the cell output **Ct**

The sigmoid decides which values could be updated. It produces an array of 0's and 1's.

The tanh function creates a new set of candidate values Ct (~). A pointwise addition operation is done and added to the cell state



Example - this is where we omit the old gender and add the new gender to the data.

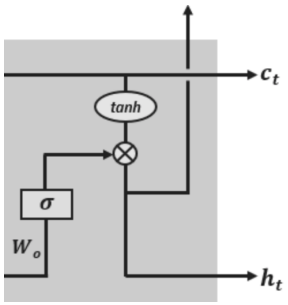
Output layer

Here we produce the output of the cell. The output is the cell state after being updated by the input layer and is filtered before being sent to the next cell.

We run a sigmoid to decide what part of the cell state we are going to output. The cell state is put through the tanh function to squash the value between -1 and +1 and is multiplied by the sigmoid to get what we want.

$$O_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = O_t * \tanh(C_t)$$



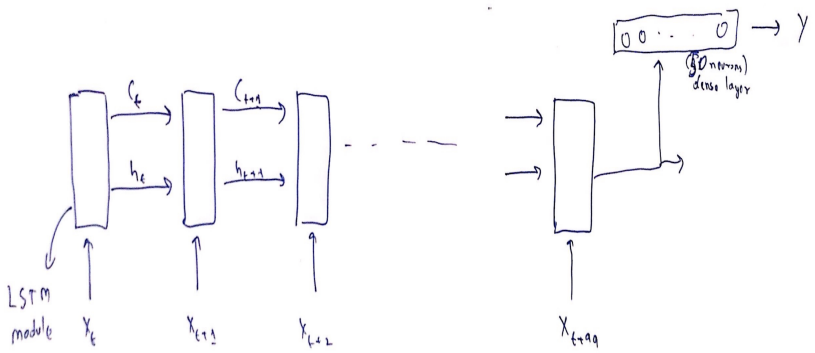
Note - If **xt** and **xt-1**are almost the same data points in terms of context then the new cell state will be the same as the previous cell state.

Note - As **Ct** is a function of **tanh**, the gradients are distributed between -1 and 1 and hence the problem of vanishing gradients is prevented.

A. The working of LSTM cells

The Xt, Xt+1, .. Xt+99 is where the 100 slots where 100 data points go in and the predicted value for the data points is Y. The output of the last LSTM cell is squashed using a 50 neuron dense layer. The cell state (Ct +99) and ht+99 are passed into the 50 neuron dense

network and then we get the predicted output. The loss function is then calculated and backprop is started .




```
# create and fit the LSTM network
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(x_train.shape[1],1)))
model.add(LSTM(units = 50))
model.add(Dropout(0.2))
model.add(Dense(1))

model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(x_train, y_train, epochs=1, batch_size=1, verbose=2)
#X = model.evaluate(x_train,y_train,batch_size=1,verbose=2)

#predicting 246 values, using past 60 from the train data
inputs = new_data[len(new_data) - len(test) - 60:].values
inputs = inputs.reshape(-1,1)
inputs  = scaler.transform(inputs)

X_test = []
for i in range(60,inputs.shape[0]):
    X_test.append(inputs[i-60:i,0])
X_test = np.array(X_test)

X_test = np.reshape(X_test,(X_test.shape[0],X_test.shape[1],1))
closing_price = model.predict(X_test)
closing_price = scaler.inverse_transform(closing_price)
```

LSTM-RNN:

RNN are used in applications where the mean and variation keep changing as more data is collected. It sees applications in natural language processing, time sharing forecasting like stocks etc.

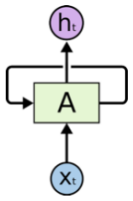
A simple example to understand the working of RNN would be a spam detector. When given a sentence the software has to detect whether it's a spam subject or not. NLP is often used here along with algorithms like BOW (bag-of-words), TF-IDF or WORD2VEC, these algorithms only consider if the words in the sentence are +ve or -ve and based on that make their prediction.

None of these algorithms consider the sequence of those words i..e the meaning of the sentence in making their predictions. This is very important in applications as well like speech detection and intent parsing in natural language processing applications like implementing voice assistants where not just the words but the sequence also carries a lot of importance.

This is what an RNN does, RNN passes the data from the previous layer to the next and the next layer may decide to retain/update/add new data to the incoming data from the previous layer.

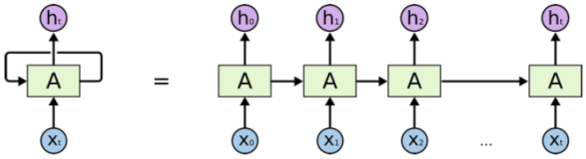
With a system like this the flow of words in a sentence is taken into account while making a prediction.

Below is the structure of an RNN



Recurrent Neural Networks have loops.

A is the hidden layer with a number of neurons inside. The input is **Xt** which is input at some time t and the output is **ht.** The output at time t is fed into the loop along with the new input at time **t+1** which gives an output **ht+1.** The outputs are continuously fed into the loop. The representative of the structure made in layers given below-



An unrolled recurrent neural network.

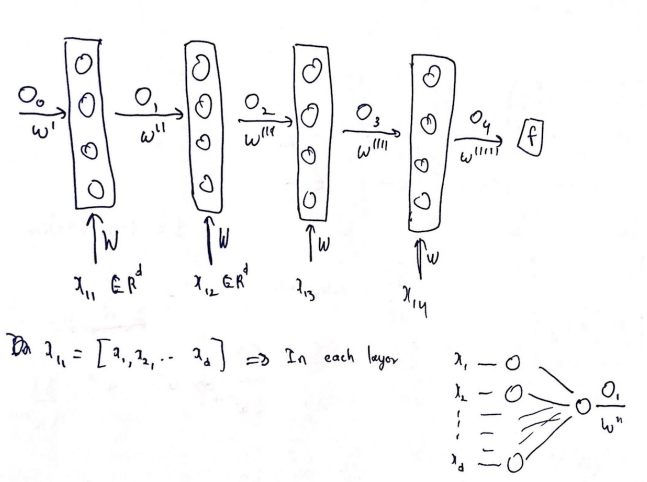
Forward propagation

The use case considered here to describe this technique is word processing.

