Predicting Stock Prices Using LSTM Networks

Project Team Number: 8

Project Team Members: Aidan Pierre-Louis

Abstract: This project explores the use of Long Short-Term Memory (LSTM) networks for predicting and forecasting stock prices. I implemented an LSTM model to predict daily closing prices, utilizing historical data split into training, validation, and test sets. We examined iterative prediction methods to extend forecasting beyond initial test set predictions. Key metrics include prediction errors, mean square error evaluation of testing data, and visual comparisons of predicted vs. true stock prices.

Contributions: All work done by Aidan Pierre-Louis with the assistance of ChatGPT in resolving errors, fixing mal-funcitioning code, and writing pseudocode

Link to Project Collab Notebook:

https://colab.research.google.com/drive/1_2_zCkfEYM46hKYi3eYZbH8PyLNmmcMM?usp=sharing

Introduction

Problem Statement: Forecasting stock prices is a complex task influenced by numerous factors and exhibits significant volatility. Accurate prediction of stock prices can benefit investors and financial analysts by providing insights into future market trends. The purpose of this project is to create an LSTM neural network that could hypothetically be used to aid in the prediction of stock market trends, particularly within the Dow Jones market, by such traders and analysts.

Related Work: Baseline methods include traditional statistical techniques like ARIMA [1] and more recent approaches such as machine learning models including RNNs and LSTMs [2]. These methods have shown varied success, with LSTMs often outperforming due to their ability to capture long-term dependencies in time series data compared to more traditional autoregressive models. More recently Transformer networks have shown even better comprehension of long-term dependencies in the time series natured data of stock prices [3].

Data

Source of the Dataset: yfinance Python library was used to download datasets and can be found at https://pypi.org/project/yfinance/. yfinance is an open-source Python library that uses Yahoo! Finance's publicly available APIs to scrape market data. Refer to appendix A for code example.

Details/Format of Data Samples: The dataset includes historical prices of stocks. Each record contains a date and corresponding price features. For the purposes of this project, only daily closing prices were used in the training of LSTM models for predicting stock prices.

Data Partitioning: Data was split into training (70%), validation (15%), and test (15%) sets to train and evaluate the performance of the LSTM models.

Data Preprocessing: Data was normalized to a range suitable for LSTM input (-1, 1). The closing prices were converted into sequences of past prices used for training and validation.

Missing price values were dropped from the datasets. A process of lagging the price features was used to leverage the temporal significance of more recent price values in the prediction of future prices and was passed to the model as the second (window size) dimension of the input data. Refer to appendix B for relevant code. After lagging price features, a third dimension containing the closing price feature for that sample date was added to be properly processed by the LSTM.

Sample Feature Vectors: The shape of the input data is (number of samples, lagged price feature window, price feature) where number of samples is equal to the number of days in the dataset and price feature is the price at those days. The lagged price feature window is an array of size n_lags, a hyperparameter specifying the number of sample dates to include in the window, where index 0 in the array is the price at that day, 1 is the price the previous day, …, and index n_lags-1 is the price of the day n_lags-1 days before. Refer to appendix C for an example of the data before and after preprocessing.

Tasks Performed Overview

- Designing and Implementing an LSTM model for time series forecasting of stock prices
- Training models on historical stock prices and validating performance
- Implementing iterative prediction method to forecast predictions beyond dataset horizon
- Libraries used: TensorFlow, Numpy, Pandas, Matplotlib, Sklearn

Model Design: Models were designed with 2 LSTM layers, 2 dropout layers and 2 dense layers. Hyperparameter hidden_units specified the number of units in each LSTM layer, and p represented the probability of dropout in dropout layers. Attempted hidden units of (50, 60, 100) and probability of (0.1, 0.2, 0.25, 0.5) then settling with 50 and 0.2 respectively witnessing their consistency in lowering loss prediction and underfitting. Interestingly overfitting was scarce, to be further discussed in the following section. Refer to appendix D for relevant code.

