PROJECT REPORT

ON

Application of LSTM for Stock Price prediction

BY

UMAR AYOUB HAJAM

(PROJECT LEAD)

Executive Summery

Stock market prediction is an attempt of determining the future value of a stock traded on a stock exchange. This project report attempts to provide an optimal Long Short Term Memory model for the prediction of stock prices from the experiments it is found that Stacked LSTM with Hyperbolic Tangent and Rectified Linear Unit activation function is most successful in stock price prediction. In addition the experiments have suggested suitable values of the look-back period that could be used with LSTM.

<u>Note: I have performed many experiments this report is for the stock price prediction with 1 minute interval</u>

GITHUB LINK: https://github.com/1UmAr1/LSMT-Stock-Price-Prediction/tree/main/Deploy%20GRU%201%20MIN%20NFTY

Machine and ide details

PyCharm 2021.1 (Community Edition)

Build #PC-211.6693.115, built on April 6, 2021

Runtime version: 11.0.10+9-b1341.35 amd64

VM: Dynamic Code Evolution 64-Bit Server VM by JetBrains s.r.o.

Windows 10 10.0

GC: ParNew, ConcurrentMarkSweep

Memory: 9933M

Cores: 8

Modules/Libraries

The modules/libraries used in model generation are

1. Sklearn
2. Pandas
3. Matplotlib
4. Numpy
5. Keras

Importing the required Libraries

```
import keras.models
import pandas as pd
import numpy as np
import yfinance as yf
# Get the data
# create a model
from keras.models import Sequential
from keras.layers import Dense, SimpleRNN, LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
# data = yf.download(tickers="MSFT", interval="1m")
data = pd.read_csv("DataFrame.csv")
# Print the data
print(data.describe())
print(data.head())
print(data.tail())
print(data.columns)

print(data["close"])
```

OUTPUT

```
2021-05-05 14:24:12.576587: I tensorflow/stream_executor/cuda/cuda_blas.cc:2
               open          Date          high           low         close
count  22805.000000  2.280500e+04  22805.000000  22805.000000  22805.000000
mean   14703.322396  2.021022e+07  14707.486481  14699.102201  14703.260816
std      391.377498  8.285616e+01    391.265231    391.551643    391.366248
min    13604.750000  2.021010e+07  13614.400000  13596.750000  13602.800000
25%    14462.250000  2.021012e+07  14465.950000  14458.050000  14461.950000
50%    14739.500000  2.021022e+07  14743.000000  14735.850000  14739.450000
75%    15037.000000  2.021031e+07  15041.250000  15032.500000  15036.850000
max    15429.500000  2.021033e+07  15431.750000  15427.900000  15430.200000
       open   Time      Date      high       low     close
0  13997.90  09:16  20210101  14020.85  13991.35  14013.15
1  14014.85  09:17  20210101  14018.55  14008.15  14009.05
2  14008.05  09:18  20210101  14013.10  14005.05  14012.70
3  14013.65  09:19  20210101  14019.10  14013.65  14016.20
4  14015.45  09:20  20210101  14017.80  14011.95  14015.45
           open   Time      Date      high       low     close
In[3]:
```

Creating a function to split the dataset into training and testing sets

```
def ts_train_test(all_data, time_steps, for_periods):
    # create training and test set
    ts_train = all_data[:18000].iloc[:, 0:1].values
    ts_test = all_data[19000:].iloc[:, 0:1].values
    ts_train_len = len(ts_train)
    ts_test_len = len(ts_test)

    # create training data of s samples and t time steps
    X_train = []
    y_train = []
    y_train_stacked = []
    for i in range(time_steps, ts_train_len - 1):
        X_train.append(ts_train[i - time_steps:i, 0])
        y_train.append(ts_train[i:i + for_periods, 0])
    X_train, y_train = np.array(X_train), np.array(y_train)

    # Reshaping X_train for efficient modelling
    X_train = np.reshape(X_train, (X_train.shape[0],
                         X_train.shape[1], 1))

    # Preparing to create X_test
    inputs = pd.concat((all_data["close"][:18000],
```

```
all_data["close"][19000:]),
axis=0).values
inputs = inputs[len(inputs) - len(ts_test) - time_steps:]
inputs = inputs.reshape(-1, 1)

X_test = []
for i in range(time_steps, ts_test_len + time_steps -
for_periods):
X_test.append(inputs[i - time_steps:i, 0])

X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0],
X_test.shape[1], 1))

return X_train, y_train, X_test
```

Applying the function to create train and test splits

```
X_train, y_train, X_test = ts_train_test(data, 5, 2)
print(X_train.shape[0], X_train.shape[1])
print(len(X_train))
print(len(X_test))
```

Converting the 3-Dimensional shape of X_train and X_test to a dataframe so we can visualize it

```
# Convert the 3-D shape of X_train to a data frame so we can
see:
X_train_see = pd.DataFrame(np.reshape(X_train,
(X_train.shape[0], X_train.shape[1])))
y_train_see = pd.DataFrame(y_train)
pd.concat([X_train_see, y_train_see], axis=1)

# Convert the 3-D shape of X_test to a data frame so we can
see:
X_test_see = pd.DataFrame(np.reshape(X_test,
(X_test.shape[0],X_test.shape[1])))
pd.DataFrame(X_test_see)
```

