Stock Market Prediction

Nifty50

Problem

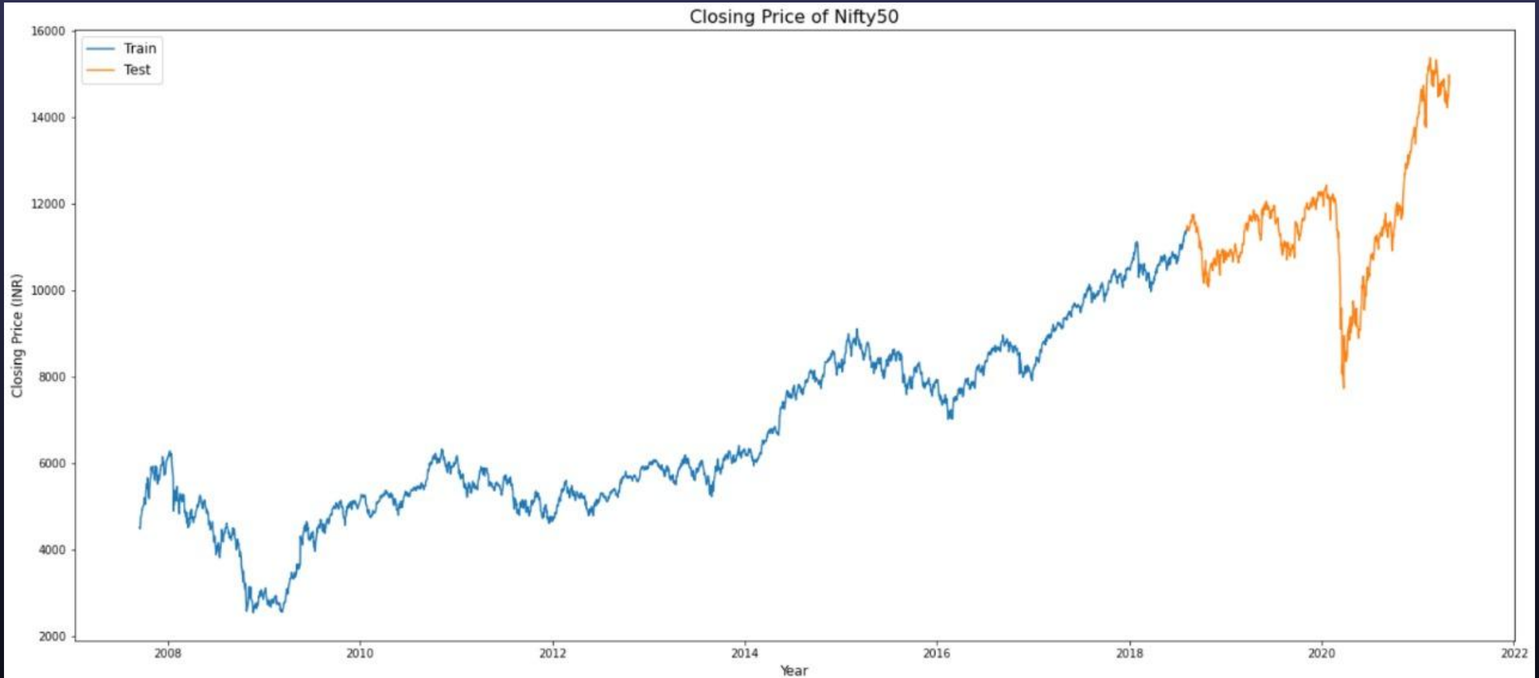
Predict the future closing value of Nifty 50

Solution

Time series forecasting models are capable of predicting future values based on previously observed values.

Dataset

Nifty 50 dataset: Yahoo Finance from 2007 to 2021



Models Used:

Model	Training Error (RMSE)	Testing Error (RMSE)
ARIMA	-	177
Vanilla LSTM	96	278
Stacked LSTM	99	390
Bidirectional LSTM	105	1430
CNN	86	210
CNN-LSTM	120	302

Preprocessing Steps

1. Removing irrelevant features
2. Imputing
3. Train-test split the time series data
4. Scaling
5. Create time windows

Preprocessing Steps

1. Removing Irrelevant Features:
   Features removed included Volume

```
cols= list(Data)[1:5]
stock_prices = Data[cols].astype(float)
```

2. Imputation:
   Data didn't contain null values, and thus no need to replace them.

Preprocessing Steps

3. Train Test Split (2/3rd Train)
   The time series data wasn't shuffled. Training was done on 2013-19 data and testing on 2020-21 data.

4. Scaling
   Standard Scalar was used to speed up computation

Normalizing Data

```
[ ] from sklearn.preprocessing import StandardScaler
    scaler = StandardScaler()
    scaler = scaler.fit(stock_prices)
    stock_prices_scaled = scaler.transform(stock_prices)
    print('df_for_training_scaled shape == {}.'.format(stock_prices_scaled.shape))

    df_for_training_scaled shape == (3326, 4).
```

Test Train Data Split

```
[ ] train_size = int(len(stock_prices_scaled) * 0.66)
    test_size = len(stock_prices_scaled) - train_size
    train, test = stock_prices_scaled[0:train_size,:], stock_prices_scaled[train_size:len(stock_prices_scaled),:]
```

Preprocessing Steps

5. Create time windows
   Time sequences were created of window size 14 days.
   Data was trained on each sample for 14 days to predict the 15th day.

```
[9]  train_inputs = []
     train_labels=[]

[10] n_future = 1   # Number of days we want to predict into the future
     n_past = 14   # Number of past days we want to use to predict the future
     for i in range(n_past, len(train) - n_future +1):
         train_inputs.append(train[i - n_past:i, 0:train.shape[1]])
         #train_opening_labels.append(train[i + n_future - 1:i + n_future, 0])
         train_labels.append(train[i + n_future - 1:i + n_future, 1])
     train_inputs, train_labels = np.array(train_inputs), np.array(train_labels)

     print('train_input shape == {}.'.format(train_inputs.shape))
     print('train shape == {}.'.format(train_labels.shape))

     train_input shape == (2181, 14, 4).
     train shape == (2181, 1).
```

Long Short Term Memory – LSTM

Long Short Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies.Remembering information for long periods of time is practically their default behavior, not something they struggle to learn! All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.

The key to LSTMs is the cell state, it does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.

Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through.

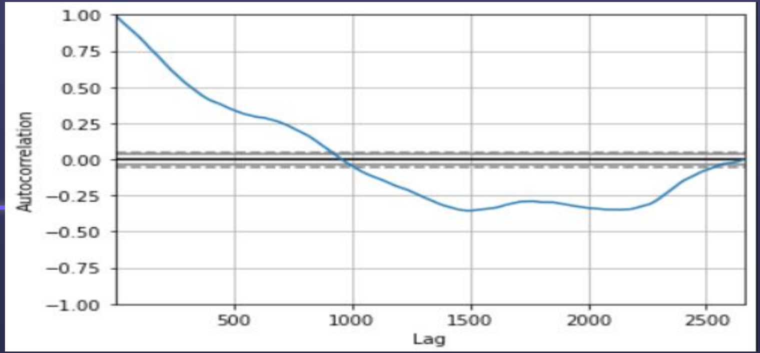
Arima

AutoRegressive Integrated Moving Average (ARIMA) is a model that captures a suite of different standard temporal structures in time series data.

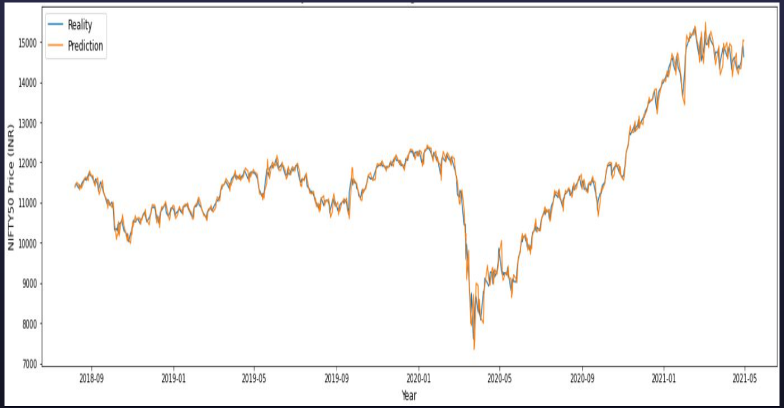
p: The number of lag observations included in the model, also called the lag order.
d: The number of times that the raw observations are differenced, also called the degree of differencing.
q: The size of the moving average window, also called the order of moving average.

ARIMA Algorithm

```
from statsmodels.tsa.arima.model import ARIMA

history = list(train)
order_predictions = []

for i in range(len(test)):
    model = ARIMA(history, order=(2 ,2 ,0)) # defining ARIMA model
    model_fit = model.fit() # fitting model
    y_hat = model_fit.forecast() # predicting 'return'

    order_predictions.append(y_hat[0])
    history.append(test[i])
    print('Prediction: {} of {}'.format(i+1,len(test)), end='\r')
```



Autocorrelation vs Lag



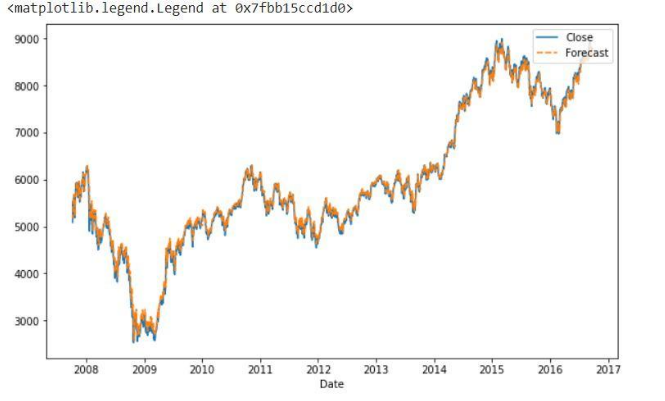
ARIMA Test data graph,
RMSE value: 177.27

Vanila LSTM

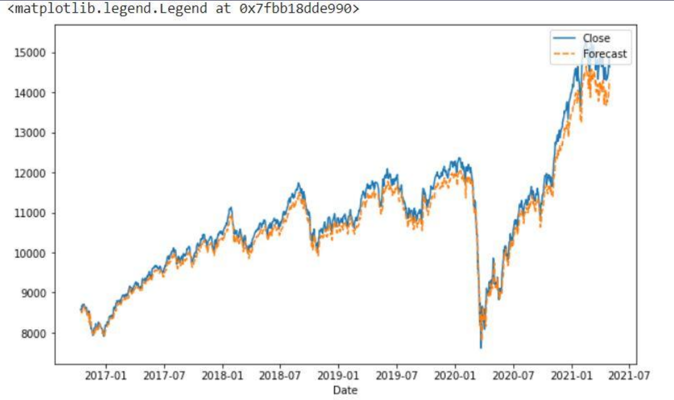
A Vanilla LSTM is an LSTM model that has a single hidden layer of LSTM units, and an output layer used to make a prediction.Shape of the input is key in defining the model. The number of time steps as input is the number we chose when preparing our dataset as an argument. The model is fit using the efficient Adam version of stochastic gradient descent and optimized using the mean squared error, or ‘mse‘ loss function.This model expects the input shape to be three-dimensional and requires reshaping to the single input sample before making the prediction.

LSTM Model Design

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 64)                17664
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense (Dense)                (None, 1)                 65
=================================================================
Total params: 17,729
Trainable params: 17,729
Non-trainable params: 0
_________________________________________________________________
```



LSTM Train data graph, RMSE value: 96.03



LSTM Test data graph, RMSE value: 278.97

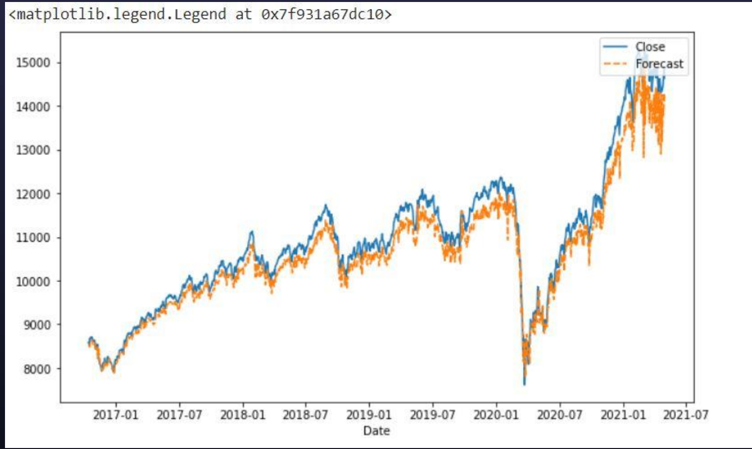
Stacked LSTM

Multiple hidden LSTM layers can be stacked one on top of another in what is referred to as a Stacked LSTM model.
An LSTM layer requires a three-dimensional input and LSTMs by default will produce a two-dimensional output as an interpretation from the end of the sequence.
This model addresses this by having the LSTM output a value for each time step in the input data by setting the return_sequences=True argument on the layer which enables us to have 3D output from hidden LSTM layer as input to the next.

Stacked LSTM model design

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_3 (LSTM)                (None, 14, 64)            17664
_________________________________________________________________
lstm_4 (LSTM)                (None, 32)                12416
_________________________________________________________________
dropout_2 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 33
=================================================================
Total params: 30,113
Trainable params: 30,113
Non-trainable params: 0
_________________________________________________________________
```

Stacked LSTM train graph, RMSE: 99.89

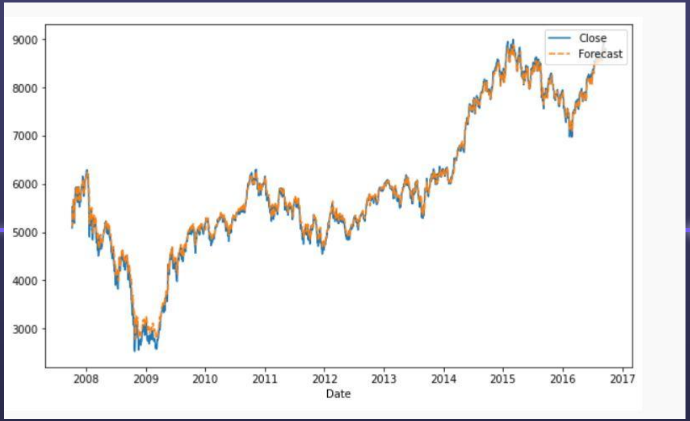


Stacked LSTM test graph, RMSE: 390.2

Bi-directional LSTM

Bidirectional LSTM model allows the LSTM model to learn the input sequence both forward and backwards and concatenate both interpretations. This can be beneficial on some sequence prediction problems.

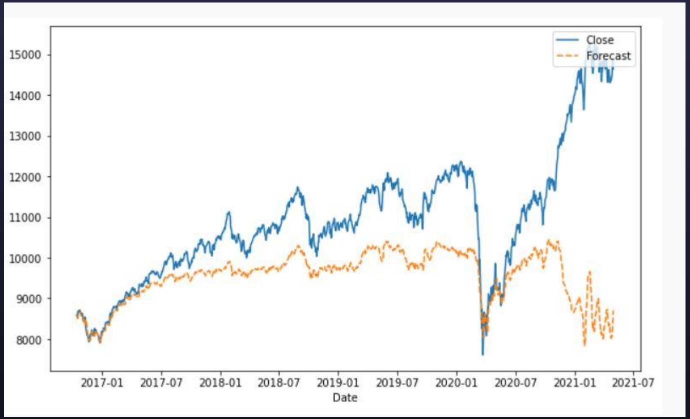
A Bidirectional LSTM for univariate time series forecasting can be implemented by wrapping the first hidden layer in a wrapper layer called Bidirectional.



Bi-Directional LSTM train graph, RMSE: 105.69

Bi-Directional LSTM Model

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
bidirectional (Bidirectional (None, 14, 128)           35328
_________________________________________________________________
dropout (Dropout)            (None, 14, 128)           0
_________________________________________________________________
bidirectional_1 (Bidirection (None, 64)                41216
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense (Dense)                (None, 1)                 65
=================================================================
Total params: 76,609
Trainable params: 76,609
Non-trainable params: 0
```



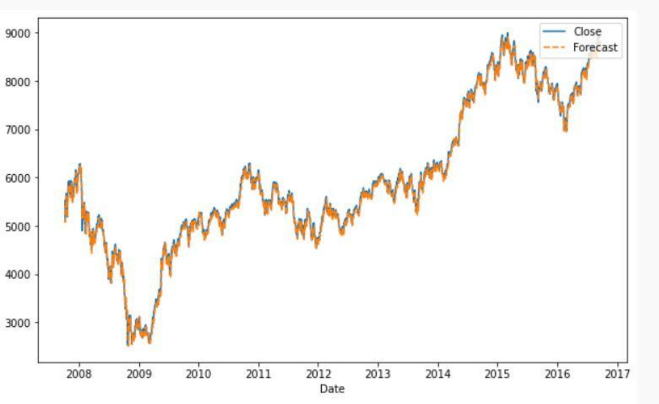
Bi-Directional LSTM test graph, RMSE: 1430.30

CNN

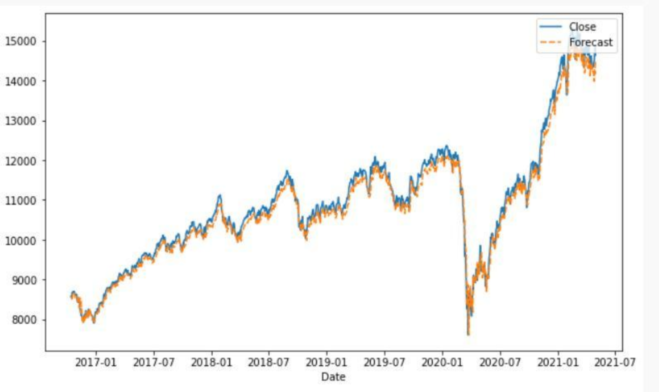
A convolutional neural network, or CNN for short, is a type of neural network developed for working with two-dimensional image data. The CNN can be very effective at automatically extracting and learning features from one-dimensional sequence data such as univariate time series data.

CNN Model

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_6 (Conv1D)            (None, 12, 64)            832
_________________________________________________________________
conv1d_7 (Conv1D)            (None, 10, 64)            12352
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 5, 64)             0
_________________________________________________________________
flatten_1 (Flatten)          (None, 320)               0
_________________________________________________________________
dense_2 (Dense)              (None, 100)               32100
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 101
=================================================================
Total params: 45,385
Trainable params: 45,385
Non-trainable params: 0
_________________________________________________________________
```



