UDACITY ADVANCE MACHINE LEARNING NANODEGREE

FUTUTRE STOCK PREDICTION

DEFINITION

1 INTRODUCTION

Stock price prediction is a very popular topic now a days. Earlier statistical methods and stochastic analysis were being used to make stock price prediction but from a while machine learning techniques are being used to predict stock price in recent years. In their research Support Vector Machines for Prediction of Futures Prices in Indian Stock Market, Shom Prasad Das[1] and Sudarsan Padhy [1] discuss Back Propagation Technique and Support Vector Machine Technique to predict futures prices traded in Indian stock market. Then advancement came in the form of recurrent neural nets as depicted in Comparative Study of Stock Trend Prediction Using Time Delay, Recurrent and Probabilistic Neural Networks by Emad W. Saad [2], Then further improvement was observed using LSTM (long-short term memory) approach by Murtaza Roondiwala[3] , Harshal Patel[3], Shraddha Varma[3] to forecast stock index prices. I would like to take this project to explore the concepts and working of LSTM.

2 PROBLEM STATEMENT

The purpose of the project is to create a neural network model that can predict stock price by using historical information as a time-series data. The task is to build a stock price predictor that takes daily trading data over a time of certain years as input, and outputs projected estimates for given query dates. The inputs will contain multiple metrics, such as opening price (Open), highest price the stock traded at (High), how many stocks were traded (Volume) and closing price adjusted for stock splits and dividends (Adjusted Close); my system will predict the Adjusted Close price.

TASKS TO BE DONE:

- Explore stock prices
- Implement the basic benchmark model using regression
- Implement LSTM using keras
- Compare the results

3 EVALUATION METRICS

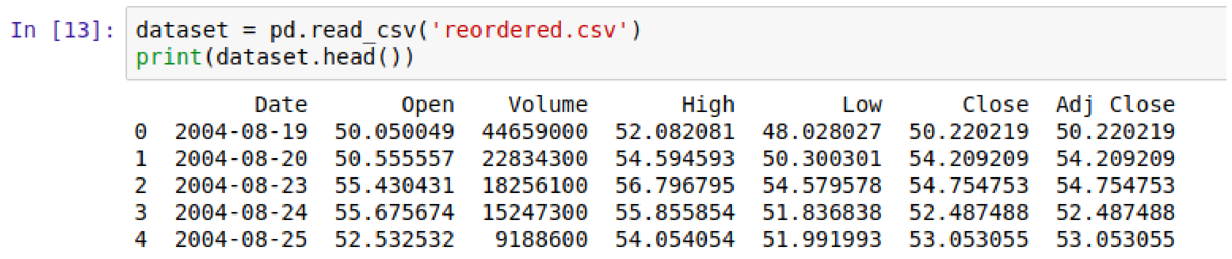
For this problem I feel that the metrics that would be appropriate are R-square and root-mean-squared-error. R-square can provide how much variation in the dependent variable can be explained by the variation in the independent variables. Root-mean-squared-error can provide what is the average deviation of the prediction from the true value, and it can be compared with the mean of the true value to see whether the deviation is large or small. So these two together will serve the purpose.

ANALYSIS

1 Data Exploration

There are 505 companies in the S&P 500. The list of companies and symbols can be find here. My data will include all these stocks' Open, High, Low, Close, Adj Close price and trading Volume. The data will start from 2001-05-13 and end at 2017-5-13.'

A glimpse of a typical stock's data is shown below:

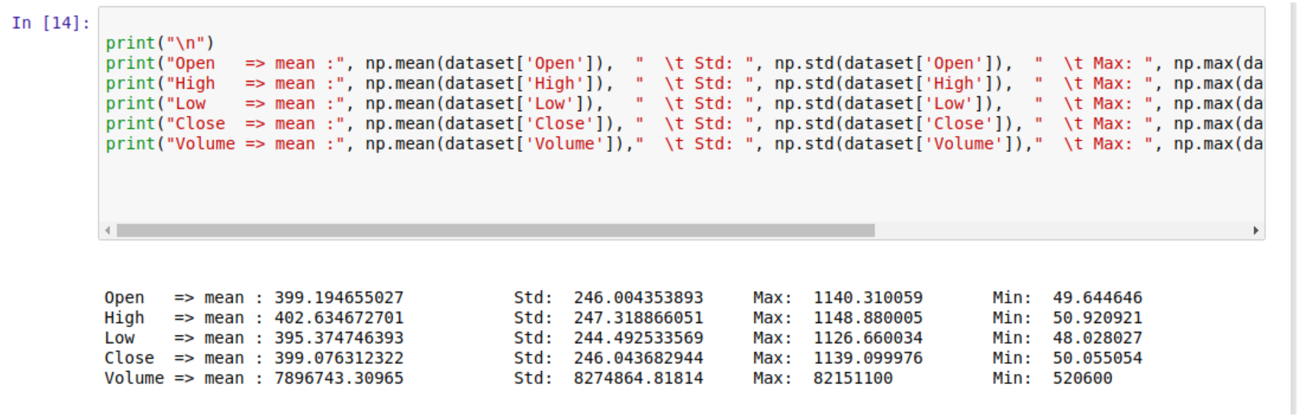


```
In [13]: dataset = pd.read_csv('reordered.csv')
         print(dataset.head())
```

```
         Date       Open    Volume       High        Low      Close  Adj Close
0  2004-08-19  50.050049  44659000  52.082081  48.028027  50.220219  50.220219
1  2004-08-20  50.555557  22834300  54.594593  50.300301  54.209209  54.209209
2  2004-08-23  55.430431  18256100  56.796795  54.579578  54.754753  54.754753
3  2004-08-24  55.675674  15247300  55.855854  51.836838  52.487488  52.487488
4  2004-08-25  52.532532   9188600  54.054054  51.991993  53.053055  53.053055
```