Here the set of words are represented by **x**11,**x**12,**x**13,**x**14. The word is converted into a vector each of which is fed into the RNN in a timely manner. Let's consider the word to be a d-dimensional vector. Each word is multiplied by the same weight vector and a bias is added to the result. An activation function is then applied to this result and we get out first output **O**1. **O**1 can be represented as -

$$O1 = f(x11 * W + Oo * W')$$

The activation function - **sigmoid**



Here **O**o is the initial output that's passed along with the weight vector **W'**. They are either zero padded values or randomly initialised.The output **O**1 is multiplied by the weight **W**'' and is fed into the next layer (technically, the second loop). All this is done at time **t=1.**

At **t=2** the next word is processed. Vector **x12** is multiplied by weights **W** and added to **O1 * W'' .** The activation function is applied to this result to produce output **O2.** The resulting equation is given by -

$$O2 = f(x12 * W + W'' * O1)$$

Similarly,

$$O3 = f(x13 * W + W''' * O2)$$

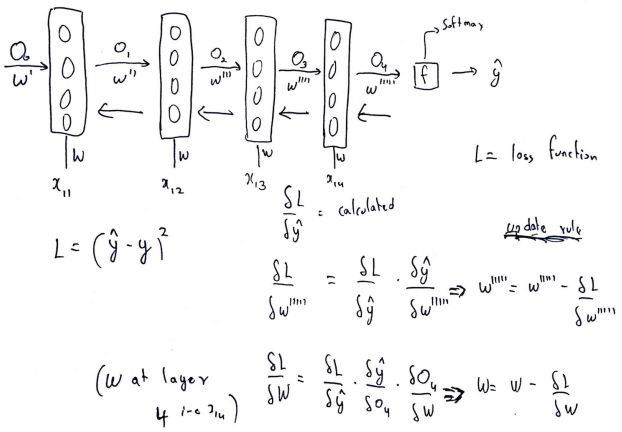
$$O4 = f(x14 * W + W'''' * O3)$$

To the output **O4** a softmax function is applied to get a binary output between 0 and 1.

In this network the outputs of each layer are retained and sent to the next layer to get the next output. Hence, **O4** is dependent on **O3** & **x14**, **O3** is dependent on **O2** & **x13**, **O2** is dependent on **O1** & **x12** and finally **O1** is dependent on **Oo** & **x11**. Therefore, this dependency allows information to be passed on.

Backward Propagation

After getting the predicted value, a loss function is used (in our case **mean squared error (MSE)** has been used) to calculate the loss and back propagation is started and keeps going till we have made the loss as minimum as possible. The weights are modified according the derivative of the loss function with respect to that weight. The derivatives are calculated using the chain rule.



All the weights are updated accordingly and once again forward propagation is started.

Problem with RNN: Vanishing Gradient

This simplistic approach is useful in a lot of places but not for stock price forecasting, the main flaw being the RNN networks run into the problem of vanishing gradients. As we move from one loop to another the derivative of the loss function becomes very small and after a point the weight vectors become stagnant. The reason for this is the fact that we are using sigmod as our activation function and we use the same weight vector W for each

input **x1i**. The derivative of sigmoid lies between 0 and 0.25 and this when multiplied by another number makes the result even smaller, hence the vanishing gradient problem.

These problems are solved by modifying the RNN to make what is called a Long Short Term Memory model. In this, not just the last but a number of elements before the element being processed contribute directly to the current output.

```
# create and fit the LSTM network
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(x_train.shape[1],1)))
model.add(Dropout(0.2))
model.add(LSTM(units = 50, return_sequences = True))
model.add(Dropout(0.2))
model.add(LSTM(units = 50, return_sequences = True))
model.add(Dropout(0.2))
model.add(LSTM(units = 50))
model.add(Dropout(0.2))
model.add(Dense(units = 1))

model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(x_train, y_train, epochs=10, batch_size=10, verbose = 2)

#predicting 246 values, using past 60 from the train data
inputs = new_data[len(new_data) - len(test) - 60:].values
inputs = inputs.reshape(-1,1)
inputs  = scaler.transform(inputs)

X_test = []
for i in range(60,inputs.shape[0]):
    X_test.append(inputs[i-60:i,0])
X_test = np.array(X_test)

X_test = np.reshape(X_test,(X_test.shape[0],X_test.shape[1],1))
closing_price = model.predict(X_test)
closing_price = scaler.inverse_transform(closing_price)
```

Accuracy Indicators:

1) Mean Square Error(MSE)/Root Mean Square Error(RMSE)

While R Square is a relative measure of how well the model fits dependent variables, Mean Square Error is an absolute measure of the goodness of fit.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

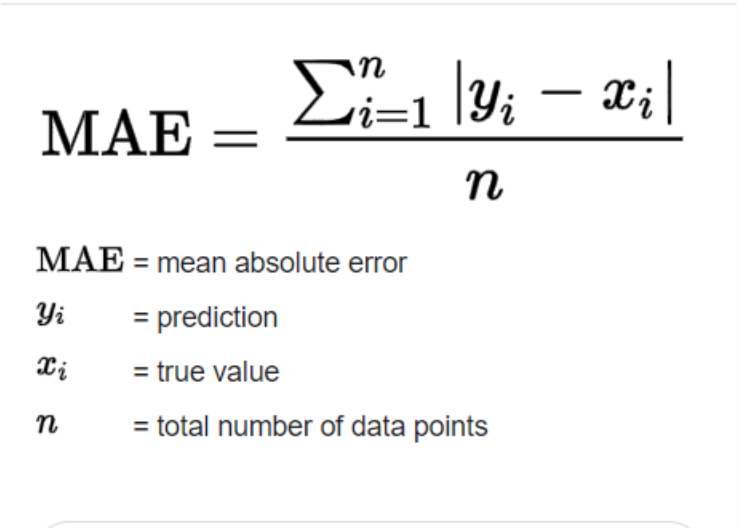
MSE = mean squared error
n = number of data points
Y_i = observed values
$\hat{Y}_i$ = predicted values

MSE is calculated by the sum of squares of prediction error which is real output minus predicted output and then divided by the number of data points. It gives you an absolute number on how much your predicted results deviate from the actual number. You cannot interpret many insights from one single result but it gives you a real number to compare against other model results and help you select the best regression model.