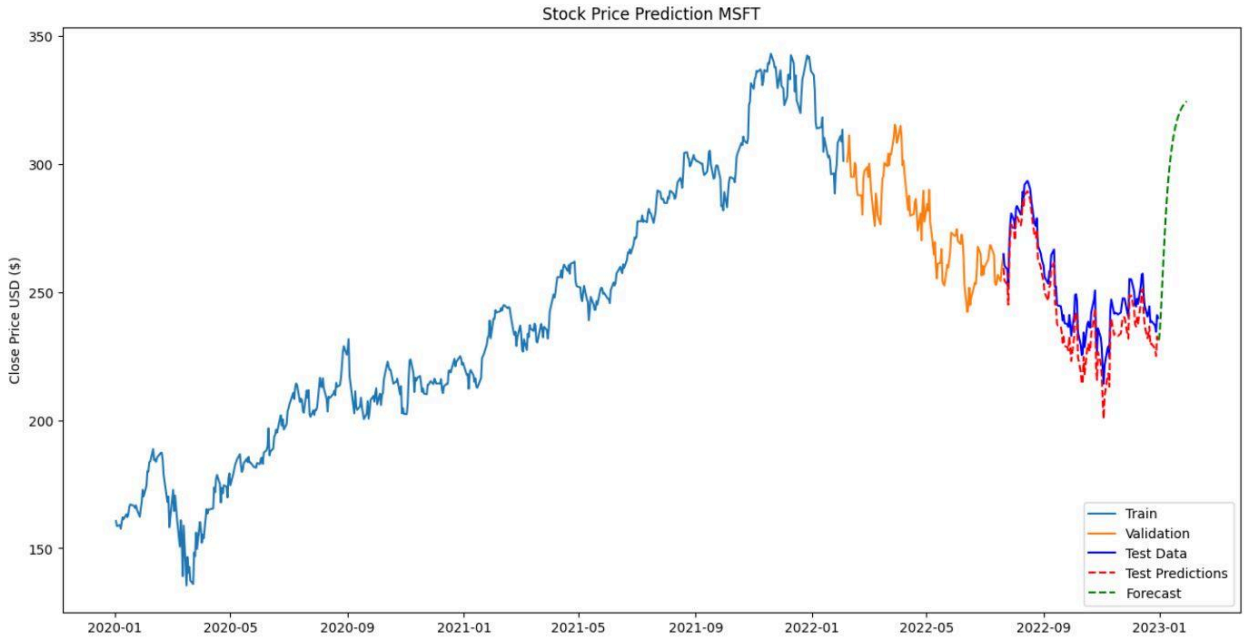
Model Training: Models were trained using Adam optimizer with specified learning rates, and loss was computed during training using Mean Squared Error. After experimenting with the number of epochs(50, 100, 200), training loader batch size(16, 32, 128, 256), and learning rate(0.1, 0.01, 0.001) hyperparameters, 100, 16, and 0.001 were chosen respectively due to the speed at which the model's loss decreased during training. Loss minimization was not particularly prone to these hyperparameters as it were to the lagging window size, dropout probability, and number of hidden units in the LSTM layers. Refer to appendix E for relevant code.

Model Forecasting: Forecasting stock price predictions beyond the end of the dataset was implemented via a method of iteratively predicting the next days closing price based on the previous input sequence data, appending it to the input sequence and then continuing to predict the amount of future days specified by variable forecast_length. This hyperparameter is chosen at user discretion, though because the predictions are rolling forward, the theoretical loss compounds making it less reliable the further it predicts. This has proven to be somewhat consistent in forecasting the trajectory of the stock but seemingly fails to replicate stock price volatility. Refer to appendix F for relevant code.

