10. LSTMClosePricePrediction

August 30, 2023

[ ]:
```
# dataset @ https://finance.yahoo.com/quote/MSFT/history/

# If you want the exact same dataset as the YouTube video,
# use this link: https://drive.google.com/file/d/
 ↪1WLm1AEYgU28Nk4lY4zNkGPSctdImbhJI/view?usp=sharing
```

[ ]:
```
import pandas as pd
import datetime

# Stocks :- AAPL, MSFT, AMZN, NVDA, TSLA, GOOGL, UNH
# Sector Indices :- SPINF (^SP500-45)


method= 'TextBlob'
ticker = "AAPL"

df = pd.read_csv(f"SentimentAnalysis/{method}/
 ↪{ticker}sentiment_agg_stock_trend_output.csv")
df
```

[ ]:
```
df = df[['Date', 'Close', 'polarity']]
```

[ ]:
```
# # Normalize the 'Close' and 'polarity' columns using min-max scaling
# min_close = df['Close'].min()
# max_close = df['Close'].max()

# min_polarity = df['polarity'].min()
# max_polarity = df['polarity'].max()

# df['Close'] = (df['Close'] - min_close) / (max_close - min_close)
# df['polarity'] = (df['polarity'] - min_polarity) / (max_polarity -␣
 ↪min_polarity)
```

[ ]:
```
df["Date"] = pd.to_datetime(df['Date'], format="%Y-%m-%d")
df.index = df.pop('Date')
```

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

plt.plot(df.index, df['Close'])
plt.xticks(rotation=90)
```

[ ]:
```
import numpy as np

def df_to_windowed_df(dataframe, first_date_str, last_date_str, n=3):
  first_date = pd.to_datetime(first_date_str, format="%Y-%m-%d")
  last_date  = pd.to_datetime(last_date_str, format="%Y-%m-%d")

  target_date = first_date

  dates = []
  X, Y = [], []

  last_time = False
  while True:
    df_subset = dataframe.loc[:target_date].tail(n+1)

    if len(df_subset) != n+1:
      print(f'Error: Window of size {n} is too large for date {target_date}')
      return

    values = df_subset['Close'].to_numpy()
    x, y = values[:-1], values[-1]

    dates.append(target_date)
    X.append(x)
    Y.append(y)

    next_week = dataframe.loc[target_date:target_date+datetime.
 ↪timedelta(days=7)]
    next_datetime_str = str(next_week.head(2).tail(1).index.values[0])
    next_date_str = next_datetime_str.split('T')[0]
    year_month_day = next_date_str.split('-')
    year, month, day = year_month_day
    next_date = datetime.datetime(day=int(day), month=int(month),␣
 ↪year=int(year))

    if last_time:
      break

    target_date = next_date

    if target_date == last_date:
      last_time = True
```

```
  ret_df = pd.DataFrame({})
  ret_df['Target Date'] = dates

  X = np.array(X)
  for i in range(0, n):
    X[:, i]
    ret_df[f'Target-{n-i}'] = X[:, i]

  ret_df['Target'] = Y

  return ret_df

# Start day second time around: '2021-03-25'
windowed_df = df_to_windowed_df(df,
                                '2021-03-25',
                                '2022-03-25',
                                n=3)
windowed_df
```

[ ]:
```
def windowed_df_to_date_X_y(windowed_dataframe):
  df_as_np = windowed_dataframe.to_numpy()

  dates = df_as_np[:, 0]

  middle_matrix = df_as_np[:, 1:-1]
  X = middle_matrix.reshape((len(dates), middle_matrix.shape[1], 1))

  Y = df_as_np[:, -1]

  return dates, X.astype(np.float32), Y.astype(np.float32)

dates, X, y = windowed_df_to_date_X_y(windowed_df)

dates.shape, X.shape, y.shape
```

[ ]:
```
X
```

[ ]:
```
q_80 = int(len(dates) * .8)
q_90 = int(len(dates) * .9)

dates_train, X_train, y_train = dates[:q_80], X[:q_80], y[:q_80]

dates_val, X_val, y_val = dates[q_80:q_90], X[q_80:q_90], y[q_80:q_90]
dates_test, X_test, y_test = dates[q_90:], X[q_90:], y[q_90:]

plt.plot(dates_train, y_train)
```

```
plt.plot(dates_val, y_val)
plt.plot(dates_test, y_test)
plt.xticks(rotation=90)
plt.legend(['Train', 'Validation', 'Test'])
```

[ ]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import layers

model = Sequential([layers.Input((3, 1)),
                    layers.LSTM(64),
                    layers.Dense(32, activation='relu'),
                    layers.Dense(32, activation='relu'),
                    layers.Dense(1)])

model.compile(loss='mse',
              optimizer=Adam(learning_rate=0.001),
              metrics=['mean_absolute_error'])

model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=100)
```

[ ]:
```
loss, mae = model.evaluate(X_test, y_test, verbose=0)
print(f"Mean Squared Error on the test set: {loss}")
```

[ ]:
```
train_predictions = model.predict(X_train).flatten()

plt.plot(dates_train, train_predictions)
plt.plot(dates_train, y_train)
plt.xticks(rotation=90)
plt.legend(['Training Predictions', 'Training Observations'])
```

[ ]:
```
val_predictions = model.predict(X_val).flatten()

plt.plot(dates_val, val_predictions)
plt.plot(dates_val, y_val)
plt.xticks(rotation=90)
plt.legend(['Validation Predictions', 'Validation Observations'])
```

[ ]:
```
test_predictions = model.predict(X_test).flatten()

plt.plot(dates_test, test_predictions)
plt.plot(dates_test, y_test)
plt.xticks(rotation=90)
plt.legend(['Testing Predictions', 'Testing Observations'])
```

[ ]:
```
plt.plot(dates_train, train_predictions)
plt.plot(dates_train, y_train)
```

```
plt.plot(dates_val, val_predictions)
plt.plot(dates_val, y_val)
plt.plot(dates_test, test_predictions)
plt.plot(dates_test, y_test)
plt.xticks(rotation=90)
plt.legend(['Training Predictions',
            'Training Observations',
            'Validation Predictions',
            'Validation Observations',
            'Testing Predictions',
            'Testing Observations'])
```

[ ]: