Time Series and Survival Analysis Project

February 4, 2021

1 Summary

The dataset for this project originates from kaggle and contains Google daily stock prices between 2012 and 2016.

In this project, we will employ Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) to to predict stock market indices. We are interested in forecasting the 'Close' series.

2 Load and Exploratore the Data

[1]:
```
import sys
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.simplefilter(action='ignore')
import pandas as pd
from datetime import datetime
import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers import Dense, SimpleRNN, LSTM, Activation, Dropout
import math
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
```

[2]:
```
data = pd.read_csv('./Google_Stock_Price_Data.csv',sep=",")
data.head()
```

[2]:

	Date	Open	High	Low	Close	Volume
0	1/3/2012	325.25	332.83	324.97	663.59	7,380,500
1	1/4/2012	331.27	333.87	329.08	666.45	5,749,400
2	1/5/2012	329.83	330.75	326.89	657.21	6,590,300
3	1/6/2012	328.34	328.77	323.68	648.24	5,405,900
4	1/9/2012	322.04	322.29	309.46	620.76	11,688,800

[3]:
```
data['Close'].isnull().sum()
```

[3]: 0

[4]:
```python
data = data[['Date', 'Close']]
data.sample(5)
```

[4]:

	Date	Close
515	1/22/2014	1,161.83
1142	7/19/2016	736.96
998	12/21/2015	747.77
49	3/14/2012	614.3
520	1/29/2014	1,103.89

3 Feature Transformation

- Replace comma in **Close** column and convert values into float64
- Transform **Date** column into a datetime object

[5]:
```python
data['Close'] = data['Close'].str.replace(',','')
data['Close'] = data['Close'].apply(lambda x : float(x))
```

[6]:
```python
def make_date(row):
    return datetime(year = int(row.split('/')[2]),
                    month = int(row.split('/')[0]),
                    day = int(row.split('/')[1]))

data['Date'] = data['Date'].apply(make_date)
data.set_index(data.Date,inplace=True)
data.drop(columns=['Date'], inplace=True)

plt.plot(data, 'm')
```

[6]:
```
[<matplotlib.lines.Line2D at 0x20f160d2880>]
```

4 Split the Data and Apply Feature Scaling

- Split the data into train and test data sets using **timestep = 50 days**
- use **MinMaxScaler** to scale the data

[7]:
```
timesteps = 50
```

[8]:
```
train = data[:len(data)-timesteps]['Close'].values
test = data[len(train):]['Close'].values
train=train.reshape(train.shape[0],1)
test=test.reshape(test.shape[0],1)
```

[9]:
```
sc = MinMaxScaler(feature_range= (0,1))
train = sc.fit_transform(train)
```

[10]:
```
train_X = []
train_y = []

for i in range(timesteps, train.shape[0]):
    train_X.append(train[i-timesteps:i,0])
    train_y.append(train[i,0])

train_X = np.array(train_X)
train_X = train_X.reshape(train_X.shape[0], train_X.shape[1], 1)
train_y = np.array(train_y)
```

[11]:
```python
print('Training input shape: {}'.format(train_X.shape))
print('Training output shape: {}'.format(train_y.shape))
```

```
Training input shape: (1158, 50, 1)
Training output shape: (1158,)
```

[12]:
```python
inputs = data[len(data) - len(test) - timesteps:]
inputs = sc.transform(inputs)

test_X = []

for i in range(timesteps, 100):
    test_X.append(inputs[i-timesteps:i,0])

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

[13]:
```python
test_X.shape
```

[13]:
```
(50, 50, 1)
```

5 Train models

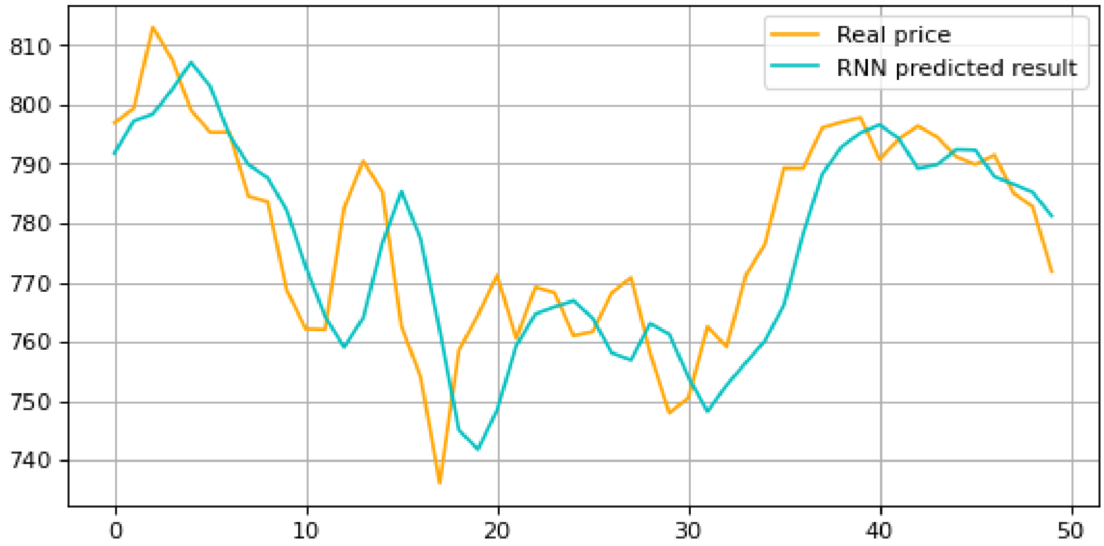
- Simple **RNN** layers each with 50 hidden units and tanh activation function per cell
- **LSTM** with 70 hidden units per cell
- Define the loss function and optimizer strategy
- Fit the model with 100 epochs
- Predict and plot the results

5.1 RNN

[14]:
```python
model = Sequential()

model.add(SimpleRNN(50, activation='tanh',
                    input_shape=(train_X.shape[1],1), return_sequences = True))
model.add(Dropout(0.2))
model.add(SimpleRNN(50, activation='tanh', return_sequences = True,))
model.add(Dropout(0.2))
model.add(SimpleRNN(50, activation='tanh', return_sequences = True,))
model.add(Dropout(0.2))
model.add(SimpleRNN(50, activation='tanh'))
# output layer to make final predictions
model.add(Dense(1))

model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(train_X, train_y, epochs=100, batch_size=32, verbose=0)
```

[14]:
```
<tensorflow.python.keras.callbacks.History at 0x20f196436d0>
```

[15]:
```
predicted = model.predict(test_X)
predicted = sc.inverse_transform(predicted)

plt.figure(figsize=(8,4), dpi=80, facecolor='w', edgecolor='k')
plt.plot(test,color="orange",label="Real price")
plt.plot(predicted,color="c",label="RNN predicted result")
plt.legend()
plt.grid(True)
plt.show()
```



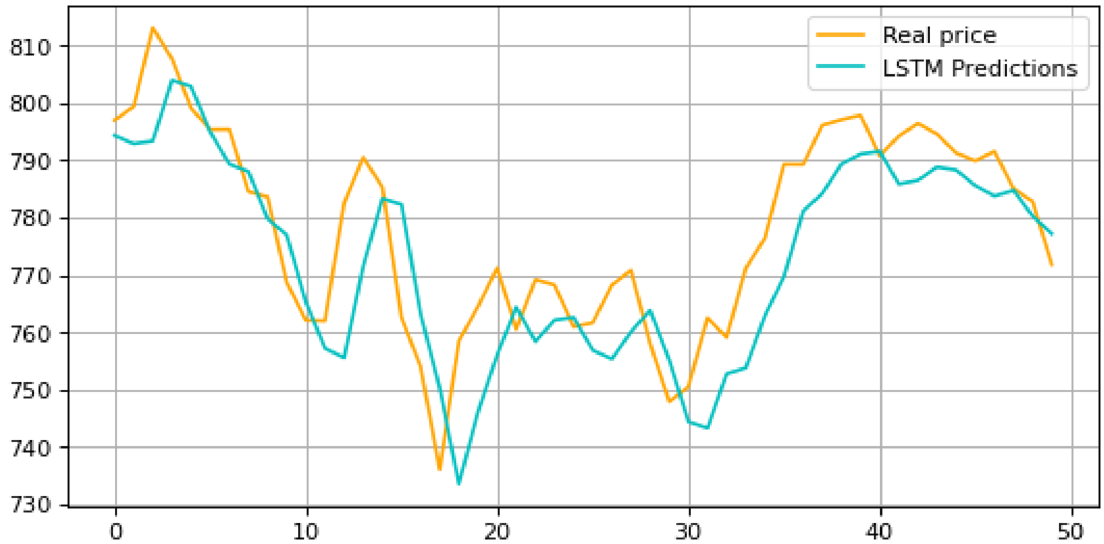
5.2 LSTM

[16]:
```
model2 = Sequential()
model2.add(LSTM(70, input_shape=(train_X.shape[1],1)))
model2.add(Dense(1))

model2.compile(loss='mean_squared_error', optimizer='adam')
model2.fit(train_X, train_y, epochs=100, batch_size=32, verbose=0)
```

[16]:
```
<tensorflow.python.keras.callbacks.History at 0x20f1f00da30>
```

[17]:
```
predicted2 = model2.predict(test_X)
predicted2 = sc.inverse_transform(predicted2)

plt.figure(figsize=(8,4), dpi=80, facecolor='w', edgecolor='k')
plt.plot(test,color="orange",label="Real price")
plt.plot(predicted2,color="c",label="LSTM Predictions")
plt.legend()
```

```python
plt.grid(True)
plt.show()
```



[18]:
```python
# RNN structure
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn (SimpleRNN)       (None, 50, 50)            2600
_________________________________________________________________
dropout (Dropout)            (None, 50, 50)            0
_________________________________________________________________
simple_rnn_1 (SimpleRNN)     (None, 50, 50)            5050
_________________________________________________________________
dropout_1 (Dropout)          (None, 50, 50)            0
_________________________________________________________________
simple_rnn_2 (SimpleRNN)     (None, 50, 50)            5050
_________________________________________________________________
dropout_2 (Dropout)          (None, 50, 50)            0
_________________________________________________________________
simple_rnn_3 (SimpleRNN)     (None, 50)                5050
_________________________________________________________________
dense (Dense)                (None, 1)                 51
=================================================================
Total params: 17,801
Trainable params: 17,801
```

```
Non-trainable params: 0
_________________________________________________________________
```

```
[19]: # LSTM structure
model2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 70)                20160
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 71
=================================================================
Total params: 20,231
Trainable params: 20,231
Non-trainable params: 0
_________________________________________________________________
```

6 Results

If we compare the model summary for **Simple RNN** with the model summary for **LSTM**, we can see that there are more trainable parameters for the **LSTM**, which explains why it took a longer time to train this model.

Overall the plots show that our **LSTM** model with a less complex structure still performed better than our Simple RNN.

7 Next Steps

To improve the quality of forecasts over many time steps, we'd need to use more data and more sophisticated LSTM model structures. We could try training with more data or increasing cell_units and running more training epochs.