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.

Note: I have performed many experiments this report is for the stock price prediction with 1 day interval

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

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
6. Yfinance

Importing the required libraries

```
import keras.models
import pandas as pd
import numpy as np
import yfinance as yf
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
```

Downloading the data with 1 day interval from yfinance

```
data = yf.download(tickers="MSFT")
# data = pd.read_csv("DataFrame.csv")
# Print the data
print(data.describe())
print(data.head())
print(data.tail())
print(data.columns)
```

Splitting the data into train and testing sets

```
print(data["Adj Close"])


def ts_train_test(all_data, time_steps, for_periods):
    # create training and test set
    ts_train = all_data[:'2017'].iloc[:, 0:1].values
    ts_test = all_data['2018':].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["Adj Close"][:'2017'],
```

```
                        all_data["Adj Close"]['2018':]),
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

```
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))
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)
```

Normalizing the Data

```
def ts_train_test_normalize(all_data, time_steps,
for_periods):
    # create training and test set
    ts_train = all_data[:'2017'].iloc[:, 0:1].values
    ts_test = all_data['2018':].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["Adj Close"][:'2017'],
                        all_data["Adj Close"]['2018':]),
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
```

Creating LSTM MODEL

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

    model = Sequential()
    model.add(LSTM(64, activation="tanh",
return_sequences=True, input_shape=(X_train.shape[1], 1)))
    model.add(LSTM(64, activation="tanh",
return_sequences=True))
    model.add(LSTM(64, activation="tanh",
```

```
return_sequences=True))
    model.add(LSTM(64, activation="tanh",
return_sequences=True))
    model.add(LSTM(32))
    model.add(Dense(1))

    model.compile(optimizer="adam", loss='mean_squared_error')

    # fit the RNN model
    model.fit(X_train, y_train, epochs=100, batch_size=150,
verbose=0)

    # Finalizing predictions
    predictions = model.predict(X_test)

    return model, predictions
```

Training and testing the model

```
X_train, y_train, X_test, sc = ts_train_test_normalize(data,
5, 2)
model, predictions_2 = model(X_train, y_train, X_test, sc)
print(predictions_2[1:10])
actual_pred_plot(predictions_2)
model.save("rnn.pkl")
```

Part 2

Deploying the model using flask

```
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


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


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"rnn.pkl")
    if request.method == "POST":
        data = request.files['file']
        input_data = pd.read_csv(data, parse_dates=["Date"])
        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("output.png")
        return render_template('index.html',
prediction_text=output)
       #return render_template('index.html',
prediction_text=plt.show())


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