Root Mean Square Error(RMSE) is the square root of MSE. It is used more commonly than MSE because sometimes MSE values can be too big to compare easily. Secondly, MSE is calculated by the square of error, and thus square root brings it back to the same level of prediction error and makes it easier for interpretation.

2)Mean Absolute Error(MAE)

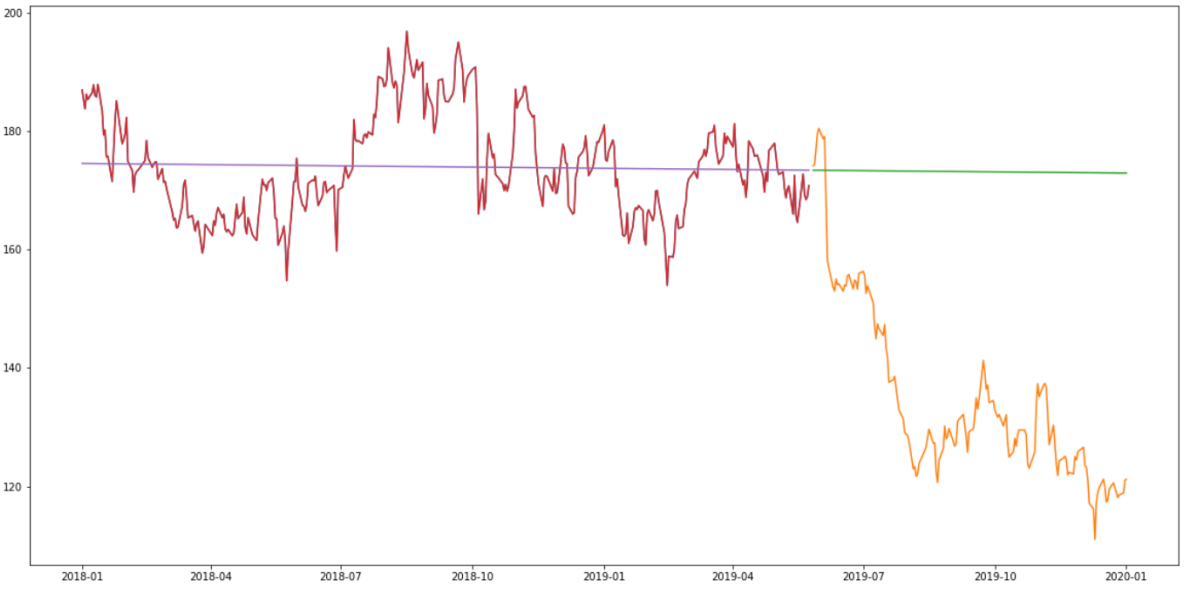
Mean Absolute Error(MAE) is similar to Mean Square Error(MSE). However, instead of the sum of square of error in MSE, MAE is taking the sum of absolute value of error.It gives us the measure of how far the predictions were from the actual output. However, they don't give us any idea of the direction of the error i.e. whether we are under predicting the data or over predicting the data. Compared to MSE or RMSE, MAE is a more direct representation of the sum of error terms. MSE gives larger penalisation to big prediction error by square it while MAE treats all errors the same.



Implementation and Results :

We began this study with **Linear Regression**

ICICI BANK



GAIL

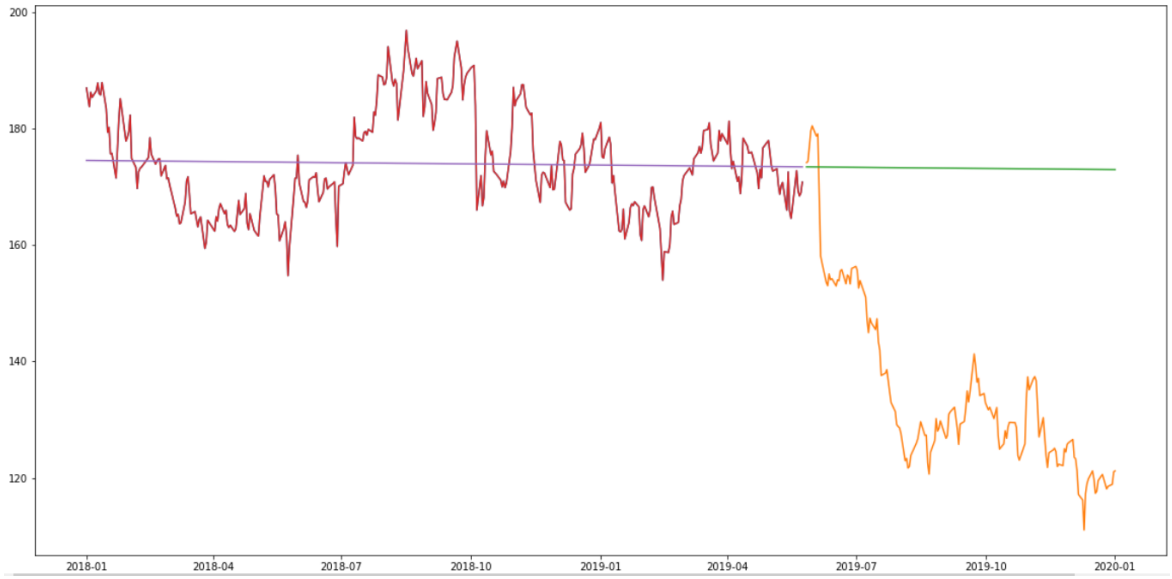
ONGC

	RMSE: TRAIN DATA	RMSE: TEST DATA
ICICIBANK	25.5895	57.3500
GAIL	8.4087	41.047
ONGC	11.0870	13.6549

These will be our base value which we aim to minimise.

We try one more linear type model called **Lasso Regressor**

ICICI BANK



GAIL

ONGC

	RMSE: TRAIN DATA	RMSE: TEST DATA
ICICIBANK	25.5895	57.3711
GAIL	8.4087	41.0725
ONGC	11.0870	13.6472

This type of linear regressor removes redundant features and shrinks weights if necessary. We see small changes in weights but no feature was deemed redundant as the data is sequential in nature. Therefore very small changes are seen in the values reported above as well.

We now shift our to Non Linear models in this report.

KNN. It's a neural network type model implementation.

ICICI BANK

GAIL

ONGC

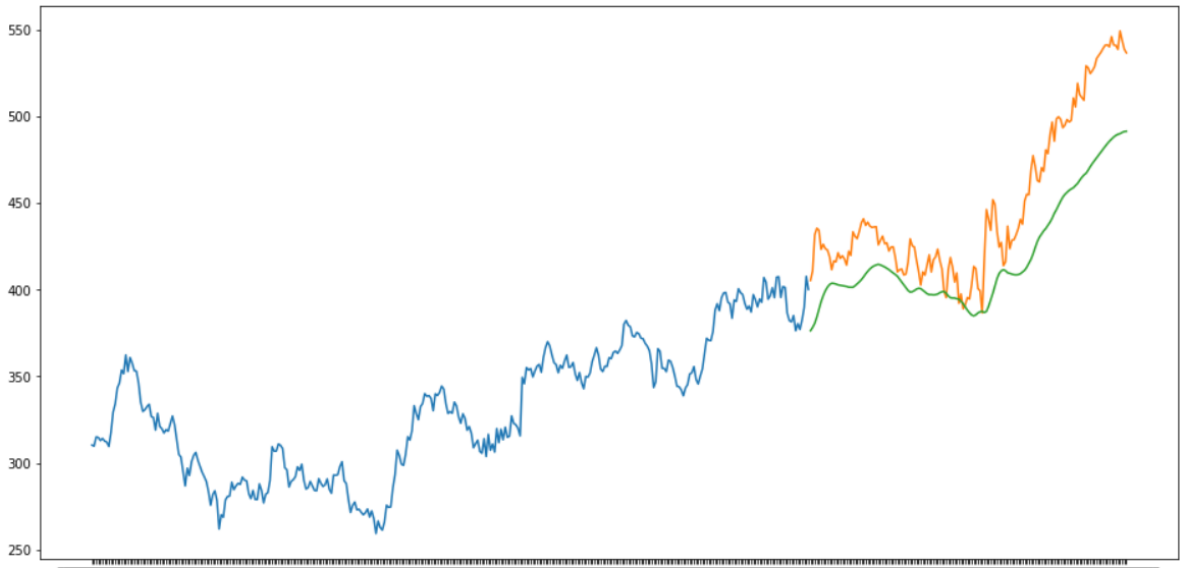
	RMSE: TRAIN DATA	RMSE: TEST DATA
ICICIBANK	3.1248	121.0556
GAIL	1.6309	42.7172
ONGC	1.5192	29.8479

The above image shows a close overlap between predicted values and actual values for train data. Therefore we see very small RMSE value for train data. This is a case of over complexity in the model. The model is highly accurate for train data therefore losing its generalisation ability and results in very poor performance on test data.

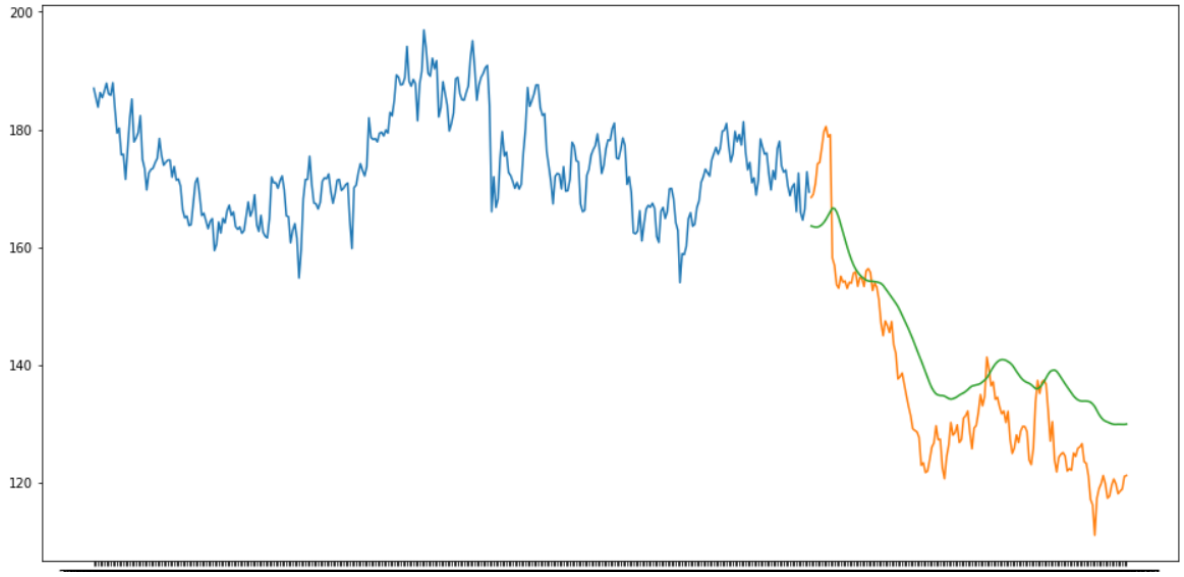
As this model is a case for over complexity, it is also interesting to note that it is unable to follow the sudden change in trend for the stock GAIL. We need to tackle this problem.

We need to minimize both variance and bias such that RMSE for TEST DATA is minimised, not just RMSE for TRAIN DATA.

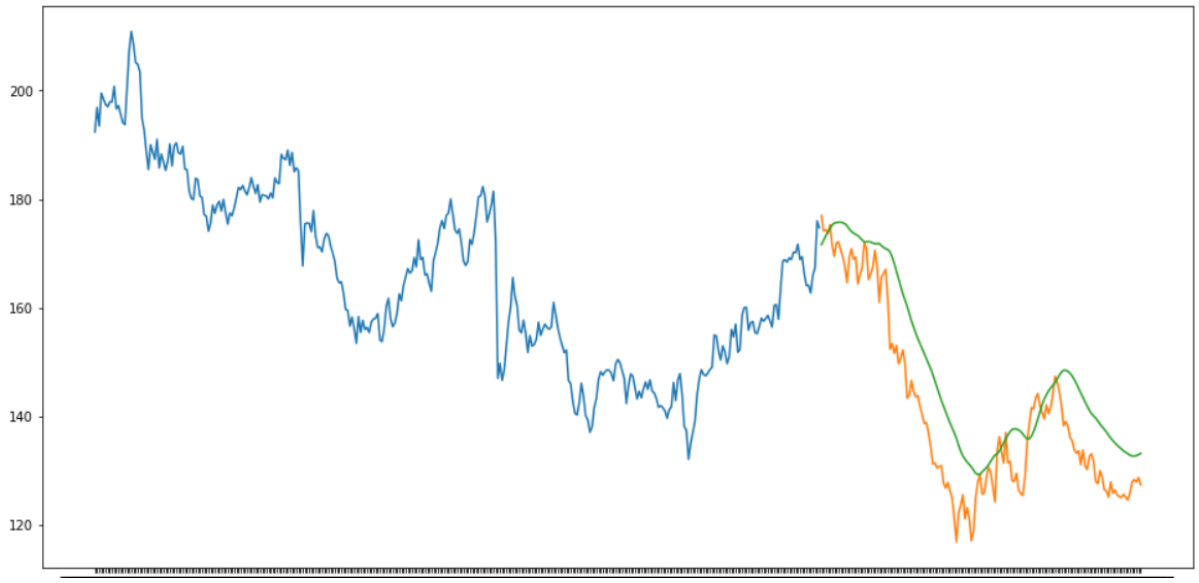
Another Neural network that we implement is **LSTM.** We have chosen to remember 60 days worth of data for our LSTM black box



ICICI BANK



GAIL



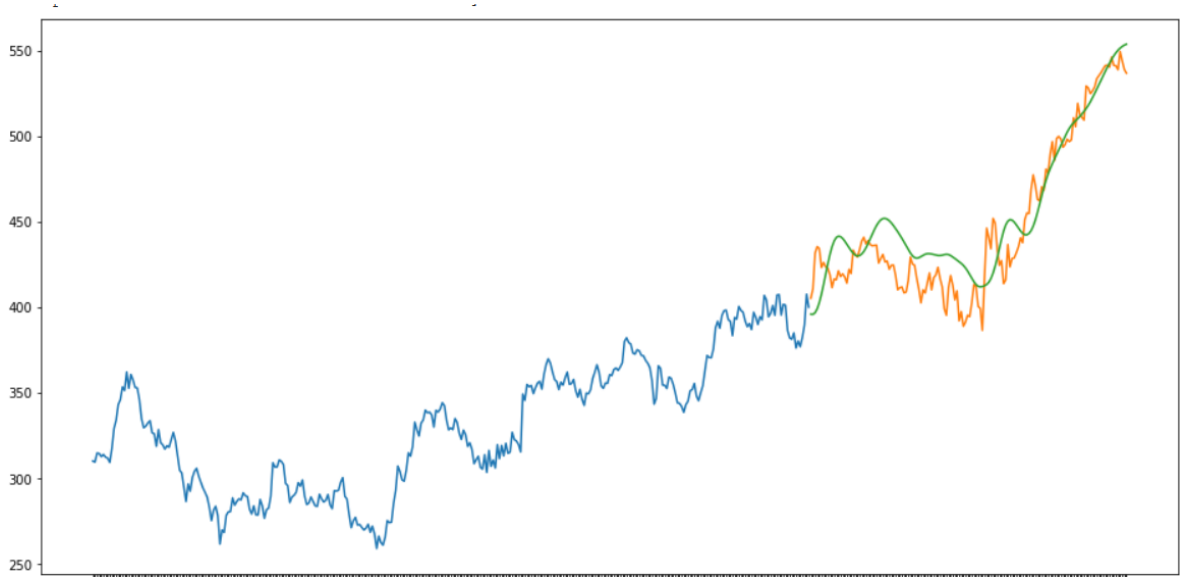
ONGC

Even before reporting the values we see a very good trend of prediction line following the actual line. But as the model is run on a neural network we have ran it 5 times and taken the mean of these metrics as well.

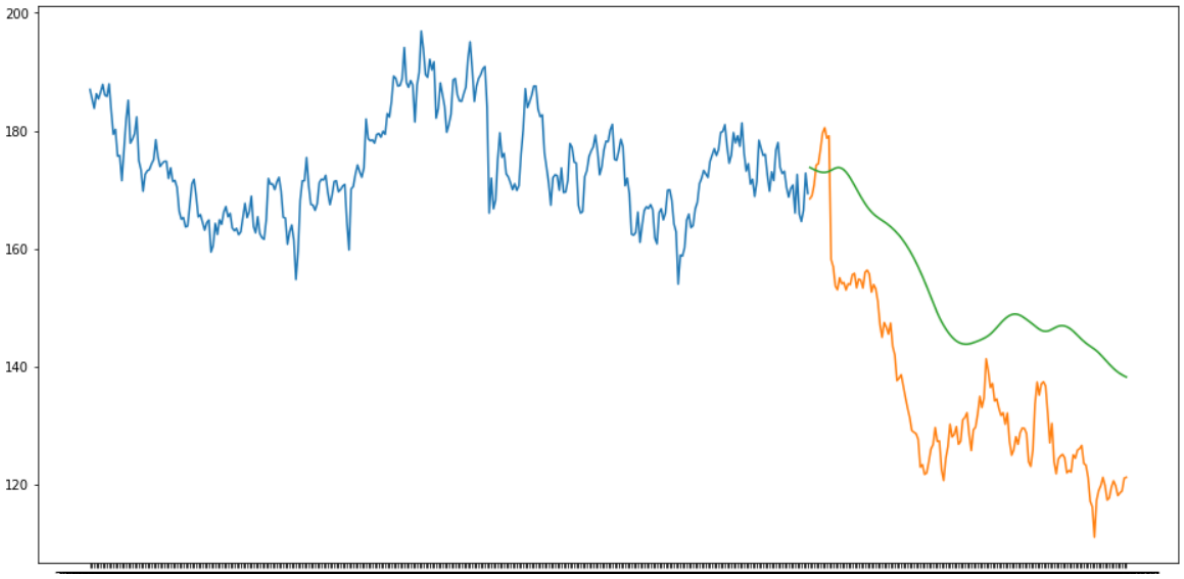
	TEST 1	TEST 2	TEST 3	TEST 4	TEST 5	MEAN
ICICI BANK	23.9800	23.5764	43.2541	17.4907	36.1801	28.8962
GAIL	13.5627	10.6294	13.4094	22.0085	12.9647	14.5149
ONGC	9.1690	12.4163	9.7112	8.0333	6.3083	9.1276

We have reduced complexity but now we will increase the complexity by implementing RNN in LSTM to check if we have not over pruned the complexity by implementing LSTM.