Results and Discussions

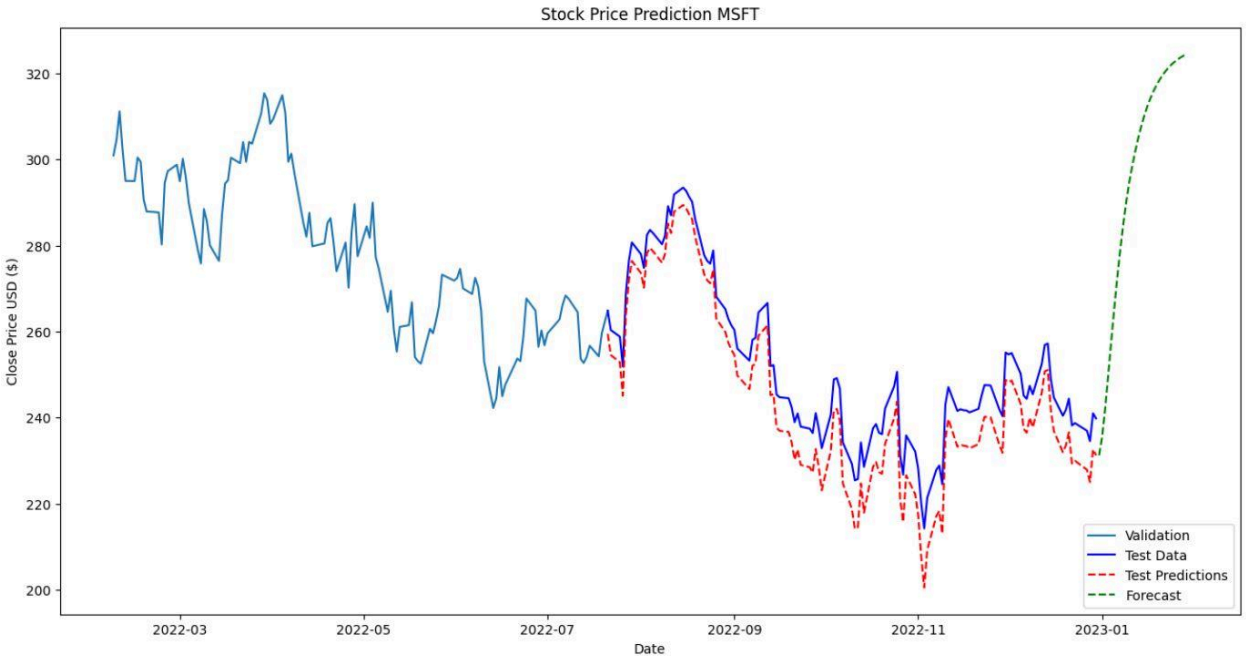
Experiments Performed: Models were fitted to various Dow Jones companies' stocks throughout varying time periods. The following figure is an example of Microsoft data from 2020-01-01 to 2023-01-01, the legend represents which parts of the data were used as training, validation and testing, with forecasting data extending beyond the total set.

Figure 1: Microsoft (MSFT) Whole Dataset



The following figures focus on the validation and testing data to better illustrate the model's ability to predict and forecast stock prices.

Figure 2: Microsoft (MSFT) Excluding Training Data



The next figure is the training and validation loss of MSFT. What is worth noting is how unusual the validation loss is less than the training loss, which is a trend prevalent in many of the models that seem to predict the data well. Some research and consideration of the model's ability to predict other companies' stock prices, my current working hypothesis is that, due to the vast size of the training data relative to the validation, the model is underfitting because the broader directional trends in the data across longer sequences are difficult for it to capture, compared to the validation data which covers a smaller window. Therefore it seems it is easier for it to predict the validation and testing sets because they don't contain these longer term temporal dependencies driving broader directional trends spanning months and years, but rather weeks and months. Test loss was computed using MSE.

Figure 3: Microsoft (MSFT) Loss Data

Many of the plots for each of the companies exhibited much of the same as the above for Microsoft but there were outliers such as the data for Netflix.

