Tom McIlwain
STATS 202
Kaggle Team Name: *Mr. Tomtastic*
August 27, 2021

Final Report

Introduction

At the end of this course, we are focusing on real-world applications of data mining. There are many important applications of data mining, from healthcare to the weather system. The stock market is perhaps one of the most important applications due to the volatile nature of the stock market and its relevance to everyday life. For the final project, we were tasked with predicting opening security ticker prices for 9 days in the future at 5 second intervals for a total of 5040 intervals per day (45,360 predictions total for each ticker) using real-world financial security candlestick data of past security prices for approximately 87 days. At each interval we are given multiple attributes that describe the price throughout the specific interval, and we are asked to predict the opening price. There are many different models that are well-suited for this task, such as ARIMA or Facebook Prophet. In this report, I describe the implementation of the Long Short Term Memory (LSTM) model for time-series forecasting and evaluate the performance of my implementation.

Figure 1: Ticker opening prices for each interval over time.

Selection

When predicting a seemingly chaotic system such as security prices in the stock market, it is important to choose the correct attributes to use to maximize the performance of the model. Given to us for each ticker at each interval is the opening, highest, lowest, closing, and average price for that interval. Given the nature of the task, I thought it would necessary to minimize the complexity of what goes into the model since I thought trying to extract information from four or five attributes would be too computationally expensive and the impact that these attributes have on the final prediction would be negligible compared to the opening price, which is what we are trying to predict. Therefore I decided to

use only the opening price for each interval. In hindsight, I think this was a flaw in my logic that negatively affected the final performance of the model.

Preprocessing

To get the data ready for use, I formatted the data based on the time interval. In other words, each row of the dataset represented an observation from each of the 10 tickers. I added four more features to the end (day, hour of day, minute of hour, and second of minute) so the model would have the capabilities to capture variation depending on the hour or day.

If data was missing, I set the first missing value in that section of missing data to be the mean of the corresponding ticker and perform a random walk as time progressed, increasing or decreasing the price by $0.001 until we reach a known datapoint. Therefore there was no extended period throughout the data that had missing data and the model could be trained using this data. I chose a random walk strategy to simulate data when it was missing because it was a stochastic process that was difficult to predict, similar to that of the actual data.

Transformation

To increase the performance of the model, I used a time series generator in Keras that implements lag variables for the model to look at in addition to the observations given at the current timestamp. I used 12 lag variables, so that at each iteration, the model was looking at data for the current time as well as data from the past minute to predict the next price. I thought that the opening price would be the most useful feature to look at, but I considered transforming the dataset to look at the change in price between intervals or perhaps combine attributes. Data pre-processing took a very long time thus I stuck with a simpler approach to decrease the computational expense and did not transform individual features.

	A	B	C	D	E	F	G	H	I	J	day	hour	minute	second
0	136.01567	104.599024	96.116404	228.246553	120.658038	160.186072	173.319172	86.146967	173.129421	168.79585	0.0	6.0	0.0	0.0
1	136.01667	104.600024	96.117404	228.247553	120.659038	160.187072	173.320172	86.147967	173.130421	168.79685	0.0	6.0	0.0	5.0
2	136.01767	104.601024	96.118404	228.248553	120.660038	160.188072	173.321172	86.148967	173.131421	168.79785	0.0	6.0	0.0	10.0
3	136.01867	104.602024	96.119404	228.249553	120.661038	160.189072	173.322172	86.149967	173.132421	168.79885	0.0	6.0	0.0	15.0
4	136.01767	104.601024	96.118404	228.248553	120.660038	160.188072	173.321172	86.148967	173.131421	168.79785	0.0	6.0	0.0	20.0

Figure 2: First 5 observations in the formatted dataset.

Data Mining

There are many techniques in data mining and machine learning that are used for predicting security prices in the stock market. Some of the simplest include a moving-average model or a linear regression model. More complex time-series forecasting models include ARIMA (Autoregressive Integrated Moving Average), Facebook Prophet, and LSTM (Long Short Term Memory). ARIMA is very useful because it takes into account previous observations and uses a moving average window to predict new values, but has 3 parameters that must be tuned. Prophet was designed by Facebook and is very easy to use, requiring minimal model parameter tuning and data preprocessing. Prophet is good especially for using a large amount of data that has a seasonal pattern but could perform poorly in stock market prices

that do not have much seasonality. LSTMs are one of the most commonly used models that are very effective due to its ability to take into account past observations in the future predictions.

It is this reason that I chose to implement a LSTM model to predict future security prices using past data. LSTMs are an extension of recurrent neural networks, adding long-term memory capabilities to the network. The LSTM architecture consists of three gates (the input gate, output gate, and forget gate) using activation functions that are used to facilitate input and output of a network cell. The input gate controls the input flow into the cell by omitting certain information. The forget gate further omits information and controls the weighting of the inputs that are in the cell. The output gate is used to determine to what extent each input will be used to produce an output. LSTMS have a chain-like structure with repeating cells. They are very useful for including long-term dependencies when making future time-series predictions. Due to the nature of the data and the task at hand, I thought that the LSTM would be the best performing model due to its ability to take into account recent observations while maintaining long-term memory.

To implement the LSTM model, I used the Python deep learning API called Keras, which provides convenient ways to build neural networks. To start the neural network, I defined the model as a Sequential model for stacking of layers. Then I added two layers to the network: a LSTM layer and a dense layer (all neurons from the previous layer provide input to each neuron in the dense layer) with an output of 10 values for the open price of each symbol. Due to the size of the dataset, training the model took a very long time and consequently thorough parameter tuning was not performed. The final LSTM output dimensionality was 50 units. I trained the model using the first 80% of the given data using mean squared error as the loss function and the Adam optimizer with a learning rate of 0.001 which is faster and more efficient than stochastic gradient descent.

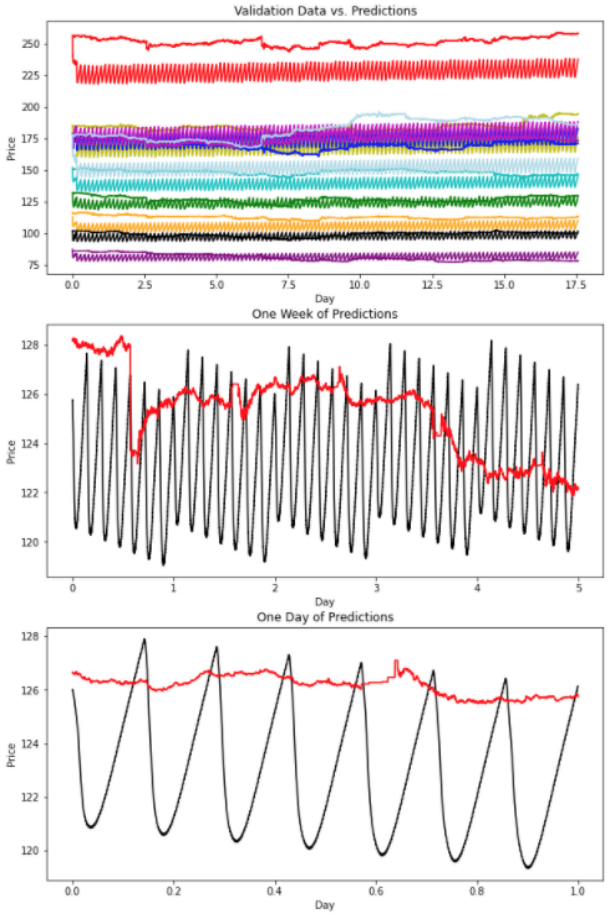
Evaluation

After I trained the model, the next step was evaluation of the performance of the model. I split up the dataset into a training set and a validation set, where the training set contained the first 80% of the data and the validation set contained the last 20%. The validation set was further split up into two periods so that error values could be calculated separately. I implemented the mean squared error calculation according to the equation given in the final project description. I used the trained LSTM model to make predictions for the validation set and plotted the results over all time, one week, and one day.

As you can see in the plots, the predicted lines were not close to the true prices and the LSTM model adopted a sort of sawtooth pattern that oscillated based on the hour and the day. It seems that there was not a pattern found between weeks. Based on the visualized results, I can conclude that my model was not effective at capturing the trend of security prices. The model's predictions oscillate in a controlled predictable manner and there is no variation that is similar to that of the original data. Instead, it seems that the model 'settled' on a value for each symbol and oscillated around that value as time increased based on the minute of the hour and the hour of the day. For each hour, the model has a peak and a dip, which both decrease slightly as the day progresses. The predictions were much different than the validation data so the model was not able to capture the chaotic nature of the original data. This is confirmed when calculating the mean squared error for the two periods. The period 1 MSE was 1811 and the period 2 MSE was 2492, both quite large values. It makes sense that period 2's error would be larger compared to period 1 due to the fact that we are able to start at a more accurate price for period 1. As time progresses, the data varies further from the original starting price. This trend of the stock may change, and

that is difficult to predict with minimal prior data, as is the case for period 2.

However, both of these values are much lower compared to the mean squared errors given from the Kaggle submission. After I predicted using the validation set, I re-trained the model using that data and then predicted 9 days into the future. After submitting these predictions to Kaggle under the team name *Mr. Tomtastic*, the period 1 test MSE was 8797 and the period 2 test MSE was 3184. These values are higher compared to what I calculated using the validation set. Interestingly, the period 2 MSE was lower compared to the period 1 MSE. This may be due to the underlying nature of the true data for that time. Based on these values, my model was not effective at predicting future stock prices. If my model was effective, we would have seen a much lower value for period 1's error. There are many possible changes that could be made to improve the performance of the model, either by transforming data or manipulating the model architecture.



It seems that including independent features for day, hour, minute, and second may not have been a good idea. As the minutes reset at the start of each hour, we see an oscillatory pattern where the prices decrease sharply at the start of the hour and then slowly recover. The same thing occurs as the days change. At the beginning of the day, the price is at its highest, then the peak price per hour decreases as the day progresses and then recovers at the beginning of the next day. If I had changed it so that the day variable represented the day of the week (1-5), then I believe we would have seen a similar oscillatory pattern that repeats every week.

To improve performance, we could include more attributes for the model to use. Rather than just looking at the open price for each time interval, we could look at all four: open, high, low, and close. We could also increase the lag so we are looking further back into the past, although this is at a big expense in computation time. We could transform the attributes to predict the change in price rather than price itself. We could also combine different attributes together. All of these are options that could be used to improve performance that involve the data itself and not the model.

The model could also be manipulated to improve performance. For example, we could increase the width of the LSTM model or stack multiple LSTM layers together to create a deeper neural network. However, these come at the expense of computation time. A different model architecture altogether could be used. The ARIMA model is another commonly used model architecture for tasks like this that requires three parameters to be tuned. Facebook Prophet seems like it would be a great next step to take this since the data format is very similar and we can tune parameters for specific seasonalities such as daily/weekly or even holiday seasonality.

Throughout this experience I've learned a lot about how to handle large amounts of data, how to pre-process it, how transformations affect performance, and much more. It has been a trial-and-error experience where I have had to learn the hard way that training models with a lot of data takes a very long time, and I had to make sure that there are no bugs before I start the long run. There were instances where I attempted to fit a model and it seemed to run but the model did not fit properly and thus I had to start over and refit again. I learned how to implement a complicated model like LSTM using data that I formatted for time-series forecasting. When building these big models, it is very computationally expensive to fit the model and predict with a large dataset, therefore it may be necessary to have some advanced knowledge about specific combinations of parameters and how they will improve performance so that a large grid search is not required to find the optimum parameters. A grid search for hyperparameters would take a very long time with the dataset given and the Keras LSTM model used. This project and class in general has introduced me to using data mining combined with deep learning for predictions of complicated chaotic systems such as stock prices. This skill is incredibly useful for my future career because I want to go into digital healthcare and use data mining techniques to improve the quality of life of people.

Appendix

```
# Imports
from google.colab import files
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from tqdm import tqdm
import warnings
import tensorflow as tf
from keras.preprocessing.sequence import TimeseriesGenerator
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
import keras


# Defining helpful methods
def time_to_idx(time, day):
  day_idx = 5040
  idx = int(int(time[6:]) / 5) + (12 * int(time[3:5])) + (12 * 60 * (int(time[0:2]) - 6)) + (day_idx *
day)
  return int(idx)

def next_time(prev_time):
  next_time = prev_time.copy()
  if prev_time[3] < 55:
    next_time[3] += 5
  elif prev_time[3] == 55:
    next_time[3] = 0
    next_time[2] += 1
  if next_time[2] == 60:
    next_time[2] = 0
    next_time[1] += 1
  if next_time[1] == 13:
    next_time[1] = 6
    next_time[0] += 1
```

```
  return next_time

def time_to_str(time):
  return f'{time[0]:01}-{time[1]:02}:{time[2]:02}:{time[3]:02}'


# Loading data
train_data = pd.read_csv('/content/drive/My Drive/train_data.csv')
train_data.head()


# Preprocessing
columns = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J']

data = pd.DataFrame(np.zeros((time_to_idx('06:00:00', 87), len(columns))), columns=columns)

for i in tqdm(range(train_data.shape[0])):
  row = train_data.iloc[i,:]
  t = row['time']
  d = row['day']
  if t == t and d == d:
    new_idx = time_to_idx(t, d)
    data[row['symbol']][new_idx] = row['open']
data.to_csv('/content/drive/My Drive/full_data.csv')

# Loading in newly formatted data
data = pd.read_csv('/content/drive/My Drive/full_data.csv', index_col=0)


# Changing missing data to random walk
warnings.filterwarnings("ignore")
means = data.mean(axis=0)
if np.where(data.iloc[0,:] == 0)[0] != []:
  loc = np.where(data.iloc[0,:] == 0)
  data.iloc[0, loc[0]] = means[loc[0]]
for i in tqdm(range(data.shape[0])):
  loc = np.where(data.iloc[i,:] == 0)
  move = np.random.choice([-0.001, 0.001], 1)
  data.iloc[i,loc[0]] = data.iloc[i-1,loc[0]] + move
data.head()


# Visualizing opening prices for each ticker over time
data.plot()
plt.title('Security Prices over Time')
plt.xlabel('Observation')
plt.ylabel('Price')


# Adding day, hour, minute, and second features to the dataset
time = [0, 6, 0, 0]
times = pd.DataFrame(np.zeros(shape=(data.shape[0], 4)), columns=['day', 'hour', 'minute', 'second'])
for i in tqdm(range(data.shape[0])):
  times.iloc[i,:] = time
  time = next_time(time)

data['day'] = times.iloc[:,0]
data['hour'] = times.iloc[:,1]
data['minute'] = times.iloc[:,2]
data['second'] = times.iloc[:,3]
```

```
data.head()

# Loading final data
data = pd.read_csv('/content/drive/My Drive/data.csv', index_col=0)


# Train-test split
train_size = 0.8
train_idx = int(train_size * data.shape[0])
targets = data.shift(1, axis=0).iloc[:,:-4]

train_data = tf.convert_to_tensor(np.asarray(data.iloc[:train_idx,:]).astype('float32'))[1:]
train_targets = tf.convert_to_tensor(np.asarray(targets.iloc[:train_idx,:]).astype('float32'))[1:]

test_data = tf.convert_to_tensor(np.asarray(data.iloc[train_idx:,:]).astype('float32'))[1:]
test_targets = tf.convert_to_tensor(np.asarray(targets.iloc[train_idx:,:]).astype('float32'))[1:]


# Generating dataset with lag variables

n_input = 12 # number of lag variables
generator = TimeseriesGenerator(train_data, train_targets, n_input, batch_size=1)


# Implementing the LSTM
model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(None, 14)))
model.add(Dense(10))
model.compile(optimizer='adam', loss='mse')

# Fitting and saving the model
model.fit_generator(generator, epochs=1)
model.save('/content/drive/My Drive/lstm_model_2')

# Loading model
model = keras.models.load_model('/content/drive/My Drive/lstm_model_2')

# Getting predictions for validation data
def predict(num_predictions, pred_list):
  time = [0, 6, 0, 0]
  times = np.zeros(shape=(num_predictions,4))
  for i in range(num_predictions):
      times[i,:] = time
      time = next_time(time)
  times = tf.expand_dims(tf.convert_to_tensor(times.astype('float32')), 1)
  time = times[0,:]
  for i in tqdm(range(num_predictions)):
    X = pred_list[-n_input:,:]
    X = tf.expand_dims(X, 0)
    pred = model.predict(X)
    pred = tf.concat([pred, time], 1)
    pred_list = tf.concat([pred_list, pred], 0)
    if i+1 < num_predictions:
      time = times[i+1,:]
  pred_list = pred_list[n_input-1:]
  return pred_list

n_input = 12
num_predictions = test_data.shape[0]
```

```
pred_list = train_data[-n_input:,:]
pred_list = predict(num_predictions, pred_list)
np.save('/content/drive/My Drive/val_predictions', pred_list)

# Visualizing the predictions
pred_list = np.load('/content/drive/My Drive/val_predictions.npy')
colors = ['c', 'orange', 'k', 'r', 'g', 'y', 'b', 'purple', 'm', 'lightblue']
predictions = np.asarray(pred_list)[1:,:-4]
time = range(predictions.shape[0])
true = np.asarray(test_data)[:,:-4]

day = 5040

# Plotting all
plt.subplots(3,1, figsize=(10,16))
plt.subplot(3,1,1)
plt.title('Validation Data vs. Predictions')
plt.xlabel('Day')
plt.ylabel('Price')
time = np.arange(0, (1+time[-1])/day, 1/day)
for i in range(10):
  plt.plot(time, predictions[:,i], color=colors[i])
  plt.plot(time, true[:,i], color=colors[i])

# Plotting a week
symbol = 4
plt.subplot(3,1,2)
plt.title('One Week of Predictions')
plt.xlabel('Day')
plt.ylabel('Price')
plt.plot(time[:day*5], predictions[day*2:day*7,symbol], colors[2])
plt.plot(time[:day*5], true[day*2:day*7,symbol], colors[3])

# Plotting a day
plt.subplot(3,1,3)
plt.title('One Day of Predictions')
plt.xlabel('Day')
plt.ylabel('Price')
plt.plot(time[:day], predictions[day*4:day*5,symbol], colors[2])
plt.plot(time[:day], true[day*4:day*5,symbol], colors[3])

# Manual Calculation of Validation MSE
num_days = (1 + pred_list.shape[0])/day
days_p1 = int(np.ceil(num_days/2))
days_p2 = num_days - days_p1

pred = pred_list[1:,:-4]
test = test_targets

# period 1
avg_sums = []
for day_num in range(1,days_p1+1):
  p1_t = test[day*day_num:day*(day_num+1)]
  p1_p = pred[day*day_num:day*(day_num+1)]
  p1_res = np.square(p1_t - p1_p)
  p1_sum = np.sum(np.sum(p1_res))
  p1_avg_sum = p1_sum / day
  avg_sums.append(p1_avg_sum)
p1_mse = np.mean(avg_sums)
print('Period 1 MSE:', np.around(p1_mse, 3))
```

```
# period 2
avg_sums = []
for day_num in range(days_p1,int(num_days)+1):
  if day_num == int(num_days):
    p2_t = test[day*day_num:]
    p2_p = pred[day*day_num:]
  else:
    p2_t = test[day*day_num:day*(day_num+1)]
    p2_p = pred[day*day_num:day*(day_num+1)]
  p2_res = np.square(p2_t - p2_p)
  p2_sum = np.sum(np.sum(p1_res))
  p2_avg_sum = p2_sum / day
  avg_sums.append(p2_avg_sum)
p1_mse = np.sum(avg_sums) / days_p2
print('Period 2 MSE:', np.around(p2_mse, 3))

# Refitting with validation data
n_input = 12
generator = TimeseriesGenerator(test_data, test_targets, n_input, batch_size=1)
model.fit_generator(generator, epochs=1)
model.save('/content/drive/My Drive/lstm_model_3')

# Generating the predictions for the next 9 days
num_predictions = time_to_idx('06:00:00', 9)
pred_list = test_data[-n_input:]
pred_list = predict(num_predictions, pred_list)
np.save('/content/drive/My Drive/good_predictions', pred_list)

# Saving predictions to file in the proper format for Kaggle submission
symbols = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J']
from tqdm import tqdm

predictions = pd.DataFrame(np.zeros(shape=(num_predictions*10, 2)), columns=['id', 'open'])
datetime = [0, 6, 0, 0]
for i in tqdm(range(len(pred_list))):
  pred = pred_list[i]
  string = time_to_str(datetime)
  for j, symbol in enumerate(symbols):
    id = f'{symbol}-{string}'
    predictions.iloc[i*10 + j,:] = np.append(id, pred[j])
  datetime = next_time(datetime)

predictions.to_csv('/content/drive/My Drive/actual_predictions.csv', index=False)
```