LSTM Alpha

March 29, 2024

1 LSTM Alpha Vantage

[ ]:
```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
%matplotlib inline

from matplotlib.pylab import rcParams
rcParams['figure.figsize']=20,10
from tensorflow.keras.models import Sequential
import tensorflow as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Embedding
from tensorflow.keras.preprocessing import sequence
import os


from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler(feature_range=(0,1))
```

[ ]:
```
import pandas as pd
from alpha_vantage.timeseries import TimeSeries
import time
```

[ ]:
```
api_key = 'IKGHGW5MQOYG9QOL'
```

[ ]:
```
ts = TimeSeries(key=api_key, output_format='pandas')
df, meta_data = ts.get_intraday(symbol='MSFT', interval = '5min', outputsize =␣
 ↪'full')
print(df)
```

```
                     1. open  2. high  3. low  4. close  5. volume
date
2024-03-20 19:55:00   427.19   427.38  426.85    427.29     3811.0
```

```
2024-03-20 19:50:00   427.21   427.40  427.00    427.00     1286.0
2024-03-20 19:45:00   427.30   427.45  427.00    427.20     1463.0
2024-03-20 19:40:00   427.45   427.45  427.03    427.36      654.0
2024-03-20 19:35:00   427.50   427.50  427.03    427.24     1741.0
…                       …       …       …         …
2024-02-28 04:20:00   406.22   406.22  406.01    406.03      367.0
2024-02-28 04:15:00   406.44   406.49  406.01    406.22     1193.0
2024-02-28 04:10:00   406.42   406.68  406.21    406.49     1175.0
2024-02-28 04:05:00   406.34   406.51  406.18    406.30     1526.0
2024-02-28 04:00:00   406.48   407.44  406.16    406.30     1047.0

[3072 rows x 5 columns]
```

[ ]:
```
# For the default date string index behavior
ts = TimeSeries(key=api_key,output_format='pandas', indexing_type='date')
ts
print(ts)
```

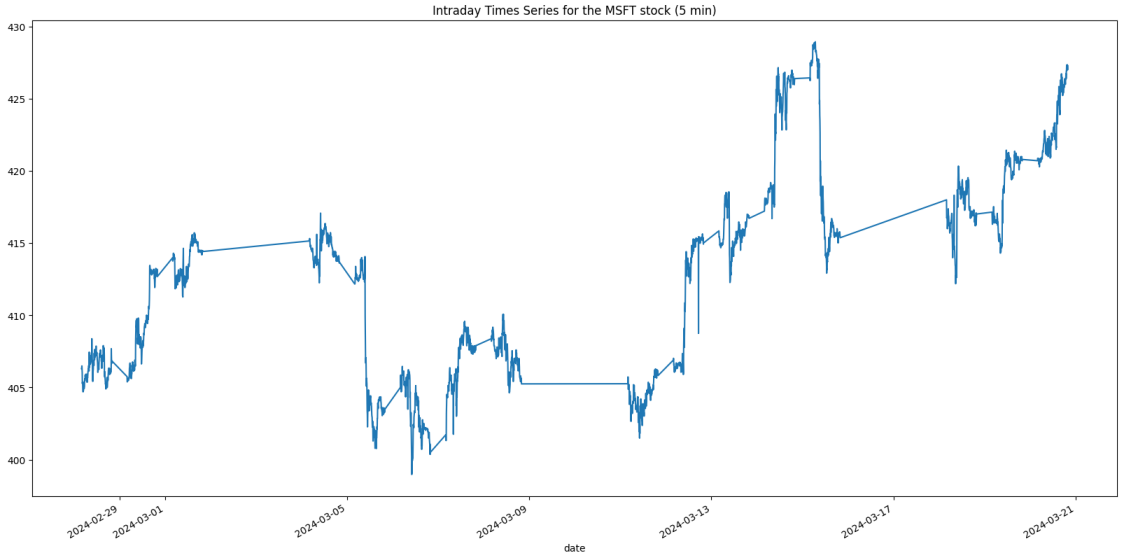
```
<alpha_vantage.timeseries.TimeSeries object at 0x0000016C3062C390>
```

[ ]:
```
from alpha_vantage.timeseries import TimeSeries
from pprint import pprint
ts = TimeSeries(key=api_key, output_format='pandas')
data, meta_data = ts.get_intraday(symbol='MSFT',interval='5min',␣
 ↪outputsize='full')
print(data)
```

```
                    1. open  2. high  3. low  4. close  5. volume
date
2024-03-20 19:55:00   427.19   427.38  426.85    427.29     3811.0
2024-03-20 19:50:00   427.21   427.40  427.00    427.00     1286.0
2024-03-20 19:45:00   427.30   427.45  427.00    427.20     1463.0
2024-03-20 19:40:00   427.45   427.45  427.03    427.36      654.0
2024-03-20 19:35:00   427.50   427.50  427.03    427.24     1741.0
…                       …       …       …         …
2024-02-28 04:20:00   406.22   406.22  406.01    406.03      367.0
2024-02-28 04:15:00   406.44   406.49  406.01    406.22     1193.0
2024-02-28 04:10:00   406.42   406.68  406.21    406.49     1175.0
2024-02-28 04:05:00   406.34   406.51  406.18    406.30     1526.0
2024-02-28 04:00:00   406.48   407.44  406.16    406.30     1047.0

[3072 rows x 5 columns]
```

[ ]:
```
from alpha_vantage.timeseries import TimeSeries
import matplotlib.pyplot as plt

ts = TimeSeries(key='IKGHGW5MQOYG9QOL', output_format='pandas')
```

```
df, meta_data = ts.get_intraday(symbol='MSFT',interval='5min',␣
 ↪outputsize='full')
df['4. close'].plot()
plt.title('Intraday Times Series for the MSFT stock (5 min)')
plt.show()
```



[ ]:
```
i = 1
# while i==1:
#     df, meta_data = ts.get_intraday(symbol='MSFT', interval = '1min',␣
 ↪outputsize = 'full')
#     df.to_csv("Dataset\Out.csv")
#     time.sleep(60)
```

[ ]:
```
df, meta_data = ts.get_intraday(symbol='MSFT', interval = '1min', outputsize =␣
 ↪'full')
df.to_csv("Dataset\Out.csv")
```

[ ]:
```
df = pd.read_csv("Dataset\Out.csv")
df.head()
```

[ ]:

	date	1. open	2. high	3. low	4. close	5. volume
0	2024-03-20 19:59:00	427.095	427.29	426.90	427.29	157.0
1	2024-03-20 19:58:00	427.140	427.29	426.85	427.07	945.0
2	2024-03-20 19:57:00	427.350	427.35	427.00	427.15	326.0
3	2024-03-20 19:56:00	427.020	427.38	427.00	427.08	2076.0
4	2024-03-20 19:55:00	427.190	427.30	427.00	427.25	307.0

[ ]:
```
data=df.sort_index(ascending=True,axis=0)
new_dataset=pd.DataFrame(index=range(0,len(df)),columns=['date','4. close'])

for i in range(0,len(data)):
    new_dataset["date"][i]=df['date'][i]
    new_dataset["4. close"][i]=df["4. close"][i]
```

[ ]:
```
new_dataset.index=new_dataset.date
new_dataset.drop("date",axis=1,inplace=True)

final_dataset=new_dataset.values

train_data=final_dataset[0:987,:]
valid_data=final_dataset[987:,:]

scaler=MinMaxScaler(feature_range=(0,1))
scaled_data=scaler.fit_transform(final_dataset)

x_train_data,y_train_data=[],[]

for i in range(60,len(train_data)):
    x_train_data.append(scaled_data[i-60:i,0])
    y_train_data.append(scaled_data[i,0])

x_train_data,y_train_data=np.array(x_train_data),np.array(y_train_data)

x_train_data=np.reshape(x_train_data,(x_train_data.shape[0],x_train_data.
 ↪shape[1],1))
```