Note: I did not observe any abnormality in datasets, i.e, no feature is empty and does not cont ains any incorrect value as negative values.

The mean, standard deviation, maximum and minimum of the data are depicted below:



```
In [14]:
print("\n")
print("Open   => mean :", np.mean(dataset['Open']),   "  \t Std: ", np.std(dataset['Open']),   "  \t Max: ", np.max(da
print("High   => mean :", np.mean(dataset['High']),   "  \t Std: ", np.std(dataset['High']),   "  \t Max: ", np.max(da
print("Low    => mean :", np.mean(dataset['Low']),    "  \t Std: ", np.std(dataset['Low']),    "  \t Max: ", np.max(da
print("Close  => mean :", np.mean(dataset['Close']),  "  \t Std: ", np.std(dataset['Close']),  "  \t Max: ", np.max(da
print("Volume => mean :", np.mean(dataset['Volume']),"  \t Std: ", np.std(dataset['Volume']),"  \t Max: ", np.max(da
```

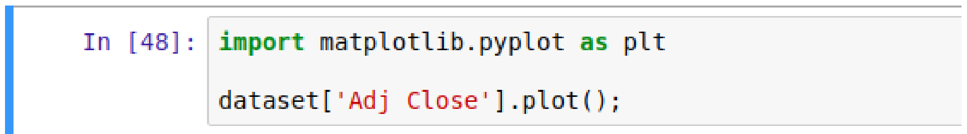
```
Open   => mean : 399.194655027         Std:  246.004353893     Max:  1140.310059       Min:  49.644646
High   => mean : 402.634672701         Std:  247.318866051     Max:  1148.880005       Min:  50.920921
Low    => mean : 395.374746393         Std:  244.492533569     Max:  1126.660034       Min:  48.028027
Close  => mean : 399.076312322         Std:  246.043682944     Max:  1139.099976       Min:  50.055054
Volume => mean : 7896743.30965         Std:  8274864.81814     Max:  82151100          Min:  520600
```

We can infer from this dataset that date, high and low values are not important features of th e data when we consider our benchmark model. But all the features except date highly contribute towards the accuracy of LSTM model.

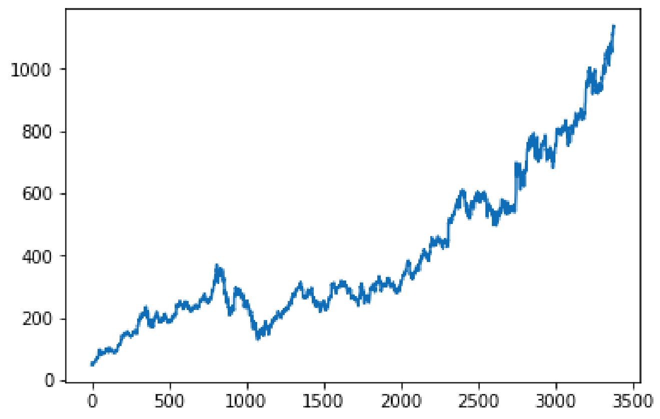
2 Data Visualization

We want to predict the values of adjusted closing prices and trend of the data is as shown below:

DATA EXPLORATION



```
In [48]: import matplotlib.pyplot as plt

         dataset['Adj Close'].plot();
```



3 Algorithms and Techniques

The goal of this project was to study time series data i.e. when the data has a certain kind of sequence or dependence on the previous value and explore various possible ways to predict with a decent accuracy. I discovered that LSTM which are a modification of the RNN network have been used on such sequential data and give good results.[3] The LSTMS are networks which contain memory cells which is controlled with the help of gates. A much more detailed explanation is as given below:

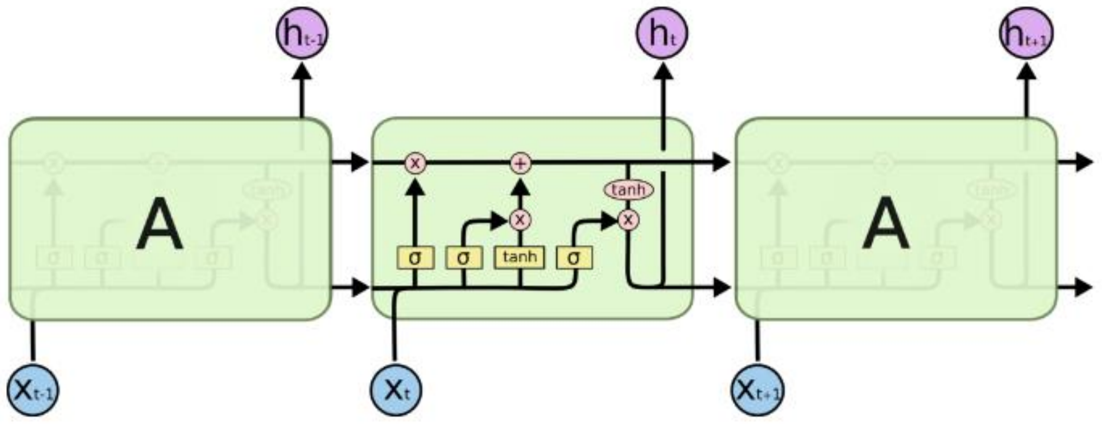
- **Recurrent Neural Networks**

  The basic idea behind RNNs is to make use of sequential information. In a traditional neural network we assume that all inputs are independent of each other but If you want to predict the next word in a sentence you better know which words came before it, there comes the concept of RNNs, also called recurrent because they perform the same task for every element of a sequence, with the output being depended on the previous computations. The interesting feature by which this is accomplished is that they have a "memory" which captures information about what has been calculated so far. But this has a few drawbacks like exploring gradient and vanishing gradient problem.

- **Long Short Term Memory**

  In order to solve the drawback of RNN there came LSTMs which don't have different architecture from RNNs, but use a different function to compute the hidden state. The memory in LSTMs are called cells that take as input the previous state and current input. Internally these cells decide what to keep in (and what to erase from) memory. They then combine the previous state, the current memory, and the input. It turns out that these types of units are very efficient at capturing long-term dependencies hence it will be suitable for my problem.

  The basic LSTM unit is composed of a **cell**, an **input gate**, an **output gate** and a **forget gate** as shown below:



FORGET GATE:

A forget gate is responsible for removing information from the cell state. The information that is no longer required for the LSTM to understand things or the information that is of less importance is removed.

INPUT GATE:

The input gate is responsible for the addition of information to the cell state. This addition of information is basically three-steps :

- Regulating what values need to be added to the cell state by involving a sigmoid function.
- Creating a vector containing all possible values that can be added
- Multiplying the value of the regulatory filter to the created vector and then adding this useful information to the cell state.

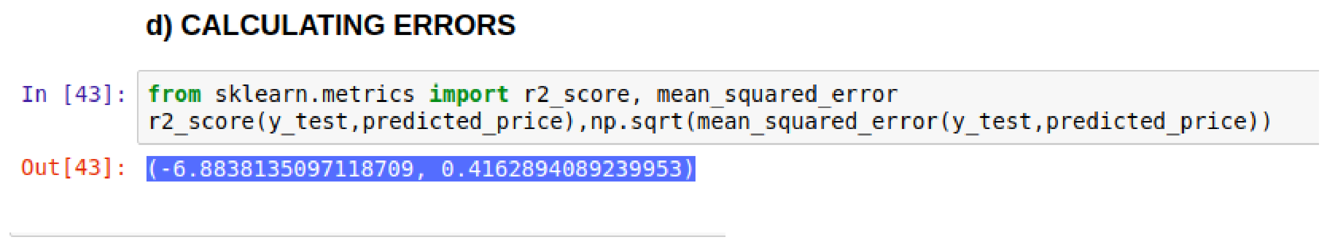
OUTPUT GATE:

This job of selecting useful information from the current cell state and showing it out as an output is done via the output gate.

BENCHMARK MODEL

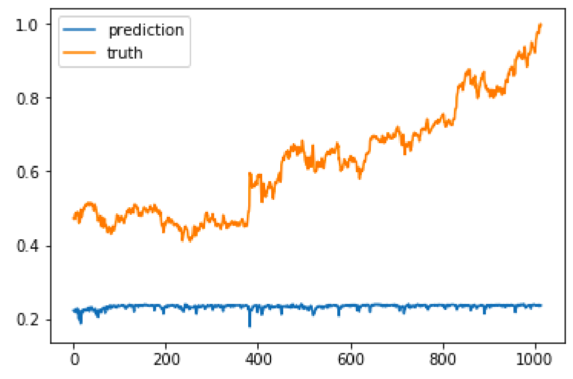
The benchmark model for this project would be using linear regressions since my main goal to compare and contrast the accuracies of the various machine learning models. This linear regression model would be based on one of the course by Udacity on Trading and will serve as a basis of comparison for the error rates.