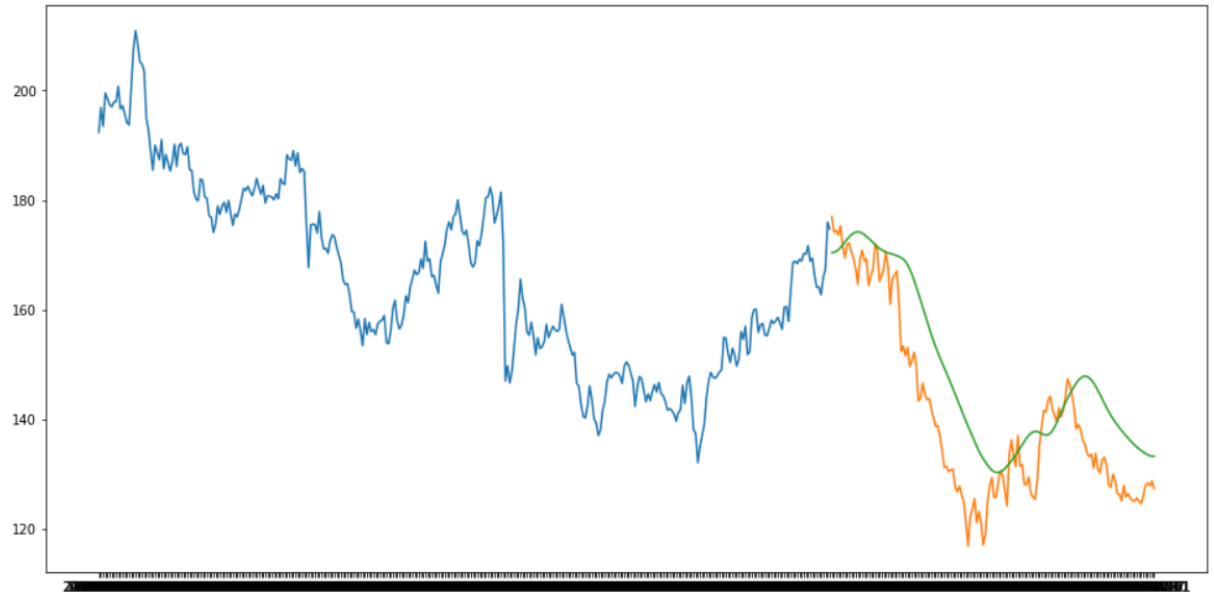
LSTM RNN Model ran has 4 layers and is ran for 10 iterations with batch learning implemented. Batch of size 10-is used.



ICICI BANK



GAIL



ONGC

Again as the model is run on a neural network we have ran it 5 times and taken the mean of these metrics as well.

	TEST 1	TEST 2	TEST 3	TEST 4	TEST 5	MEAN
ICICI BANK	31.3150	30.0731	25.3738	21.74764	36.7183	29.0455
GAIL	19.4842	21.4351	19.1100	22.0085	22.9647	21.0005
ONGC	10.2661	7.3627	9.7112	7.4038	6.9138	8.3315

With increased NN layers we see better retaining ability of experience from previous data but it fails if the actual data sees a sharp change in trend. Small improvements can be seen sometimes if the dataset has a similar behaviour but still sometimes LSTM is better over LSTM RNN. Therefore Concluding LSTM and LSTM with RNN both are better modals. And it is advisable to manage the NN layers depending on the amount of data the model is being trained on. Number of lags chosen for NN models was 1 as the data was small

enough to be implemented with that. The choice of Increasing layers needs a Domain expert as some datasets have volatile nature in movement and LSTM is better to handle them at the cost of the prediction might overgeneralise the result.

So one such use for above models could be that LSTM is implemented to check the nature of data. If volatile change in the dataset is predicted other ways to predict a sharper result need to be researched upon. But if found otherwise LSTM with RNN can prove very useful. (example: dataset ONGC).

So it can be concluded that model to be deployed at this stage of the report can be a mix of LSTM and LSTM RNN

Stock Selection Model using K-Means:

This report is on a clustering model that we tried implementing out of interest. This was not included in our main project idea submission. We have used K-Means clustering for 11 stock prices.

```
companies_dict = {
 'ICICIBANK': 'ICICIBANK.NS',
 'SBI': 'SBIN.NS',
 'HDFCBANK': 'HDFCBANK.NS',
 'GAIL': 'GAIL.NS',
 'INFOSYS': 'INFY.NS',
 'RELIANCE':'RELIANCE.NS',
 'HCL': 'HCLTECH.NS',
 'SAIL': 'SAIL.NS',
 'NTPC': 'NTPC.NS',
 'ONGC': 'ONGC.NS',
 'IOC': 'IOC.NS',
}
```

We have used 3 Stocks from BANK sector of NSE and we expect them to be clustered together.

K-Means:

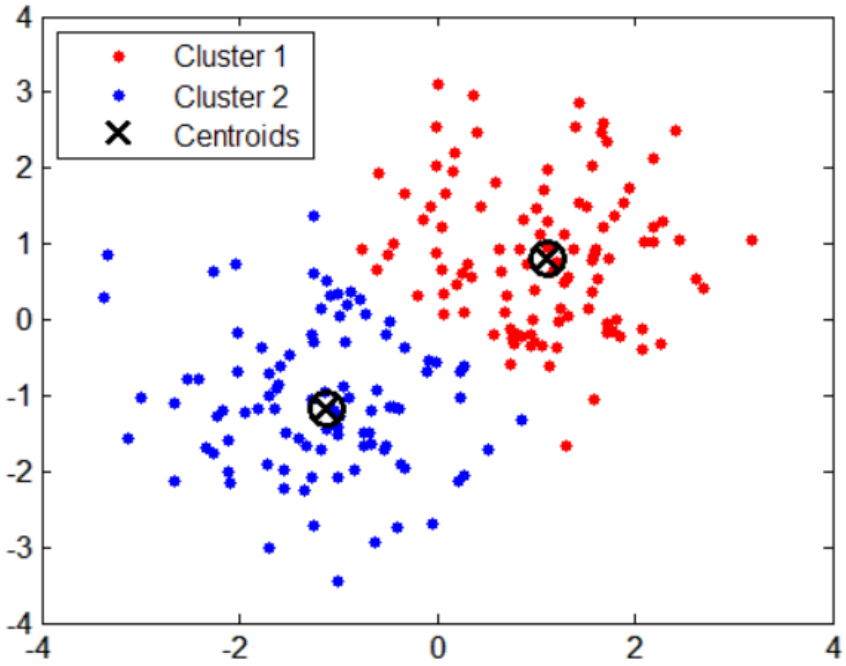
Kmeans algorithm is an iterative algorithm that tries to partition the dataset into Kpre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to only one group. It tries to make the intra-cluster data points as similar as possible while also keeping the clusters as different (far) as possible. It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster's centroid (arithmetic mean of all the data points that belong to that cluster) is at the minimum. The less variation we have within clusters, the more homogeneous (similar) the data points are within the same cluster.

The way kmeans algorithm works is as follows:

Specify number of clusters K.

Initialize centroids by first shuffling the dataset and then randomly selecting K data points for the centroids without replacement.

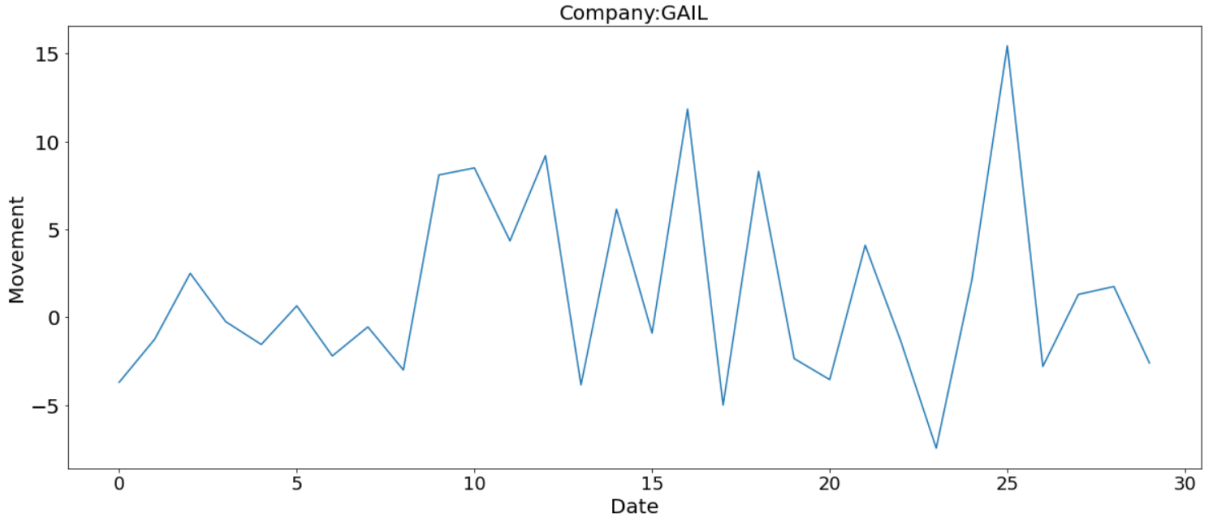
Keep iterating until there is no change to the centroids. i.e assignment of data points to clusters isn't changing.



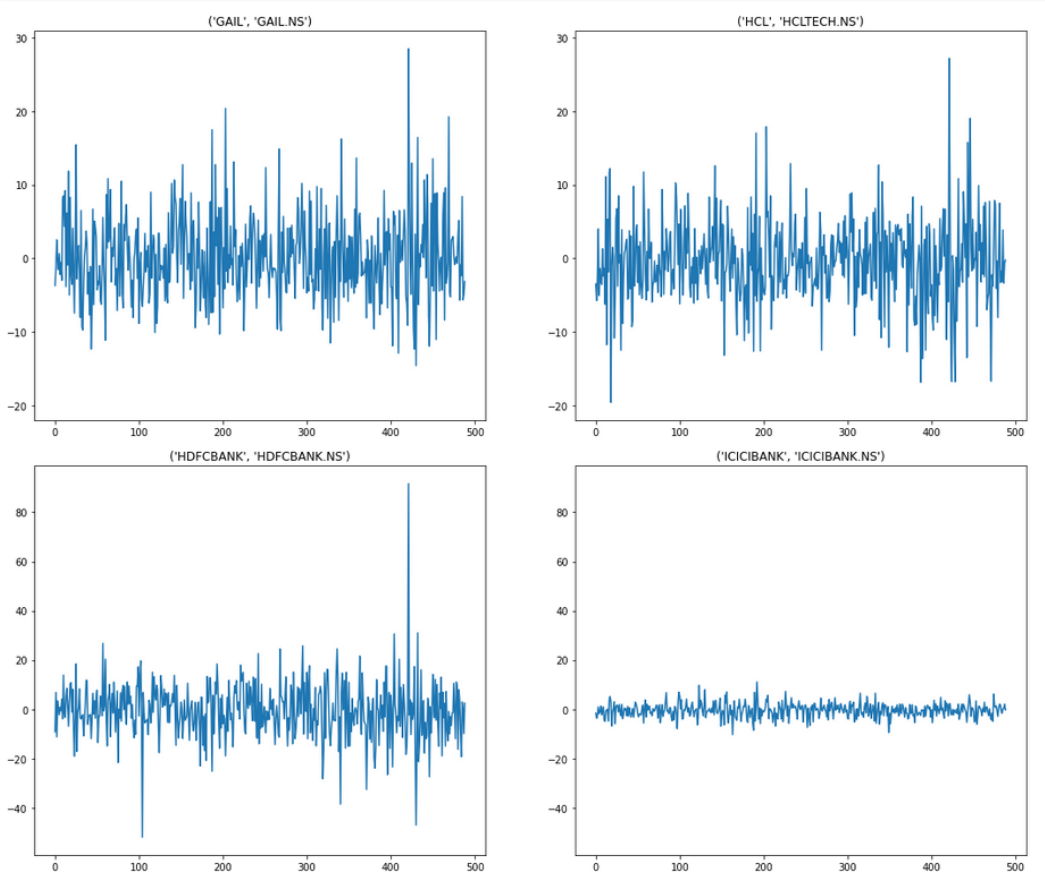
The approach kmeans follows to solve the problem is called Expectation-Maximization. The E-step is assigning the data points to the closest cluster. The M-step is computing the centroid of each cluster.

In the model we have only used Opening Price and Closing Price from raw data, and this raw data is again extracted from Yahoo Api therefore can be reused with different stocks and can be scaled to more quantity as well.

We have used stock price movement for this clustering. For example:



Data such as above for each stock is made via subtracting Opening and closing prices date wise.



Before clustering a need for normalising the data arises as it can be implied by some of the graphs above, therefore it was also implemented.

K-Means clustering was done for 5 clusters and uses the movements as pipeline

	labels	companies
0	0	(GAIL, GAIL.NS)
1	0	(HCL, HCLTECH.NS)
7	0	(ONGC, ONGC.NS)
3	1	(ICICIBANK, ICICIBANK.NS)
8	1	(RELIANCE, RELIANCE.NS)
9	1	(SAIL, SAIL.NS)
10	1	(SBI, SBIN.NS)
4	2	(INFOSYS, INFY.NS)
6	2	(NTPC, NTPC.NS)
2	3	(HDFCBANK, HDFCBANK.NS)
5	4	(IOC, IOC.NS)

BANKS USUALLY HAVE SAME MOVEMENTS, SO WE EXPECT THEM TO BELONG TO SIMILAR CLUSTERS.
WE SEE ONLY **HDFC BANK** WHICH DOES NOT BELONG TO A SIMILAR CLUSTER.

WHEN MODEL IS RAN MULTIPLE TIMES WE GET FAIRLY SIMILAR CLUSTER FORMATION.

Conclusion:

We started with linear models by implementing Linear regression and Lasso regression, to which we did not get conclusive results, so we moved on to non linear models. We started with KNN which did not give us the best results due to the over complexity of the model. Further, we try to implement the LSTM model, we could see that the prediction line almost follows the actual line and gives better RMSE values. We decided to increase the complexity by implementing RNN in LSTM.With increased NN layers we see better retaining ability of experience from previous data but it fails if the actual data sees a sharp change in trend. So it can be concluded that the model to be deployed at this stage of the report can be a mix of LSTM and LSTM RNN.

References:

[1] K.Alkhatib, H.Najadat, I.Hmeidi and M.K.A.hatnawi, “Stock Price Prediction Using K-Nearest Neighbor (kNN) Algorithm” International Journal of Business, Humanities and Technology, vol. 3, pp. 32–44, March 2013.

[2] S. Lauren and S. D. Harlili, "Stock trend prediction using simple moving average supported by news classification," *2014 International Conference of Advanced Informatics: Concept*, *Theory and Application (ICAICTA)*, Bandung, 2014, pp. 135-139, doi: 10.1109/ICAICTA.2014.7005929.

[3] K. Chen, Y. Zhou and F. Dai, "A LSTM-based method for stock returns prediction: A case study of China stock market," 2015 IEEE International Conference on Big Data (Big Data), Santa Clara, CA, 2015, pp. 2823-2824, doi: 10.1109/BigData.2015.7364089.

[4] Franklin, S Joel. “Stock Market Clustering with K-Means Clustering in Python.” *Medium*, DataDrivenInvestor, 9 Dec. 2019, medium.datadriveninvestor.com/stock-market-clustering-with-k-means-clustering-in-python-4bf6bd5bd685

[5] Hegazy, Osman & Soliman, Omar S. & Abdul Salam, Mustafa. (2013). A Machine Learning Model for Stock Market Prediction. International Journal of Computer Science and Telecommunications. 4. 17-23.

[6] Brownlee, Jason. “A Gentle Introduction to Long Short-Term Memory Networks by the Experts.”MachineLearningMastery,19Feb,2020

machinelearningmastery.com/gentle-introduction-long-short-term-memory-networks-experts.

[7] Yan, Shi. “Understanding LSTM and Its Diagrams - ML Review.” *Medium*, 21 June 2018, blog.mlreview.com/understanding-lstm-and-its-diagrams-37e2f46f1714.

[8] GeeksforGeeks. “LSTM - Derivation of Back Propagation through Time.” *GeeksforGeeks*,7Aug.2020, www.geeksforgeeks.org/lstm-derivation-of-back-propagation-through-time.

[9] Harrison, Onel. “Machine Learning with the K-Nearest Neighbors Algorithm.” *Medium*, 14July.2019 towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761.

[10] “Lasso Regression in Machine Learning.” *Dataaspirant*, 15 Dec. 2020, dataaspirant.com/lasso-regression/#t-1606404715787.

[11] Katari, Kaushik. “Linear Regression Model: Machine Learning - Towards Data Science.”*Medium*,10Oct,2020, towardsdatascience.com/linear-regression-model-machine-learning-9853450c8bce.

[12] “Linear Regression — ML Glossary Documentation.” *M*, 20 Jan. 2020, ml-cheatsheet.readthedocs.io/en/latest/linear_regression.html.

Future study requires:

Larger more precise data. Meaning data needs more points per day, like 15min data per day for a year or two. More features. *Such as, features that are not only divided into sequential days rather it is also divided depending on sequential weeks or months to capture the nature of stock around weekends, monthends or quarter ends*. More models or implementation of models together for different results.

In this report the p[redictions are made are not precise to be used everyday, rather it gives an hint to where the movement of the stock will go. With more precise data we can make a model that can help traders use it for day trading and or atleast for swing trading.

Day Traders: Traders whose trades depend on the trades they make on a single week day. They need to enter and exit the market the same day.

Swing Traders: Traders whose trades depend on the trades they make over a few weeks, not months.