Build and train the LSTM model

[ ]:
```
lstm_model=Sequential()
lstm_model.add(LSTM(units=50,return_sequences=True,input_shape=(x_train_data.
 ↪shape[1],1)))
lstm_model.add(LSTM(units=50))
lstm_model.add(Dense(1))




lstm_model.compile(loss='mean_squared_error',optimizer='adam')
lstm_model.fit(x_train_data,y_train_data,epochs=10,batch_size=1,verbose=2)

inputs_data=new_dataset[len(new_dataset)-len(valid_data)-60:].values
inputs_data=inputs_data.reshape(-1,1)
inputs_data=scaler.transform(inputs_data)
```

```
Epoch 1/10
```

```
c:\Users\Nithin Kodipyaka\AppData\Local\Programs\Python\Python311\Lib\site-
packages\keras\src\layers\rnn\rnn.py:204: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

```
927/927 - 23s - 25ms/step - loss: 0.0026
Epoch 2/10
927/927 - 19s - 21ms/step - loss: 2.5203e-04
Epoch 3/10
927/927 - 20s - 22ms/step - loss: 2.1982e-04
Epoch 4/10
927/927 - 20s - 21ms/step - loss: 2.4302e-04
Epoch 5/10
927/927 - 19s - 21ms/step - loss: 2.1827e-04
Epoch 6/10
927/927 - 19s - 21ms/step - loss: 1.6195e-04
Epoch 7/10
927/927 - 19s - 20ms/step - loss: 1.7229e-04
Epoch 8/10
927/927 - 22s - 23ms/step - loss: 1.3132e-04
Epoch 9/10
927/927 - 19s - 20ms/step - loss: 1.5188e-04
Epoch 10/10
927/927 - 19s - 21ms/step - loss: 1.1911e-04
```

[ ]:
```
X_test=[]
for i in range(60,inputs_data.shape[0]):
    X_test.append(inputs_data[i-60:i,0])
X_test=np.array(X_test)
```

[ ]:
```
print(X_test.size)
```

```
856080
```

[ ]:
```
X_test=np.reshape(X_test,(X_test.shape[0],X_test.shape[1],1))
closing_price=lstm_model.predict(X_test)
closing_price=scaler.inverse_transform(closing_price)
predicted_closing_price=lstm_model.predict(X_test)
predicted_closing_price=scaler.inverse_transform(predicted_closing_price)
```

```
446/446             13s 29ms/step
446/446             9s 19ms/step
```

[ ]:
```
lstm_model.save("saved_lstm_model.h5")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
```

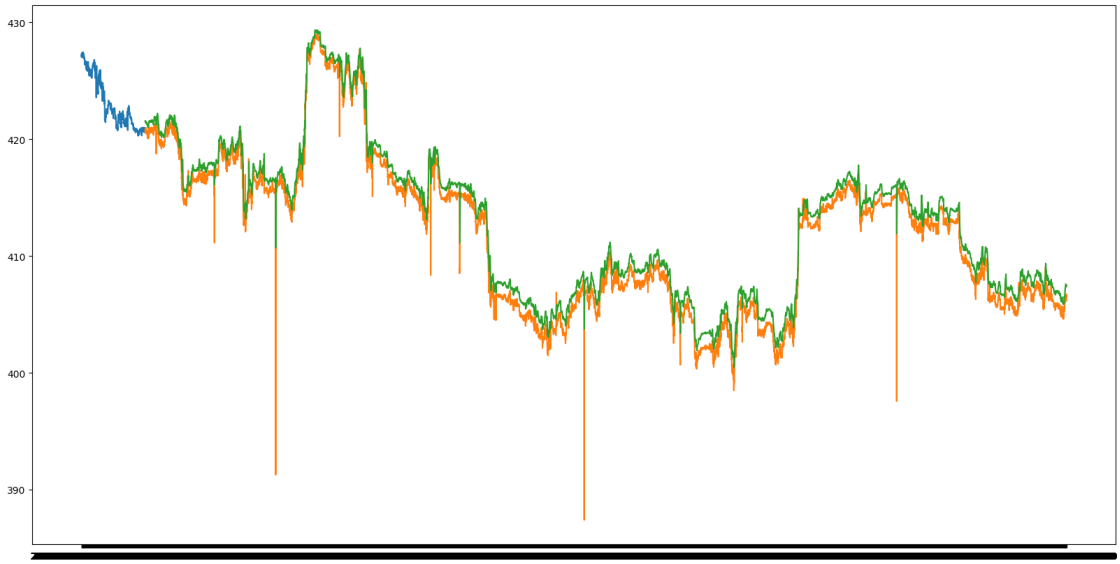
```
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.
```

Visualize the predicted stock costs with actual stock costs

[ ]:
```
train_data=new_dataset[:987]
valid_data=new_dataset[987:]
valid_data['Predictions']=predicted_closing_price
plt.plot(train_data["4. close"])
plt.plot(valid_data[['4. close',"Predictions"]])
```