The results of the benchmark model are as follows:



d) CALCULATING ERRORS

```
In [43]: from sklearn.metrics import r2_score, mean_squared_error
         r2_score(y_test,predicted_price),np.sqrt(mean_squared_error(y_test,predicted_price))

Out[43]: (-6.8838135097118709, 0.4162894089239953)
```



METHODLOGY

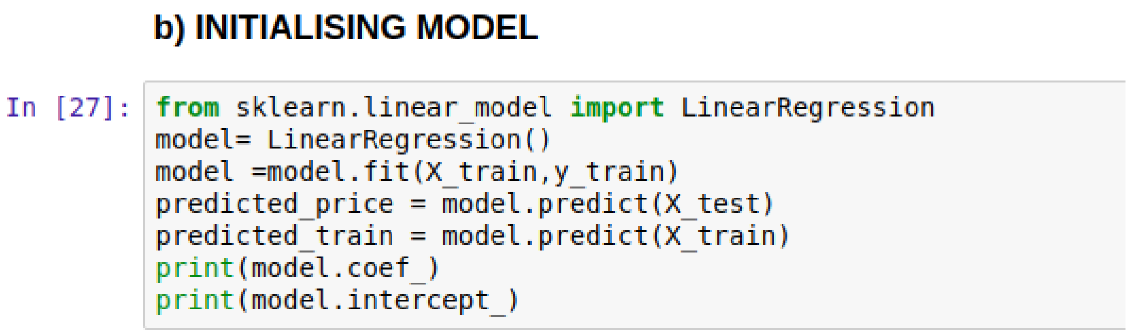
1 Data Pre-Processing

- The first step of the project would be to acquire the data set in the form of the .csv file from yahoo finance.
- Remove unimportant features (date) from the acquired data.
- We have prices of different stocks, they are in different scales. And also prices and volumes are in totally different scales. In order to make neural network converge faster, we should scale our inputs. Then the next step involves the normalizing of data which is a requirement for the LSTM networks which will be done using the function MinMaxScaler of the sklearn library.
- Split the dataset into the training (70%) and test (30%) datasets for linear regression mo del. The split was of following shape :
- Split the dataset into the training (70%) and test (3%) datasets for LSTM model. The split was of following shape :
- For the LSTM model I have tried sequence generation ie the concept of window to make a improved model. I set a rolling window of length 5 days, move this window along the time series, and record the data in each step as one row of input sequence.

2 IMPLEMENTATION

The benchmark model is implemented using the library sklearn as:

1) The First step is to import the libraries needed so I imported sklearn library and from that library I needed the linear model so the first line of the code is importing Linear Regression from sklearn library in python language.
2) Second step is we create an object for the model.
3) Third step is using the above model we train the data by using fit function like model.fit() and to this function the input are feature vector (X_train) and label vector(y_train).
4) The next step is that we predict the values for the part of dataset which is not seen by the model while training so for that we use the function predict like model.predict() And the input to this function is the feature vector of unseen data.
5) Then since linear regression basically models the data using a line we can get its slope and intercept and for that the functions are:
   Slope: model.coef_ and Intercept : model.intercept_

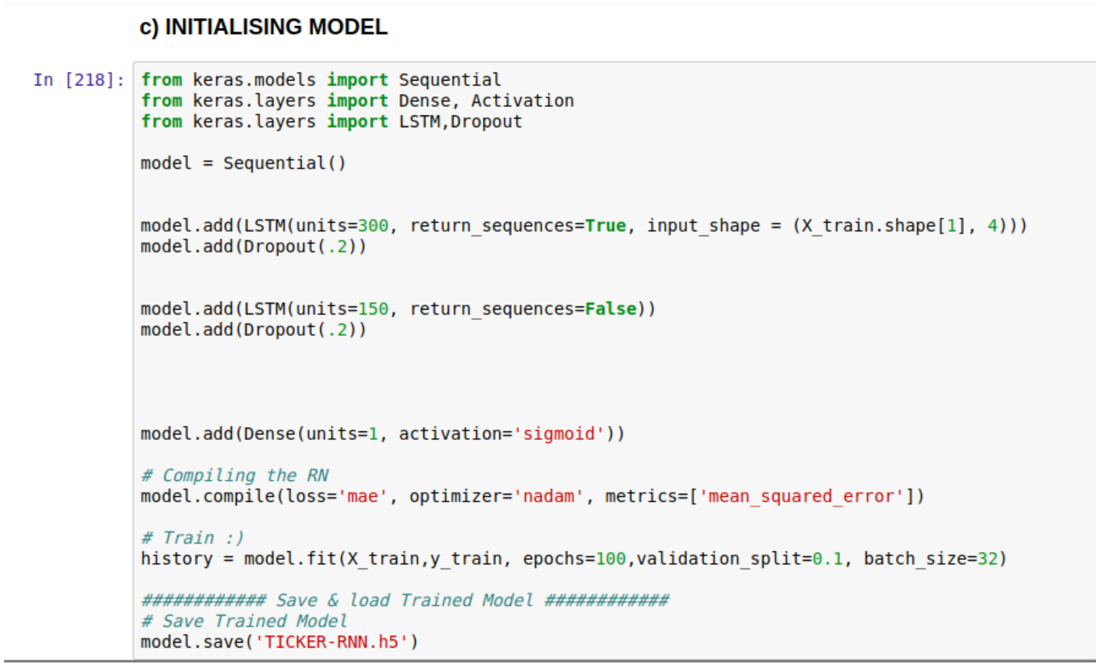


b) INITIALISING MODEL

```
In [27]: from sklearn.linear_model import LinearRegression
         model= LinearRegression()
         model =model.fit(X_train,y_train)
         predicted_price = model.predict(X_test)
         predicted_train = model.predict(X_train)
         print(model.coef_)
         print(model.intercept_)
```

The LSTM model's specifications are as follows:

1) This model will be implemented using Keras module. So we know that this is a library in python and to make any model in keras we need to import the type of model i.e. Sequential in our case hence we write like:
   From keras.models import Sequential
   Then we need to import the type of layers we need to add in out model and for that we write:
   From keras.layers import Dense, activation, Dropout ,LSTM
2) Second is to create an object for the model and we do so by calling Sequential function.
3) Then we add layers to the model using the function add
4) In this network first a LSTM layer is added which takes the 3 dimensional array as input and has dimension 300 i.e the number of neurons. The concept of return_sequences Whether to return the last output in the output sequence, or the full sequence, so we set this parameter as true for all the layers except the last because we want full output sequence.
5) Then we add a dropout layer with a value of 0.2 as a parameter to the function which means that the probability of dropping a neuron is 0.2.
6) Then again a LSTM and a dropout layer have been added.
7) The final layer is the output dense layer containing a single neuron, with a sigmoid function
8) Then we compile the function using a loss function parameter, the optimizer and the metrics we want to use to check the model's efficiency.
9) At last we call the function fit to train the model using the dataset and give input of number of epochs, batch size and validation split ie we define on how much part of data should the model validate the results while training.


c) INITIALISING MODEL

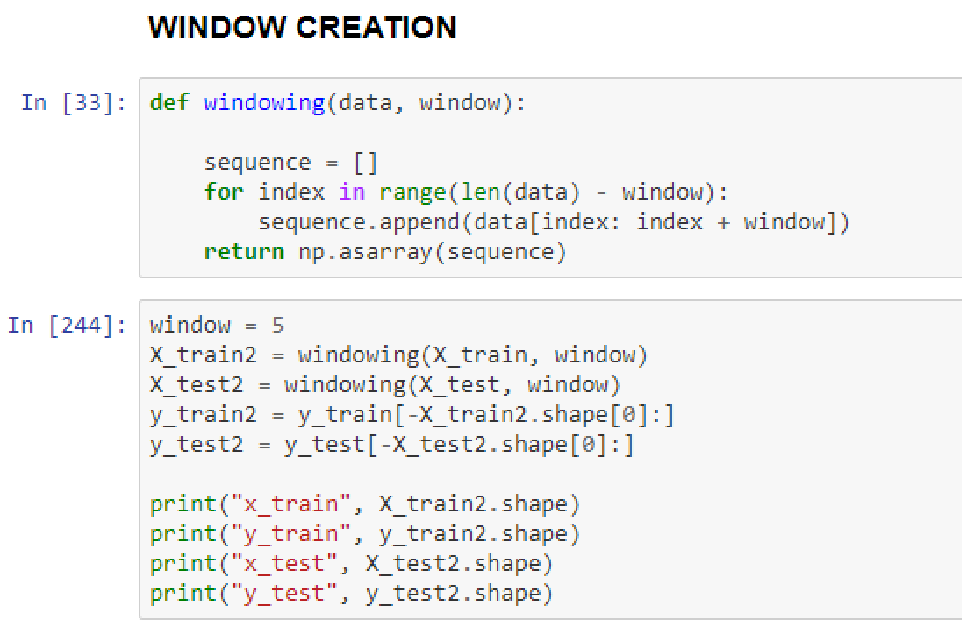
```
In [218]: from keras.models import Sequential
          from keras.layers import Dense, Activation
          from keras.layers import LSTM,Dropout

          model = Sequential()


          model.add(LSTM(units=300, return_sequences=True, input_shape = (X_train.shape[1], 4)))
          model.add(Dropout(.2))


          model.add(LSTM(units=150, return_sequences=False))
          model.add(Dropout(.2))



          model.add(Dense(units=1, activation='sigmoid'))

          # Compiling the RN
          model.compile(loss='mae', optimizer='nadam', metrics=['mean_squared_error'])

          # Train :)
          history = model.fit(X_train,y_train, epochs=100,validation_split=0.1, batch_size=32)

          ############ Save & load Trained Model ############
          # Save Trained Model
          model.save('TICKER-RNN.h5')
```

The loss function is chosen to be mean absolute error, and the optimizer is chosen to be Nesterov Adam, which is Adam RMSprop with Nesterov momentum and a track of the validation loss is kept so that we are able to prevent the model from over-fitting.

The only problems I faced while implementing this was of setting the input dimensions for the two models, It took we a lot of time to understand the difference in data dimensions and then how to implement it and then while studying I read about numpy library and its reshape function and then after some while I got it.

REFINEMENT

A new model has also been created using windowing:

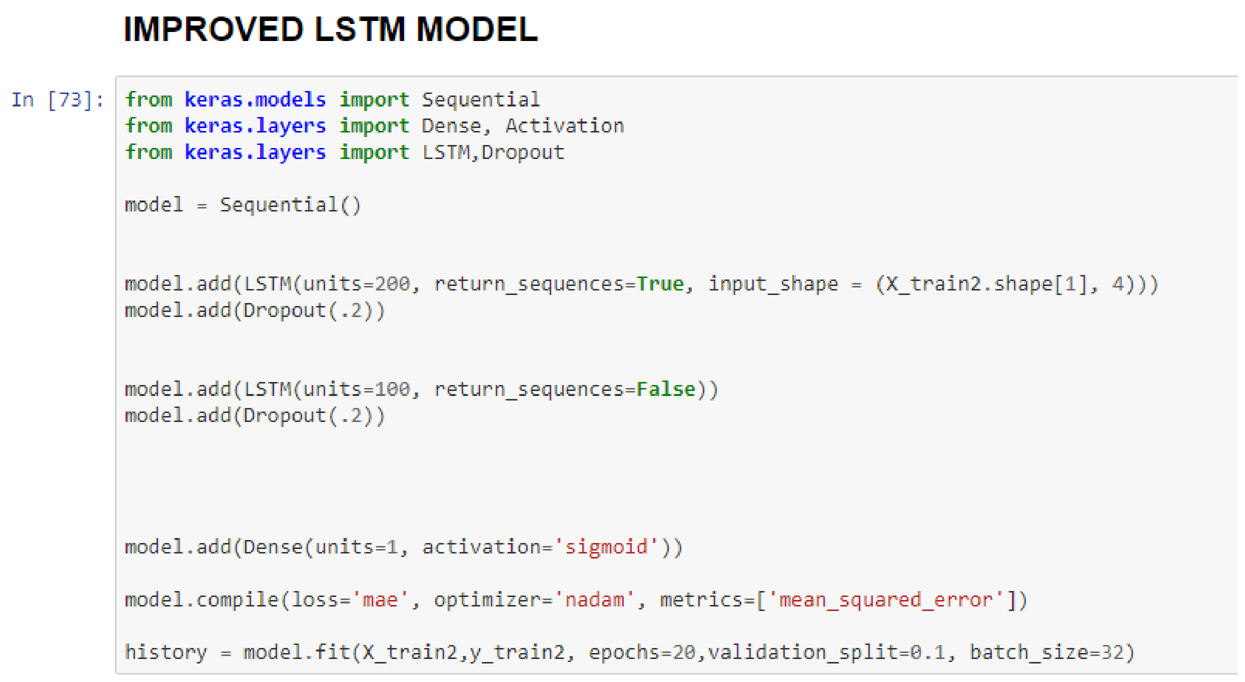


WINDOW CREATION

```
In [33]:  def windowing(data, window):

              sequence = []
              for index in range(len(data) - window):
                  sequence.append(data[index: index + window])
              return np.asarray(sequence)
```

```
In [244]: window = 5
          X_train2 = windowing(X_train, window)
          X_test2 = windowing(X_test, window)
          y_train2 = y_train[-X_train2.shape[0]:]
          y_test2 = y_test[-X_test2.shape[0]:]

          print("x_train", X_train2.shape)
          print("y_train", y_train2.shape)
          print("x_test", X_test2.shape)
          print("y_test", y_test2.shape)
```

For implementing the concept of window I have created a function named windowing which takes the window length as input and then iterates through the whole data and for every data unit append the data of window ie coontaining the next datapoints/units into one single array.



IMPROVED LSTM MODEL

```
In [73]: from keras.models import Sequential
         from keras.layers import Dense, Activation
         from keras.layers import LSTM,Dropout

         model = Sequential()


         model.add(LSTM(units=200, return_sequences=True, input_shape = (X_train2.shape[1], 4)))
         model.add(Dropout(.2))


         model.add(LSTM(units=100, return_sequences=False))
         model.add(Dropout(.2))




         model.add(Dense(units=1, activation='sigmoid'))

         model.compile(loss='mae', optimizer='nadam', metrics=['mean_squared_error'])

         history = model.fit(X_train2,y_train2, epochs=20,validation_split=0.1, batch_size=32)
```

There are a few hyper-parameters in the models which can be adjusted :

1) **SEQUENCE LENGTH:**
   I have tried sequence length of 5, 10, 20, 50(it is approximately the number of trading days in a month) and it turns out the result of 5 days was the best.
2) **NUMBER OF LAYERS.**
   I tried network structure of 3 LSTM layers with hidden units of 200-100-100 and 2 LSTM layers described above, and it turns out the result with 2 layers was better than 3 layers and the best score was observed at 300-150
3) **THE LOSS FUNCTION.**
   I tried using mean absolute error and mean squared error as loss function, and it turns out that the model with mean squared error loss function gives prediction of a straight line, while the model with mean absolute error gives a better result
4) **THE OPTIMIZER**
   I tried using RMSprop, Adam and Nesterov Adam, and the result are pretty similar. Since in the research paper I studied it is mentioned that Nestrov Adam can yield better convergence, therefore I took the Nesterov Adam .
5) **THE NUMBER OF EPOCHS**.
   According to the learning curve I will show in the next section, the loss function converges for both training and testing set converge after 100 epochs, so I tried using epochs of 300 and 200, and it turned out the results were pretty bad.

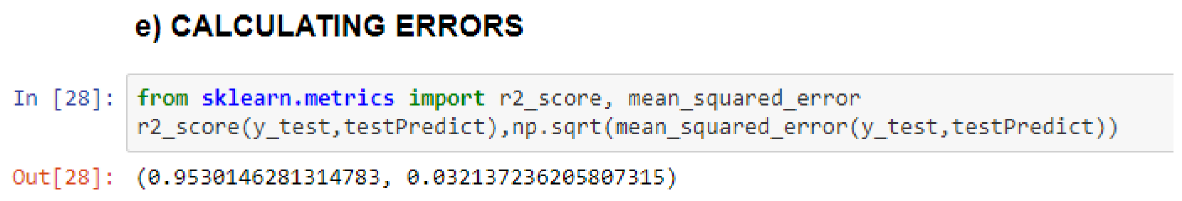
RESULTS

Model Evaluation and Validation

Final parameters of the model are

1) NORMAL LSTM MODEL:
   The number of neurons in layers are 300 -150
   Batch size =32
   Number of epochs = 100

Results are:

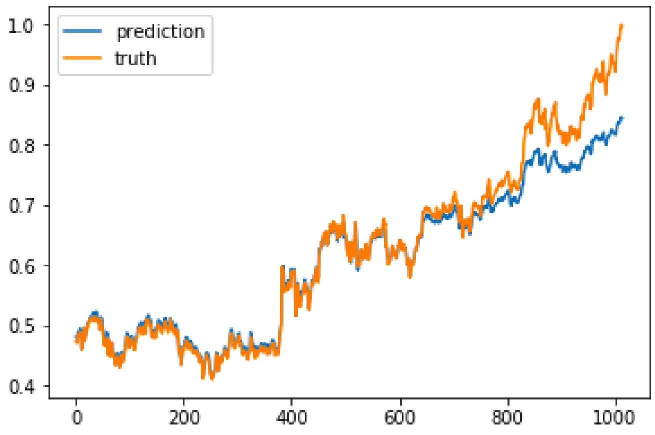


e) CALCULATING ERRORS

```
In [28]: from sklearn.metrics import r2_score, mean_squared_error
         r2_score(y_test,testPredict),np.sqrt(mean_squared_error(y_test,testPredict))

Out[28]: (0.9530146281314783, 0.032137236205807315)
```

d) DISPLAYING RESULT

In [226]:
```
plt.plot(testPredict, label='prediction')
plt.plot(y_test, label='truth')
plt.legend()
plt.show()
```



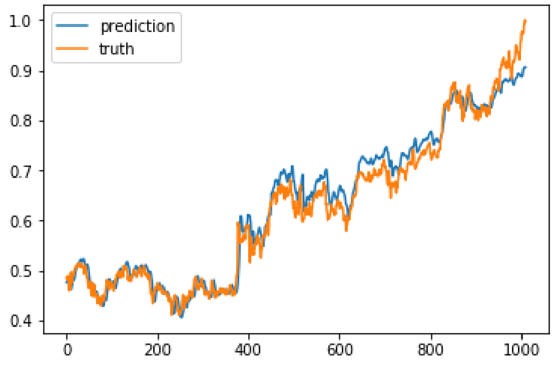
2) IMPROVED LSTM MODEL:
   The number of neurons in layers are 200 -100
   Batch size =32
   Number of epochs = 20

CALCULATING ERRORS

In [37]:
```
from sklearn.metrics import r2_score, mean_squared_error
r2_score(y_test2,testPredict),np.sqrt(mean_squared_error(y_test2,testPredict))
```

Out[37]:
```
(0.96593405620713035, 0.027365928389181816)
```

In [68]:
```
plt.plot(testPredict, label='prediction')
plt.plot(y_test2, label='truth')
plt.legend()
plt.show()
```



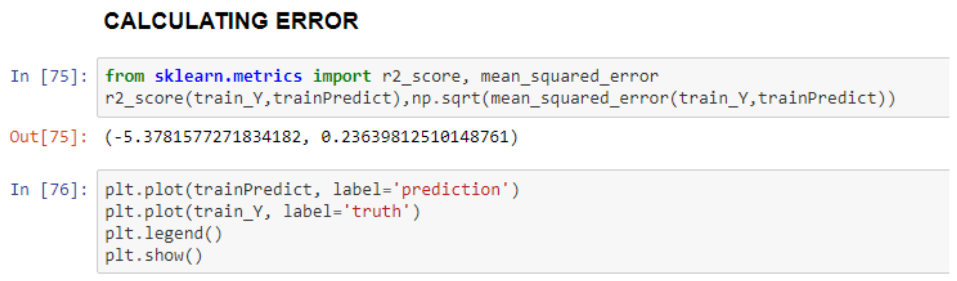
MODEL ROBUSTNESS

In order to show that my model is robust I would like to give two arguments.

First that for my improved LSTM model I am getting a R2 score of 0.96 which is a quite high value and shows that model predicts the correct values upto 96 %

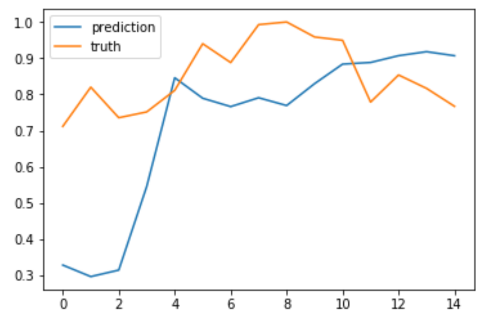
The second demonstration I would like to give is that I tried the same model on the amazon data and the results which I received are very poor and are depicted below**:**

We can see the r2 score has dropped from 0.96 to -5.3 which exactly proves my point.



CALCULATING ERROR

```
In [75]: from sklearn.metrics import r2_score, mean_squared_error
         r2_score(train_Y,trainPredict),np.sqrt(mean_squared_error(train_Y,trainPredict))

Out[75]: (-5.3781577271834182, 0.23639812510148761)

In [76]: plt.plot(trainPredict, label='prediction')
         plt.plot(train_Y, label='truth')
         plt.legend()
         plt.show()
```



JUSTIFICATION

The final results when compared to the benchmark model show significant improvement.

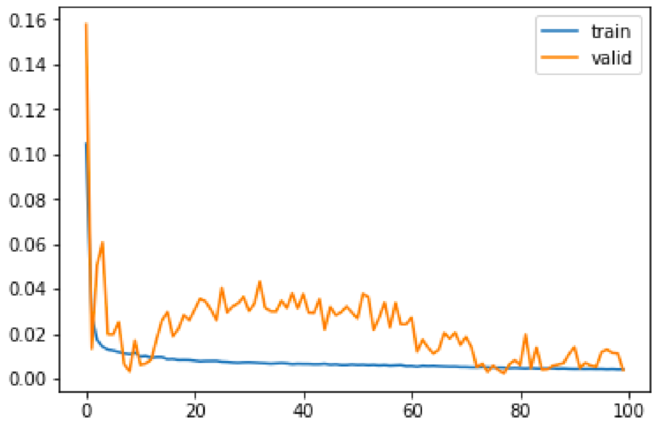
The r2 score for the benchmark model was -6.8 and after applying the LSTM model it improved to 0.95 i.e really close to the maximum value 1 and through improved LSTM model it was observed to be 0.96 which was even better. So here we see a lot of increase and can say that the LSTM models are performing way better than linear regression since R2 score is an indicator of how much variation in the dependent variable can be explained by the variation in the independent variables.

CONCLUSION

FREE-FORM VISUALIZATION

A very important topic for neural networks is its convergence of loss function. Not only the convergence will show whether the model is successfully trained, it will also tells you whether the model is under-fitting or over-fitting. The figure below shows the learning curve of the training process of the model. We can see from the curve that both the loss for training and validation set is decreasing as the number of epoch increases, and converges to almost zero. This indicates the model is successfully training, at least is converging into a local minimum. The validation loss is decreasing and is less volatile as the number of epoch increase, which indicates there is no over-fitting or under-fitting problem in the training process.

```
In [219]: plt.plot(history.history['loss'], label='train')
          plt.plot(history.history['val_loss'], label='valid')
          plt.legend()
          plt.show()
```



REFLECTION

To recap, the process undertaken in this project

- Prepare Dataset
  - Incorporate data from yahoo finance
  - Process the requested data according to the requirements of the models
  - Develop function for normalizing data
  - performing 80/20 split on training and test data across all models
- Develop Benchmark Model
  - Setting up basic Linear Regression model with Sklearn library

    - Calibrate parameters
- Develop Basic LSTM Model
    - Set up basic LSTM model with Keras and improving it to increase the accuracy.

- Analyze and understanding the results.

I started this project in order to learn a completely new algorithm, i.e, Long Short Term Memory and also to explore a real time series data sets.

The major problem i faced during the implementation of project was exploring the data and preprocessing it and understanding how to appropriately shape and split the data for the two models: regression and LSTM.

IMPROVEMENT

AS of now I feel this model could be further improved by using deep LSTM model or Stacked LSTM model and moreover currently GRU as a modification of recurrent neiral networks has also come up that can also be applied to increase the efficiency more.

REFERENCES

[1] Das, Shom Prasad, and Sudarsan Padhy Support vector machines for prediction of futures prices in Indian stock market. International Journal of Computer Applications 41.3 (2012).

[2] Emad W. Saad, Student Member, IEEE, Danil V. Prokhorov, Member, IEEE, and Donald C. Wunsch, II, Senior Member, IEEE Comparative Study of Stock Trend Prediction Using Time Delay, Recurrent and Probabilistic Neural Networks, IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 9, NO. 6, NOVEMBER 1998

[3] Murtaza Roondiwala, Harshal Patel, Shraddha Varma Predicting Stock Prices Using LSTM, International Journal of Science and Research (IJSR),2017