Creating a function to normalize the datasets

```
def ts_train_test_normalize(all_data, time_steps,
for_periods):
# create training and test set
```

```
ts_train = all_data[:18000].iloc[:, 0:1].values
ts_test = all_data[19000:].iloc[:, 0:1].values
ts_train_len = len(ts_train)
ts_test_len = len(ts_test)

# scale the data
sc = MinMaxScaler(feature_range=(0, 1))
ts_train_scaled = sc.fit_transform(ts_train)

# create training data of s samples and t time steps
X_train = []
y_train = []
y_train_stacked = []
for i in range(time_steps, ts_train_len - 1):
X_train.append(ts_train_scaled[i - time_steps:i, 0])
y_train.append(ts_train_scaled[i:i + for_periods, 0])
X_train, y_train = np.array(X_train), np.array(y_train)

# Reshaping X_train for efficient modelling
X_train = np.reshape(X_train, (X_train.shape[0],
X_train.shape[1], 1))

inputs = pd.concat((all_data["close"][:18000],
all_data["close"][19000:]),
axis=0).values
inputs = inputs[len(inputs) - len(ts_test) - time_steps:]
inputs = inputs.reshape(-1, 1)
inputs = sc.transform(inputs)

# Preparing X_test
X_test = []
for i in range(time_steps, ts_test_len + time_steps -
for_periods):
X_test.append(inputs[i - time_steps:i, 0])

X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0],
X_test.shape[1], 1))

return X_train, y_train, X_test, sc
```

Normalizing the Test and Training datasets

```
X_train, y_train, X_test, sc = ts_train_test_normalize(data,
5, 2)
```

Creating our LSTM model

```
def LSTM_model(X_train, y_train, X_test, sc):

    # create a model
    from keras.models import Sequential
    from keras.layers import Dense, SimpleRNN

    model = Sequential()
    model.add(LSTM(64, return_sequences=True,
    activation="tanh"))
model.add(LSTM(64, return_sequences=True, activation="tanh"))

model.add(LSTM(64, return_sequences=True, activation="relu"))

    model.add(SimpleRNN(32))
    model.add(Dense(2))  # The time step of the output

    model.compile(optimizer='rmsprop',
    loss='mean_squared_error')

    # fit the LSTM model
    model.fit(X_train, y_train, epochs=100, batch_size=150,
    verbose=True)

    # Finalizing predictions
    predictions = model.predict(X_test)
    from sklearn.preprocessing import MinMaxScaler
    predictions = sc.inverse_transform(predictions)

    return model, predictions
```

Plotting the graph

```
def actual_pred_plot(preds):

    actual_pred = pd.DataFrame(columns=['close',
    'prediction'])
    actual_pred['close'] = data.loc[19000:,
    'close'][0:len(preds)]
    actual_pred['prediction'] = preds[:, 0]

    from keras.metrics import MeanSquaredError
    m = MeanSquaredError()
    m.update_state(np.array(actual_pred['close']),
    np.array(actual_pred['prediction']))

    return (m.result().numpy(), actual_pred.plot())


actual_pred_plot(predictions)
```

Training the model

```
model, predictions_2 = LSTM_model(X_train, y_train, X_test,
sc)
print(predictions_2[1:10])
actual_pred_plot(predictions_2)
```

Saving the model

```
model.save("GRU_NFTY_1min.pkl")
```

Part 2

Deploying the model using flask

Python file 2

App.py

Importing the required libraries

```
import numpy as np
from flask import Flask, request, jsonify, render_template
import pandas as pd
import keras.models
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
import csv
```

Using a function to normalize the data

```
def ts_train_test_normalize(all_data, time_steps):
# create training and test set

ts_train = all_data
```

```
ts_train = all_data.iloc[:, 0:1].values

ts_train_len = len(ts_train)
# scale the data
sc = MinMaxScaler(feature_range=(0, 1))
ts_train_scaled = sc.fit_transform(ts_train)

# create training data of s samples and t time steps
X_train = []
y_train_stacked = []
for i in range(time_steps, ts_train_len - 1):
X_train.append(ts_train_scaled[i - time_steps:i, 0])
X_train = np.array(X_train)

# Reshaping X_train for efficient modelling
X_train = np.reshape(X_train, (X_train.shape[0],
X_train.shape[1], 1))
return X_train
```

Defining Flask App and routes

```
app = Flask(__name__)


@app.route('/')
def home():
return render_template("index.html")


@app.route('/predict', methods=['POST', "GET"])
def stock_analysis():
model = keras.models.load_model(r"GRU_NFTY_1min.pkl")
if request.method == "POST":
data = request.files['file']
input_data = pd.read_csv(data)
df = ts_train_test_normalize(input_data, time_steps=5)
output = model.predict(df)
output = pd.DataFrame(output)
# output = output.to_html()
plt.plot(output)
# plt.savefig("output2.png")
output.to_csv("OUTPUT2.csv")
return render_template('index.html',
prediction_text=output)
# return render_template("index.html",
prediction_test=plt.show())
```

```python
if __name__ == "__main__":
    app.run(debug=True)
```