Figure 4: Netflix (NFLX) Excluding Training Data

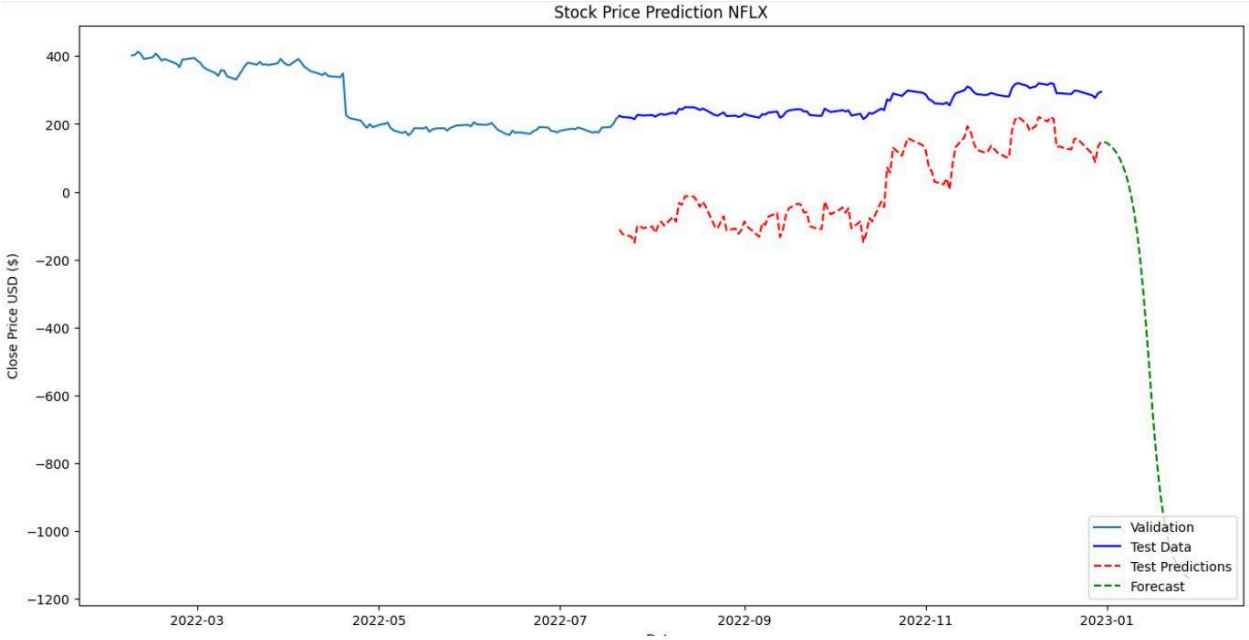
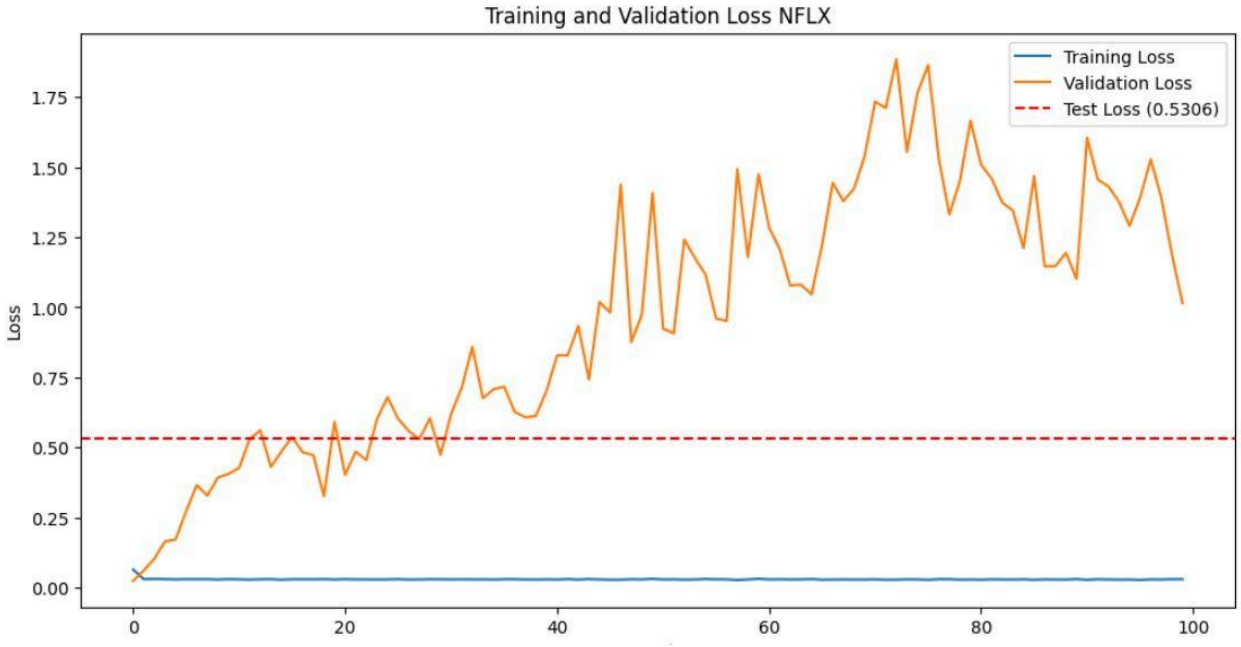


Figure 5: Netflix (NFLX) Loss Data



As can be seen from Figure 4, the model seemed to struggle with capturing the price trends in companies that had relatively stable price trends, i.e. the price changes were relatively small albeit more frequent at times. This is further reinforced by the far more volatile predictions made

by the model in Figure 4 and the overfitting in Figure 5. Changes to the n_lags lagging window size hyperparameter remedies this to varying degrees, but for the sake of presentation, consistency in the value of n_lags was decided. More thorough experimentation on which hyperparameters pair nicely with different company stocks would be needed to effectively gauge how they are linked to aiding the model in interpreting different datasets with different volatility and trend patterns.

Conclusion: The LSTM network designed in this project showed impressive results in its ability to comprehend and predict the patterns of many Dow Jones companies' stock market data, though it struggled with others. While forecasting results failed to provide precise prediction in price trend patterns, there was consistency in the direction of the forecasts for datasets the model fit well. There is potential in improving the model's design complexity to perhaps improve predictive ability for a larger variety of stock price datasets. Further experimentation with the several hyperparameters of the model could provide better insight into their relationship with the model's ability to comprehend, predict, and forecast a larger variety of datasets as well.

References

1. A. A. Ariyo, A. O. Adewumi and C. K. Ayo, "Stock Price Prediction Using the ARIMA Model," 2014 UKSim-AMSS 16th International Conference on Computer Modelling and Simulation, Cambridge, UK, 2014, pp. 106-112, doi: 10.1109/UKSim.2014.67. https://doi.org/10.1109/UKSim.2014.67
2. Nair, S., & Narayanan, A. (2020). Time series forecasting using deep learning techniques: A review. *Procedia Computer Science, 172*, 25-32. https://doi.org/10.1016/j.procs.2020.03.049

3. Ding, Q., Wu, S., Sun, H., Guo, J., & Guo, J. (2020, July). Hierarchical Multi-Scale Gaussian Transformer for Stock Movement Prediction. In *IJCAI* (pp. 4640-4646). https://idea.edu.cn/person/guojian/assets/papers/HMSG-transformer-2020.pdf

Appendix

Figure A: Downloading, Lagging, Scaling Data

```python
# Prepare stock data with lagged features
def prepare_stock_data(stock_symbol, n_lags, start='2020-01-01', end='2023-01-01'):
    data = yf.download(stock_symbol, start=start, end=end)
    series = data['Close'].dropna()
    df = create_lagged_features(series, n_lags)
    scaler = MinMaxScaler(feature_range=(-1, 1))
    #scaler = MinMaxScaler(feature_range=(0, 1))
    scaled_data = scaler.fit_transform(df)
    return scaled_data, scaler, series

# Prepare data, Hyperparam lag features
n_lags = 10
comp_idx = 20 # change the index here for selecting dow jones company to train data on
scaled_data, scaler, series = prepare_stock_data(dowjones_comps[comp_idx], n_lags)
```

Figure B: Lagging Function

```python
# Create lagged features for LSTM
def create_lagged_features(series, n_lags):
    df = pd.DataFrame(series)
    for i in range(1, n_lags + 1):
        df[f'lag_{i}'] = series.shift(i)
    #df.dropna(inplace=True)  # Drop rows with NaN values which are the result of shifting
    df.fillna(0.0, inplace=True) # replace rows with NaN values from shifting to prevent l
    return df
```

Figure C: Data Before and After Preprocessing

```python
#data = yf.download('AAPL', start='2020-01-01', end='2023-01-01')
data.iloc[:10, :]
```