CNN train graph, RMSE: 86.29



CNN test graph, RMSE: 210.67

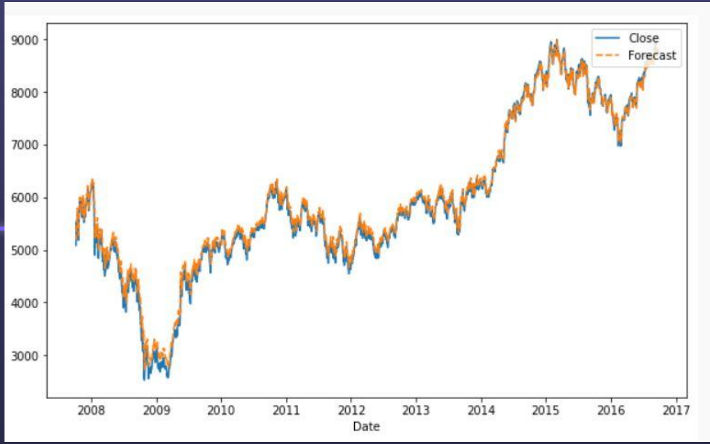
CNN-LSTM

A CNN model can be used in a hybrid model with an LSTM backend where the CNN is used to interpret subsequences of input that together are provided as a sequence to an LSTM model to interpret.

In this model, the first step is to split the input sequences into subsequences that can be processed by the CNN model.Each sample can then be split into two sub-samples, each with two time steps. The CNN can interpret each subsequence of two time steps and provide a time series of interpretations of the subsequences to the LSTM model to process as input.

The CNN model first has a convolutional layer for reading across the subsequence that requires a number of filters and a kernel size to be specified. The number of filters is the number of reads or interpretations of the input sequence. The kernel size is the number of time steps included of each ‘read’ operation of the input sequence.

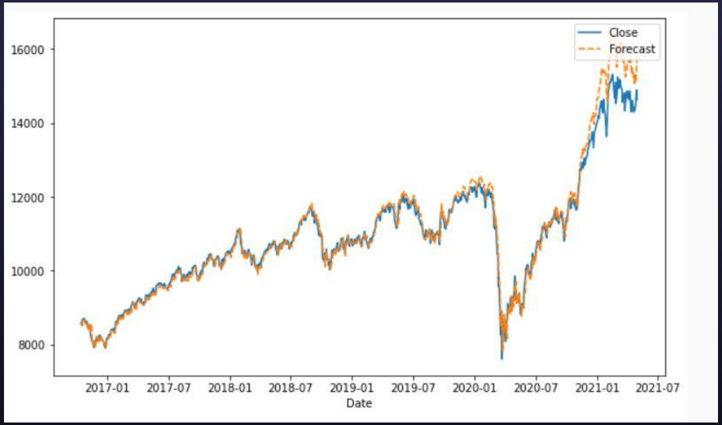
The convolution layer is followed by a max pooling layer that distills the filter maps down to 1/2 of their size that includes the most salient features. These structures are then flattened down to a single one-dimensional vector to be used as a single input time step to the LSTM layer.



CNN-LSTM train graph, RMSE: 120.97

CNN-LSTM Model

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
time_distributed (TimeDistri (None, None, 12, 64)      832
_________________________________________________________________
time_distributed_1 (TimeDist (None, None, 10, 64)      12352
_________________________________________________________________
time_distributed_2 (TimeDist (None, None, 5, 64)       0
_________________________________________________________________
time_distributed_3 (TimeDist (None, None, 320)         0
_________________________________________________________________
lstm (LSTM)                  (None, 64)                98560
_________________________________________________________________
dense (Dense)                (None, 100)               6500
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 101
=================================================================
Total params: 118,345
Trainable params: 118,345
Non-trainable params: 0
```



CNN-LSTM test graph, RMSE: 302.03