Stat 453 Project Report

Group 6

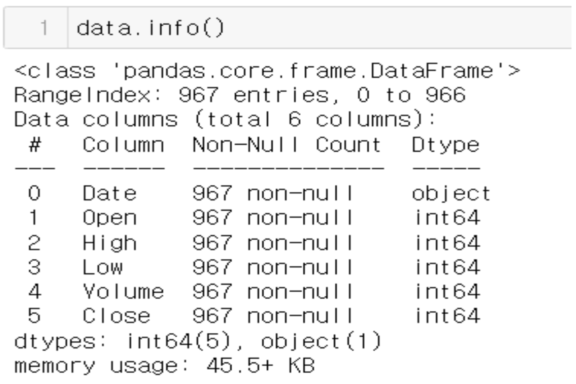
Jinyoung Han, Gyuho Shim and Shuwei Liu

Introduction/Background

Nowadays, it has been common for people non-expertise on stocks or even having no knowledge of economics, to invest their money on stock markets. Some believe that the attractiveness of stock that brings in so many people into the market is its volatility, and unpredictableness. However, the truth is, that there are a variety of factors involved in the fluctuation of the stock that clouds the pattern. For human eyes, all the factors and its interaction can not be easily caught while deep learning can be much faster and suitable to catch such patterns.

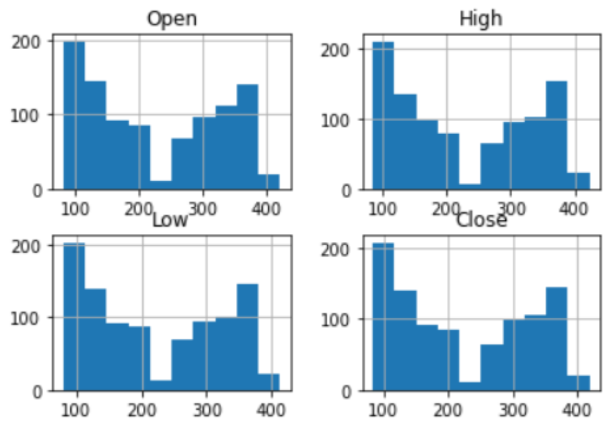
Our motivation for the topic is to enhance prediction accuracy of the companies' future value in order to maximize the efficiency of investment. For the efficiency of the research, narrowing down to one specific company's stock and applying our Recurrent Neural Network-based methodology seemed to be the adequate starting point. Netflix data was chosen based on the considerations of popularity, accessibility, credibility, and dynamicity of the dataset. As mentioned, stock price is a very volatile indicator which makes it more attractive as a data. After testing our methods on such particular data, it can later be expanded to a whole range of companies' stock.

Data



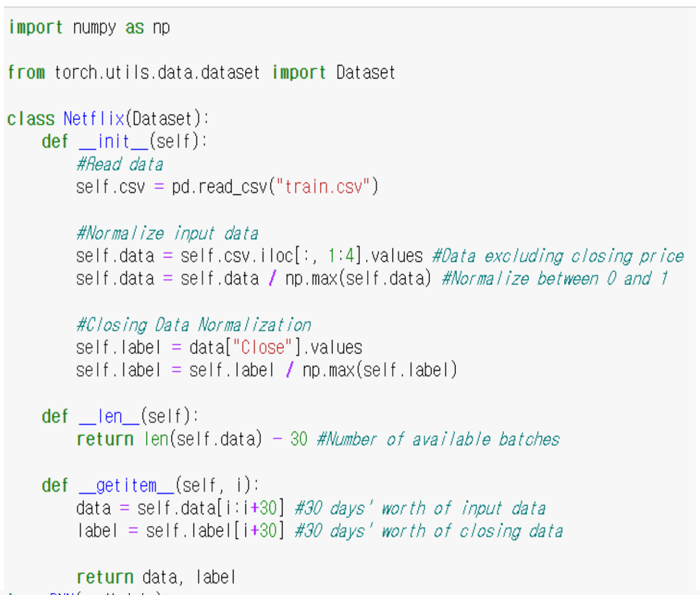
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 967 entries, 0 to 966
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    967 non-null    object
 1   Open    967 non-null    int64
 2   High    967 non-null    int64
 3   Low     967 non-null    int64
 4   Volume  967 non-null    int64
 5   Close   967 non-null    int64
dtypes: int64(5), object(1)
memory usage: 45.5+ KB
```

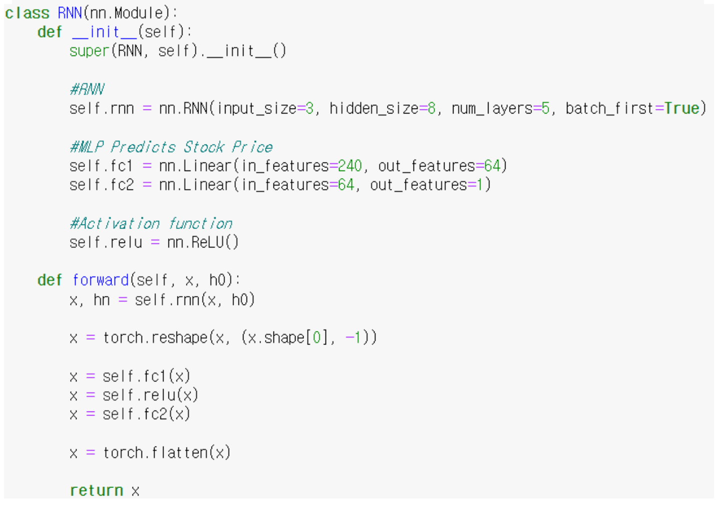


The Netflix stock price data was extracted from Kaggle and the dataset consists of 6 columns and 967 rows. To give a brief elaboration on the 6 columns, the first column is the ‘date’ column which ranges from 2015 to 2019. Second is the ‘open’, which is the opening price when trading begins on the given day. Third is ‘High’, which is the maximum of all of the accumulated stock prices traded during that time interval. Fourth is ‘Low’, similarly, the minimum of all of the accumulated stock prices traded during that time interval. Lastly, ‘close’ is the last price traded before the trading gets closed that day. All of the columns enumerated are crucial due to its trait that shows how a stock obtains varying values in a single day. Inputting them into the RNN model will strengthen the training process.

Methods/Work Process

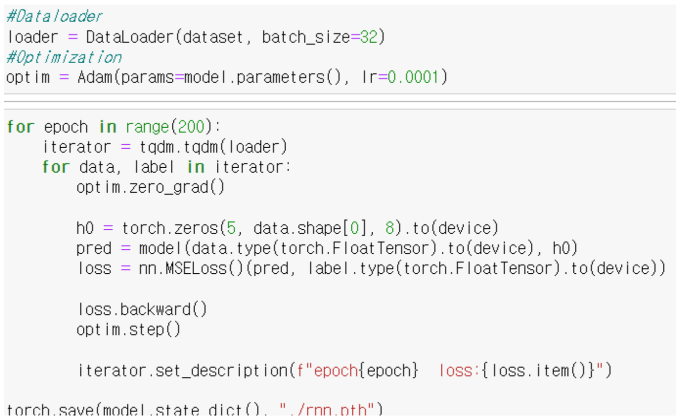