```
C:\Users\Nithin Kodipyaka\AppData\Local\Temp\ipykernel_1928\1343216786.py:3:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  valid_data['Predictions']=predicted_closing_price
```

[ ]:
```
[<matplotlib.lines.Line2D at 0x16c304038d0>,
 <matplotlib.lines.Line2D at 0x16c34c68490>]
```



[ ]:
```
from sklearn.metrics import mean_squared_error, mean_absolute_error
import math
```

[ ]:
```
lstm_model = tf.keras.models.load_model("saved_lstm_model.h5")
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be
built. `model.compile_metrics` will be empty until you train or evaluate the
```

model.

[ ]:
```
inputs_data = new_dataset[len(new_dataset)-len(valid_data)-60:].values
inputs_data = inputs_data.reshape(-1,1)
inputs_data = scaler.transform(inputs_data)
X_test = []
for i in range(60, inputs_data.shape[0]):
    X_test.append(inputs_data[i-60:i,0])
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
```

[ ]:
```
predicted_closing_price = lstm_model.predict(X_test)
predicted_closing_price = scaler.inverse_transform(predicted_closing_price)
```

```
446/446             6s 12ms/step
```

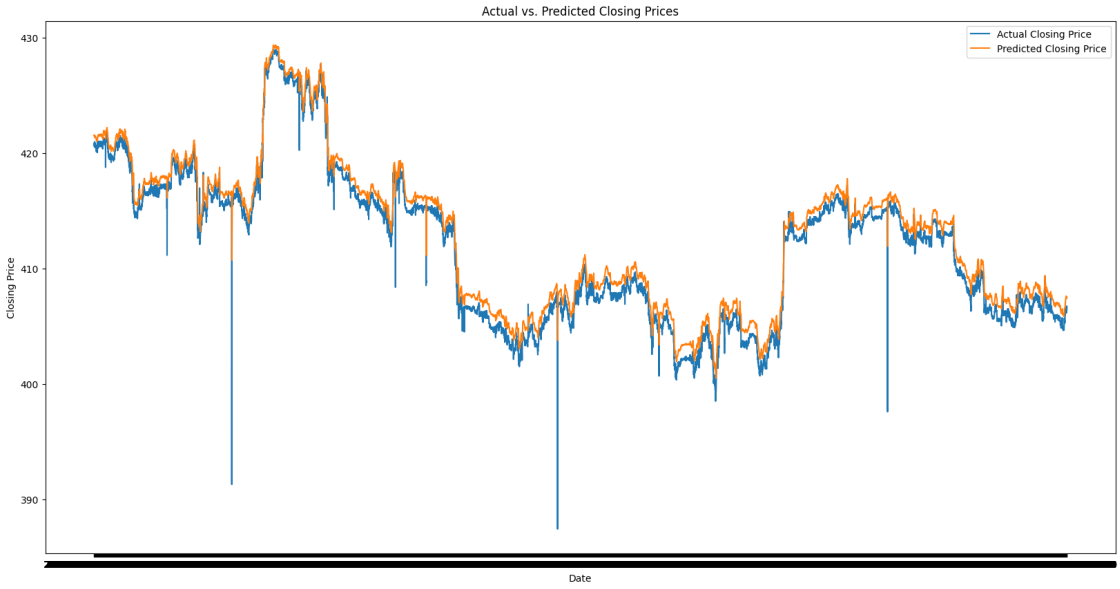
[ ]:
```
true_closing_price = valid_data['4. close'].values
mse = mean_squared_error(true_closing_price, predicted_closing_price)
mae = mean_absolute_error(true_closing_price, predicted_closing_price)
rmse = math.sqrt(mse)
```

[ ]:
```
print("Mean Squared Error (MSE):", mse)
print("Mean Absolute Error (MAE):", mae)
print("Root Mean Squared Error (RMSE):", rmse)
```

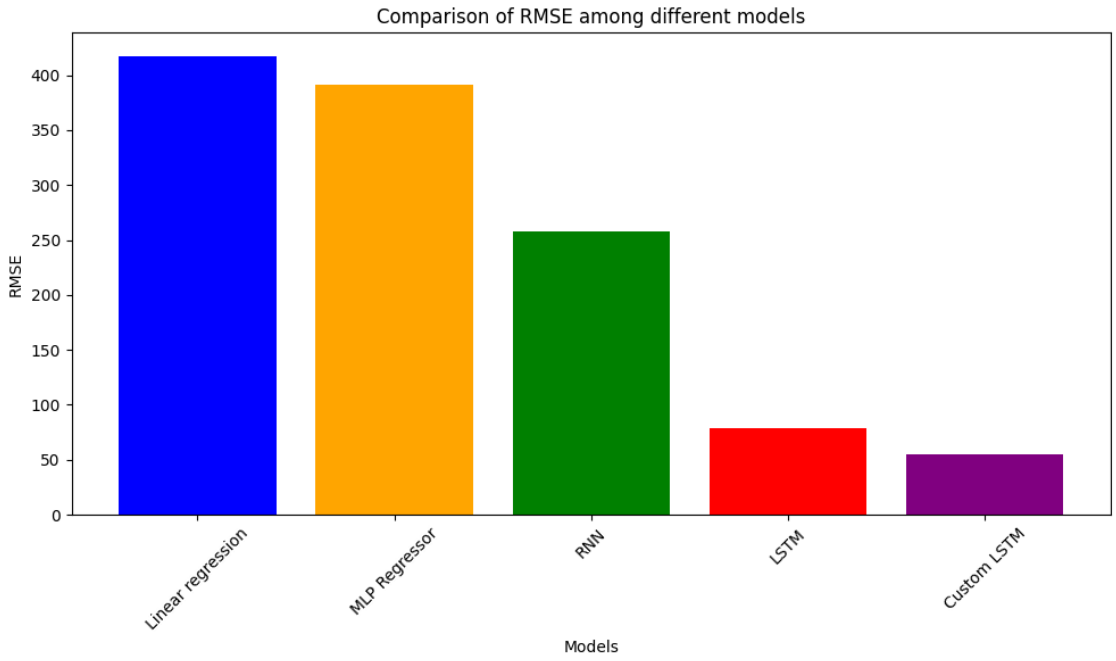
```
Mean Squared Error (MSE): 1.194742579639195
Mean Absolute Error (MAE): 0.9909038705357924
Root Mean Squared Error (RMSE): 1.093042807779821
```

[ ]:
```
plt.plot(valid_data.index, true_closing_price, label='Actual Closing Price')
plt.plot(valid_data.index, predicted_closing_price, label='Predicted Closing␣
↪Price')
plt.title('Actual vs. Predicted Closing Prices')
plt.xlabel('Date')
plt.ylabel('Closing Price')
plt.legend()
```

[ ]:
```
<matplotlib.legend.Legend at 0x16c438ef910>
```

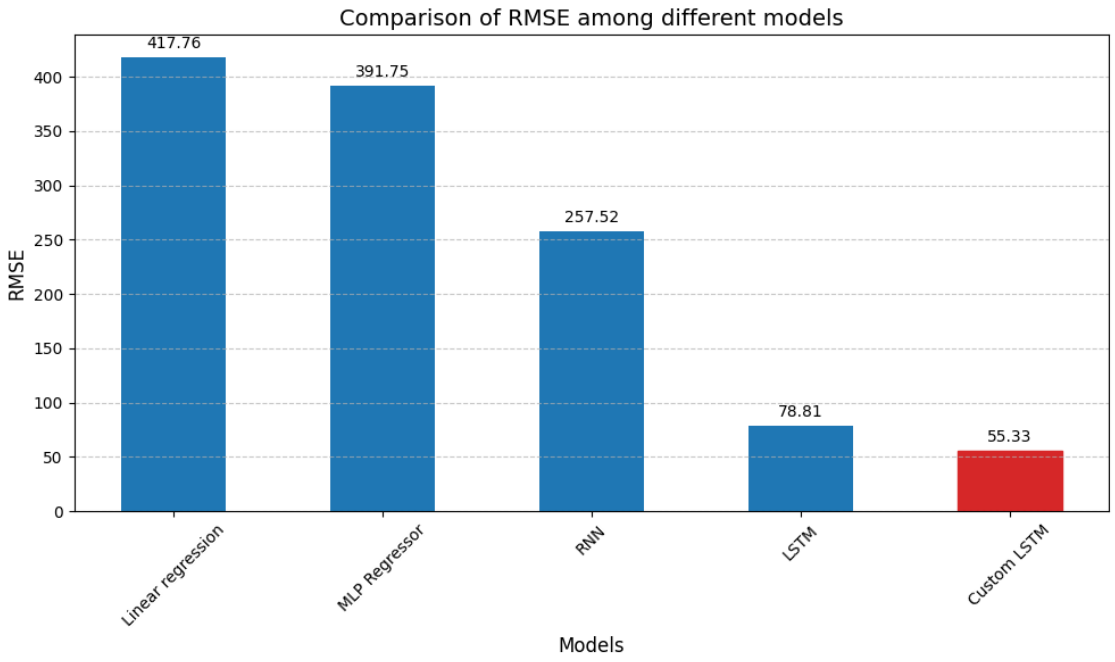


[ ]:
```python
import matplotlib.pyplot as plt

# Data
models = ['Linear regression', 'MLP Regressor', 'RNN', 'LSTM', 'Custom LSTM']
rmse = [417.757, 391.752, 257.52, 78.81, 55.33]

# Create bar chart
plt.figure(figsize=(10, 6))
plt.bar(models, rmse, color=['blue', 'orange', 'green', 'red', 'purple'])
plt.xlabel('Models')
plt.ylabel('RMSE')
plt.title('Comparison of RMSE among different models')
plt.xticks(rotation=45)
plt.tight_layout()

# Show plot
plt.show()
```



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

# Data
models = ['Linear regression', 'MLP Regressor', 'RNN', 'LSTM', 'Custom LSTM']
rmse = [417.757, 391.752, 257.52, 78.81, 55.33]

# Colors
colors = ['#1f77b4', '#1f77b4', '#1f77b4', '#1f77b4', '#d62728']  # Dark blue␣
 ↪for all bars, red for Custom LSTM

# Create bar chart
plt.figure(figsize=(10, 6))
bars = plt.bar(models, rmse, color=colors, width=0.5)  # Adjust the width for␣
 ↪thinner bars

# Highlighting the "Custom LSTM" bar with a different color
bars[-1].set_color('#d62728')

# Add data labels
for bar, value in zip(bars, rmse):
    plt.text(bar.get_x() + bar.get_width() / 2, bar.get_height() + 5, f'{value:.
 ↪2f}', ha='center', va='bottom')

# Customize chart appearance
plt.xlabel('Models', fontsize=12)
```

```
plt.ylabel('RMSE', fontsize=12)
plt.title('Comparison of RMSE among different models', fontsize=14)
plt.xticks(rotation=45, fontsize=10)
plt.yticks(fontsize=10)
plt.grid(axis='y', linestyle='--', alpha=0.7)

# Show plot
plt.tight_layout()
plt.show()
```



[ ]: