Backtesting of investment strategies and stock prediction

Syed Anas, Vaishnav R Krishnan, Shaikh Abdul Aahad, Satvik Reddy T

ABSTRACT

People lose money not because they lack understanding of the market. But simply because their trading decisions are not based on sound research and tested trading methods. Backtesting a trading strategy is the process of testing a trading hypothesis/strategy on the historical data. You can improve your likelihood of success in trading by backtesting your trading rules on historical data. In this project python programming language with machine learning and concepts like sum of moving averages (SMA) was used to compare different trading strategies, the analytical results produced will enable traders and researchers to make valid decisions. Machine learning models like polynomial regression and LSTM was used to predict the future stock prices, and the best backtesting strategy can be applied to the data to get the best returns on trading.

1. Introduction

The finance industry has adopted machine learning (ML) as a form of quantitative research to support better investment decisions. Backtesting is considered to be an important tool in a trader's toolbox. Without backtesting, traders wouldn't even think of risking money into the financial markets.

1.1 What is backtesting

Backtesting a trading strategy is the process of testing a trading hypothesis/strategy on the historical data.

Let's say you formed a hypothesis. This hypothesis states that securities that have positive returns over the past one year are likely to give positive returns over the next one month.

By using historical data, you can backtest and see whether your hypothesis is true or not. It helps assess the feasibility of a trading strategy by discovering how it performs on the historical data.

If you backtest your strategy on the historical data and it gives good returns, you will be confident to trade using it. If the strategy is performing poorly on the historical data, you will discard or re-evaluate the hypothesis.

1.2 Moving average

A moving average is a technical indicator that market analysts and investors may use to determine the direction of a trend. It sums up the data points of a financial security over a specific time period and divides the total by the number of data points to arrive at an average. It is called a "moving" average because it is continually recalculated based on the latest price data.

The simple moving average (SMA) is a straightforward technical indicator that is obtained by summing the recent data points in a given set and dividing the total by the number of time periods. Traders use the SMA indicator to generate signals on when to enter or exit a market. An SMA is backward-looking, as it relies on the past price data for a given period. It can be computed for different types of prices, i.e., high, low, open, and close. In financial markets, analysts and investors use the SMA indicator to determine buy and sell signals for securities. The SMA helps to identify support and resistance prices to obtain signals on where to enter or exit a trade.

The formula for Simple Moving Average is written as follows:

SMA = (A1 + A2 + …….An) / n

Example:

SMA = (₹23 + ₹23.40 + ₹23.20 + ₹24 + ₹25.50) / 5

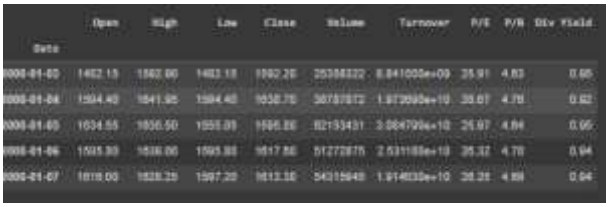
SMA = ₹23.82

Long short term memory (LSTM) is a model that increases the memory of recurrent neural

networks. Recurrent neural networks hold short term memory in that they allow earlier determining information to be employed in the current neural networks.

1.2 Dataset

In this project the dataset of NIFTY 50 from national stock exchange was considered. The data of past 10 years was taken which had the values of date, opening price, closing price, High, low, volume, Turnover, P/E ratio columns. The data set had 5000 rows of data. The data set used here is non linear in nature.



Date	Open	High	Low	Close	Volume	Turnover	P/E	P/B	Div Yield
[illegible]	1462.15	1562.90	1462.15	1562.20	25388332	8.841000e+09	25.91	4.83	0.65
[illegible]	1564.40	1641.95	1564.40	1638.70	38787872	1.973898e+10	28.07	4.78	0.62
[illegible]	1634.55	1636.50	1555.05	1595.80	82153431	3.084799e+10	25.97	4.84	0.66
[illegible]	1595.30	1636.00	1595.30	1617.80	51272875	2.531188e+10	25.32	4.78	0.64
[illegible]	1618.00	1628.25	1597.20	1613.35	54315948	1.914838e+10	26.26	4.89	0.64

2. Methodology

We defined different strategies to predict when to invest or sell. A parameter called position was used which tells us when to invest or sell.

Positions:

+1 : invest (long position)

-1 : sell (short position)

0 : No position

Strategies:

Simple strategy: initially invest in the market and do nothing. (Position +1 on a given day)

So now in the first case a simple momentum strategy was tested. In simple momentum strategy, you will be investing (+1 position) if today's return was positive. Short selling(-1 position) if today's return was negative.

Fig 2.1 Simple contrarian strategy

It can be observed from Fig 2.1 that when we use a simple contrarian strategy the return on investment is very less, the simple strategy itself shows better return rate, which is just invest and do nothing.

Fig 2.2 Flow chart of investing using backtesting strategies.

Fig 2.2 shows the process of investing using by backtesting, in this process first a hypothesis is built and then the historical model built on hypothesis is refined till the desired satisfactory performance is achieved.

```
def summary_ann(returns):
    summary = returns.agg(["mean", "std"]).T
    summary["Return"] = summary["mean"] * 252
    summary["Risk"] = summary["std"] *np.sqrt(252)
    summary.drop(columns = ["mean", "std"], inplace = True)
    return summary
```

From the above code, 252 was multiplied to the mean value because there are 252 trading days in a year. In later cases more complex strategies were used to get better returns. For this the sum of moving averages were calculated. There are many types of moving averages traders can use these days, like the SMA (Simple Moving Average) or EMA (Exponential Moving Average), but the interpretation is the same.

The reason why they differ comes from the way they are calculated. An SMA, for instance, averages the closing price of the previous periods to calculate a value corresponding to the current price.

For any moving average, the more extended the period considered, the flatter the line will become.

Trader's use moving averages to divide the market between bullish and bearish conditions. As such, when the price sits above the moving average, the market is bullish, and traders look to buy the dips. Conversely, when it sits below the moving average, the market is bearish, and traders sell the spikes. In time, traders started using multiple moving averages on the same chart. The idea is to combine their periods in such a way to obtain bullish and bearish signals.

And so did the concept of a **golden and death cross** appear. It uses two SMA's: SMA (200) and SMA (50). As the name suggests, a golden cross show bullish condition. That is, when the SMA (50) crosses above the SMA (200), the market becomes bullish, and traders will look to buy into support.

A death cross appears when the SMA (50) moves below the SMA (200). The price shows bearish conditions and will meet resistance every time it spikes into the SMA (50) or SMA (200).

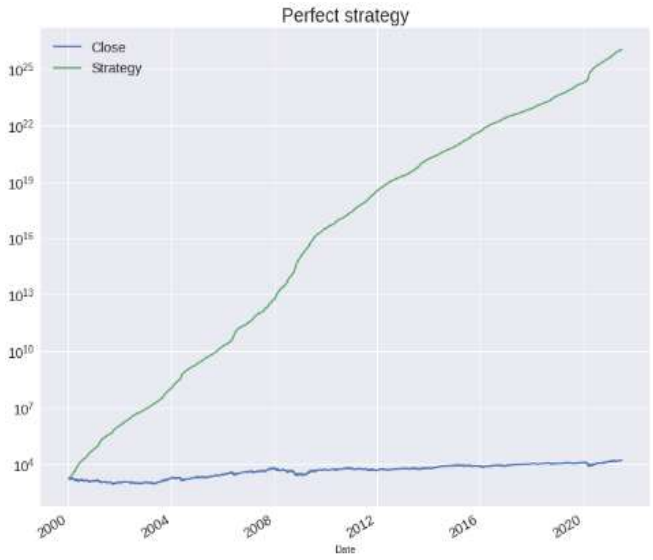
So in our strategy when SMA(50) > SMA(200), investing(position +1) was considered, and when SMA(50) < SMA(200) (position -1) short position or sell was considered.

Fig 2.3 SMA strategy

From Fig 2.3, it can be seen that when the SMA (50) is greater than SMA(200) the market is bullish and when SMA(200) is greater the market is bearish. When the SMA(50) value is greater its time to invest (position +1) and when the SMA(200) value is greater than SMA(50) value its time to sell (position -1) as seen in Fig 2.3. This is because SMA(50) is a more recent data of the recent days compared to the SMA(200) which takes the data of last 200 trading days.

Later by tuning the parameters a perfect strategy was found for trading. Where the returns where very high similar to the invest and hold strategy.

```
df["Strategy_Return"] = df["Position"] *df["Return"]
df["Strategy"] = df.Strategy_Return.add(1, fill_value
 = 0).cumprod() * df.iloc[0, 0]
df[["Close", "Strategy"]].plot(figsize = (12, 10), fo
ntsize = 15, logy=True )
plt.legend(fontsize = 15)
plt.title("Perfect strategy", fontsize = 20)
plt.show()
```



It can be seen that the prefect strategy gives better returns that the simple strategy of invest and hold. The green line shows the return of the better strategy.

3. Prediction of stock price using machine learning.

```
df["Strategy_Return"] = df["Position"].shift() * df["Return"]

from sklearn.model_selection import train_test_split

train, test = train_test_split(df, test_size=0.10)

from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree=5, include_bias=False)

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

poly_features = poly.fit_transform(X_train.reshape(-1,1))

poly_features = poly.fit_transform(X_train.reshape(-1,1))
```

```
from pandas.core.common import random_state
model = LinearRegression()
svr_poly = SVR(kernel="rbf", C=100, gamma="auto")
#model_rf = RandomForestRegressor(n_estimators=500, o
ob_score=True, random_state=100)
```

```
# Fit linear model using the train data set
model.fit(poly_features, y_train)
fitting = model.predict(poly_features)
#model_rf.fit(X_train, y_train)
fitting
plt.figure(1, figsize=(16,10))
plt.title('Linear Regression | Price vs Time')
plt.scatter(X_train, y_train, edgecolor='
w', label='Actual Price')
plt.plot(X_train, fitting, color='r', lab
el='Predicted Price')
plt.legend()
plt.show()
```

The above codes were used to predict the future stock prices of the NIFTY 50 dataset. A machine learning algorithm called polynomial regression was used.

Polynomial Regression is a regression algorithm that models the relationship between a dependent(y) and independent variable(x) as nth degree polynomial. The Polynomial Regression equation is given below.

$$y= b_0+b_1x_1+ b_2x_1^2+ b_3x_1^3+ b_4x_1^4 + b_5x_1^5$$

In the regression model used here a 5^{th} degree polynomial regression algorithm is used.

It is a linear model with some modification in order to increase the accuracy. It makes use of a linear regression model to fit the complicated and non-linear functions and datasets. In Polynomial regression, the original features are converted into Polynomial features of required degree (2,3,..,n) and then modelled using a linear model.

The need of Polynomial Regression:

If we apply a linear model on a linear dataset, then it provides us a good result as we have seen in Simple Linear Regression, but if we apply the same model without any modification on a non-linear dataset, then it will produce a drastic output. Due to which loss function will increase, the error rate will be high, and accuracy will be decreased.

So, for such cases, where data points are arranged in a non-linear fashion, we need the Polynomial Regression model. We can understand it in a better way using the below comparison diagram of the linear dataset and non-linear dataset.

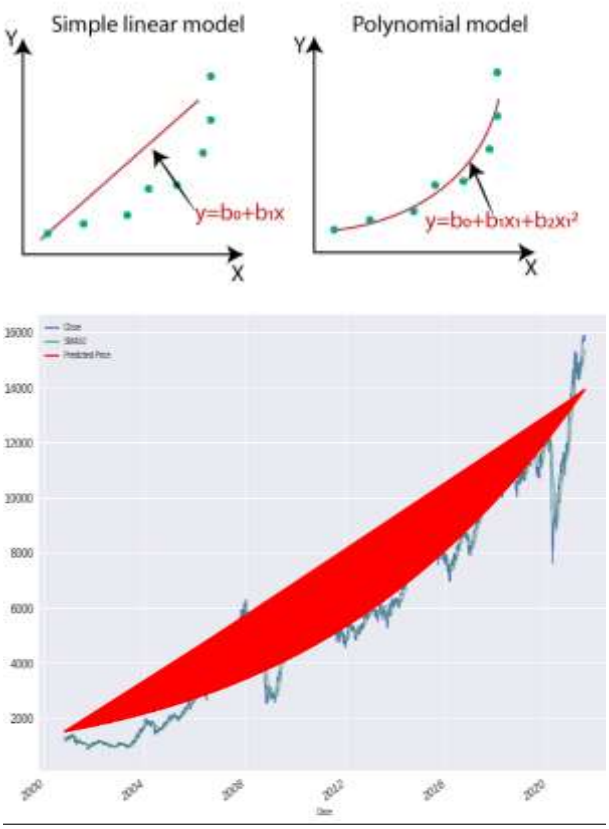
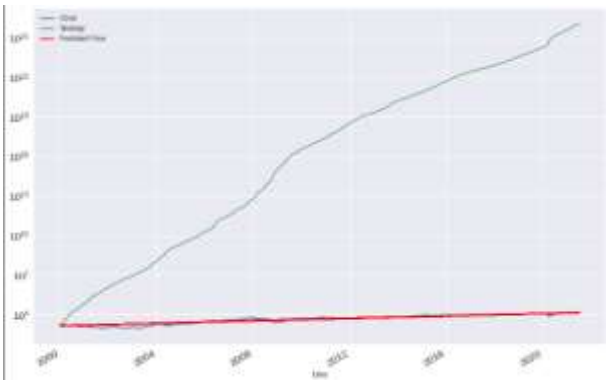


Fig 3.1 stock price prediction using Polynomial regression

Fig 3.1 shows the stock price predicted using polynomial regression for NIFTY 50, Sum of moving average strategy was used for the comparison.



Similarly, from above figure it can be seen that using prefect strategy the rate of return is much higher than the predicted value of the polynomial linear regression model.

Test result

The SMA's crossed shortly after 400th time period or 2008, the strategy for the SMA cross over was tested and had given a sell signal. The LSTM plot confirms this downtrend. And one would have made money by shorting.

4. Prediction of stock price using LSTM

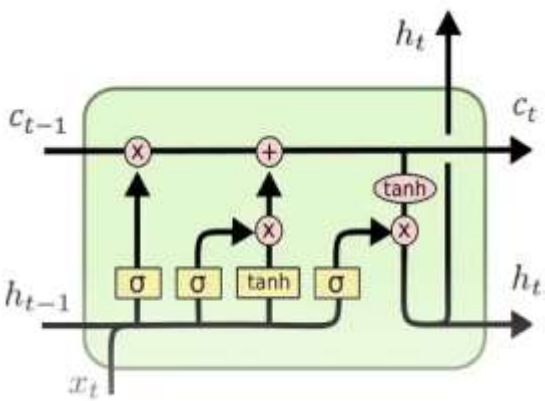
Long short term memory (LSTM) is a model that increases the memory of recurrent neural networks. Recurrent neural networks hold short term memory in that they allow earlier determining information to be employed in the current neural networks.

The compact forms of the equations for the forward pass of an LSTM cell with a forget gate are:

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

$\odot$ Denotes the Hadamard product. The subscript t indexes the time step.

- $x_t \in \mathbb{R}^d$: input vector to the LSTM unit
- $f_t \in (0,1)^h$: forget gate's activation vector
- $i_t \in (0,1)^h$: input/update gate's activation vector
- $o_t \in (0,1)^h$: output gate's activation vector
- $h_t \in (-1,1)^h$: hidden state vector also known as output vector of the LSTM unit
- $\tilde{c}_t \in (-1,1)^h$: cell input activation vector
- $c_t \in \mathbb{R}^h$: cell state vector



LSTM

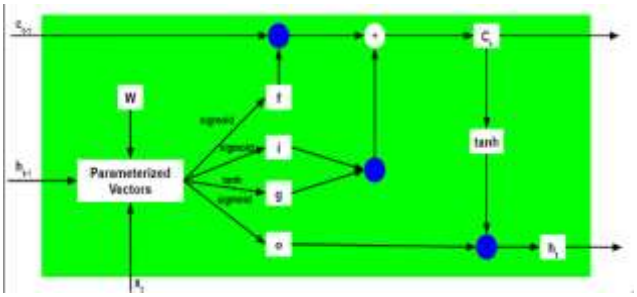
Fundamentally, we are building a NN regressor for continuous value prediction using LSTM. First, initialize the model. Then, add the 1st LSTM layer with the Dropout layer followed. Note for the LSTM layer, units is the number of LSTM neurons in the layer. 50 neurons will give the model high dimensionality, enough to capture the upwards and downward trends. We use two LSTM layers in our model and implement drop out in between for regularization. The number of units assigned in the LSTM parameter is fifty.

```
x_train = []
y_train = []
for i in range(100, df_train_arr.shape[0]):
  x_train.append(df_train_arr[i-100: i])
  y_train.append(df_train_arr[i, 0])
x_train, y_train = np.array(x_train), np.array(y_train)
from keras.layers import Dense, Dropout, LSTM
from keras.models import Sequential
```

The model had to compile 50 epochs. A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell. After fitting the data with our model, we use it for prediction. We must use inverse transformation to get back the original value with the transformed function.

working is illustrated as below:-

Fig 4.1 working of LSTM



Note that the blue circles denote element-wise multiplication. The weight matrix W contains different weights for the current input vector and the previous hidden state for each gate.

Working of an LSTM recurrent unit:

1.Take input the current input, the previous hidden state, and the previous internal cell state.

2. Calculate the values of the four different gates by following the below steps:-

a) For each gate, calculate the parameterized vectors for the current input and the previous hidden state by element-wise multiplication with the concerned vector with the respective weights for each gate.

b) Apply the respective activation function for each gate element-wise on the parameterized vectors. Below given is the list of the gates with the activation function to be applied for the gate.

3) Calculate the current internal cell state by first calculating the element-wise multiplication vector of the input gate and the input modulation gate, then calculate the element-wise multiplication vector of the forget gate and the previous internal cell state and then adding the two vectors.

$$c_t = i \odot g + f \odot c_{t-1}|$$

4) Calculate the current hidden state by first taking the element-wise hyperbolic tangent of the current internal cell state vector and then performing element-wise multiplication with the output gate.

Just like Recurrent Neural Networks, an LSTM network also generates an output at each time step and this output is used to train the network using gradient descent.

The model contains 5 layers and was trained for 70%of the data and was tested for 30%of the data. The model was successful in prediction of the trend.

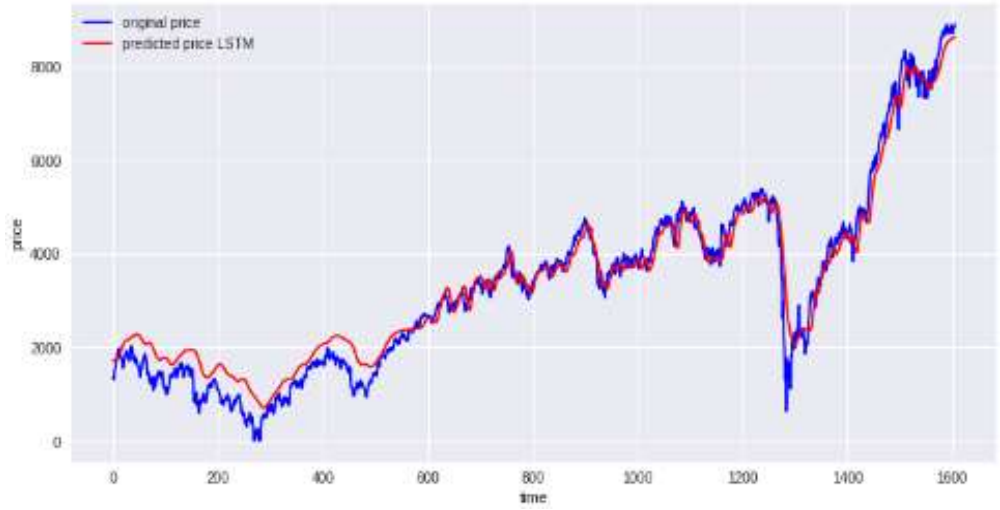


Fig 4.2 Stock price prediction using LSTM

```
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(x_train, y_train, epochs=50)
plt.figure(figsize=(12, 6))
plt.plot(y_test, 'b', label='original price')
plt.plot(y_pred, 'r', label='predicted price LSTM')
plt.xlabel('time')
plt.ylabel('price')
plt.legend()
plt.show()
```

References

[1] https://en.wikipedia.org/wiki/

[2] https://www.geeksforgeeks.org/

[3] https://towardsdatascience.com/

[4] https://www.analyticsvidhya.com/

[5] https://finance.yahoo.com/

[6] https://arxiv.org/pdf/2207.00436.pdf

[7] https://corporatefinanceinstitute.com/

[8] https://developers.google.com/

Project access links

GitHub repository link:

https://github.com/Syedanas-ctrl/LSTM.git

Google colab links:

https://colab.research.google.com/drive/1-5ECT9BFZTbPqUYF6TUuc3EnYbqLZkXV?usp=sharing

https://colab.research.google.com/drive/1JpSzmos6o6NqOYoLOGrntzDBJofSJIyx?usp=sharing

https://colab.research.google.com/drive/1Wy89e6aSqo0F2hie-2gx1iqVirBap0dA?usp=sharing

Team:

Syed Anas
Vaishnav R Krishnan
Shaikh Abdul Aahad
Satvik Reddy T

.