Date	Open	High	Low	Close	Adj Close	Volume
2020-01-02	74.059998	75.150002	73.797501	75.087502	72.960472	135480400
2020-01-03	74.287498	75.144997	74.125000	74.357498	72.251129	146322800
2020-01-06	73.447502	74.989998	73.187500	74.949997	72.826866	118387200
2020-01-07	74.959999	75.224998	74.370003	74.597504	72.484344	108872000
2020-01-08	74.290001	76.110001	74.290001	75.797501	73.650352	132079200
2020-01-09	76.809998	77.607498	76.550003	77.407501	75.214752	170108400
2020-01-10	77.650002	78.167503	77.062500	77.582497	75.384781	140644800
2020-01-13	77.910004	79.267502	77.787498	79.239998	76.995331	121532000
2020-01-14	79.175003	79.392502	78.042503	78.169998	75.955643	161954400
2020-01-15	77.962502	78.875000	77.387497	77.834999	75.630135	121923600

```python
X[:10, :]
```

```
array([[-0.69829446, -1.        , -1.        , -1.        , -1.        ,
        -1.        , -1.        , -1.        , -1.        , -1.        ],
       [-0.70988941, -0.17490793, -1.        , -1.        , -1.        ,
         -1.        , -1.        , -1.        , -1.        , -1.        ],
       [-0.70047851, -0.1829295 , -0.17490793, -1.        , -1.        ,
         -1.        , -1.        , -1.        , -1.        , -1.        ],
       [-0.7060773 , -0.17641889, -0.1829295 , -0.17490793, -1.        ,
         -1.        , -1.        , -1.        , -1.        , -1.        ],
       [-0.68701725, -0.18029223, -0.17641889, -0.1829295 , -0.17490793,
         -1.        , -1.        , -1.        , -1.        , -1.        ],
       [-0.66144494, -0.16710617, -0.18029223, -0.17641889, -0.1829295 ,
         -0.17490793, -1.        , -1.        , -1.        , -1.        ],
       [-0.65866542, -0.14941483, -0.16710617, -0.18029223, -0.17641889,
         -0.1829295 , -0.17490793, -1.        , -1.        , -1.        ],
       [-0.63233864, -0.14749191, -0.14941483, -0.16710617, -0.18029223,
         -0.17641889, -0.1829295 , -0.17490793, -1.        , -1.        ],
       [-0.64933389, -0.12927861, -0.14749191, -0.14941483, -0.16710617,
         -0.18029223, -0.17641889, -0.1829295 , -0.17490793, -1.        ],
       [-0.65465482, -0.1410362 , -0.12927861, -0.14749191, -0.14941483,
         -0.16710617, -0.18029223, -0.17641889, -0.1829295 , -0.17490793]])
```

Figure D: Design of LSTM Neural Network

```
# Hyperparam, dropout prob
p = 0.2
hidden_units = 50

# Build the LSTM model
model = Sequential([
    LSTM(hidden_units, return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])),
    Dropout(p),
    LSTM(hidden_units, return_sequences=False),
    Dropout(p),
    Dense(25),
    Dense(1)
])
```

Figure E: Training of LSTM Neural Network

```
# Hyperparameters
learning_rate = 0.001
batch_size = 16
epochs = 100

optimizer = Adam(learning_rate=learning_rate)
# Compile the model
model.compile(optimizer=optimizer, loss='mean_squared_error', metrics=['accuracy'])
# Train the model
history = model.fit(X_train, y_train, batch_size=batch_size, epochs=epochs, validation_data=(X_val, y_val), verbose=1)
```

Figure F: Prediction and Forecasting Using LSTM Network

```
# Make predictions
predictions = model.predict(X_test)
predictions = scaler.inverse_transform(np.concatenate((np.zeros((predictions.shape[0], n_lags)), predictions), axis=1))[:, -1]
```

```
def forecast_predictions_new(model, data, sequence_length, forecast_length, scaler):
    predictions = []
    last_sequence = data[-n_lags:]  # Take the last n_lags closing prices as the initial input
    for _ in range(forecast_length):
        print(f'Forecast step {_+1}:')
        # Reshape last_sequence to match the model input shape (1, sequence_length, 1)
        last_sequence = last_sequence.reshape((1, sequence_length, 1))
        # Predict the next value
        next_value = model.predict(last_sequence)[0, 0]
        # Append the prediction
        predictions.append(next_value)
        # Update the sequence, rolling it forward
        next_value = np.array([[next_value]])  # Reshape next_value to be 3D: (1, 1, 1)
        last_sequence = np.append(last_sequence[:, 1:, :], next_value.reshape(1, 1, 1), axis=1)
    # Inverse scale the predictions after the loop
    predictions = np.array(predictions).reshape(-1, 1)
    future_predictions = scaler.inverse_transform(np.concatenate((np.zeros((len(predictions), sequence_length)), predictions), axis=1))[:, -1]
    return future_predictions

# Use the model to forecast the next n_future days
forecasted_values = forecast_predictions_new(model, input_seq, sequence_length, forecast_length, scaler)
```