```python
import numpy as np

from torch.utils.data.dataset import Dataset

class Netflix(Dataset):
    def __init__(self):
        #Read data
        self.csv = pd.read_csv("train.csv")

        #Normalize input data
        self.data = self.csv.iloc[:, 1:4].values #Data excluding closing price
        self.data = self.data / np.max(self.data) #Normalize between 0 and 1

        #Closing Data Normalization
        self.label = data["Close"].values
        self.label = self.label / np.max(self.label)

    def __len__(self):
        return len(self.data) - 30 #Number of available batches

    def __getitem__(self, i):
        data = self.data[i:i+30] #30 days' worth of input data
        label = self.label[i+30] #30 days' worth of closing data

        return data, label
```



```python
class RNN(nn.Module):
    def __init__(self):
        super(RNN, self).__init__()

        #RNN
        self.rnn = nn.RNN(input_size=3, hidden_size=8, num_layers=5, batch_first=True)

        #MLP Predicts Stock Price
        self.fc1 = nn.Linear(in_features=240, out_features=64)
        self.fc2 = nn.Linear(in_features=64, out_features=1)

        #Activation function
        self.relu = nn.ReLU()

    def forward(self, x, h0):
        x, hn = self.rnn(x, h0)

        x = torch.reshape(x, (x.shape[0], -1))

        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)

        x = torch.flatten(x)

        return x
```

Since RNN can receive multiple point-in-time inputs, the opening price, the highest price, and the lowest price were grouped together in the input data, and the closing price of the day was viewed as one point in the closing data. Here, 30 days worth of data was taken as batch units. The Netflix() class provides data for learning. The __init__() function initializes the dataset, and the __getitem_() function retrieves a specific element. Since the data range is from 100 to 400, we normalized all the data to be between 0 and 1. The __init()_function returns the number of available batches. We subtracted 30 days from the total dataset length to calculate the number of available batches. Next, we built the RNN model. input_size is the number of features in the input tensor. We have the opening price, the highest price, the lowest price, so its value is 3. num_layers means the number of RNN layers to be used. We used 5 because if we use too many layers, there will be a gradient loss problem that causes the gradient to be zero, or a gradient explosion problem that radiates to infinity. Batch_first means that the batch dimension comes forward the most. Since the opening price, the highest price, and the lowest price were combined for 30 days, the input tensor was stacked by the batch size of the tensor with the shape (30, 3). Next, we created MLP layers that predict stock prices using features extracted from the RNN layer. The activation function was defined using ReLU. x returns the hidden state of the last RNN layer. Results from the RNN layer incorporated two dimensions into one dimension because the time series information, batch size, and extracted features were all distributed in different dimensions. Finally, we predicted the closing price using the MLP layer. The closing price was expressed as a single number. Therefore, the shape of the classifier output was (number of batches, 1). We converted it into a one-dimensional vector for easy conversion into a list.



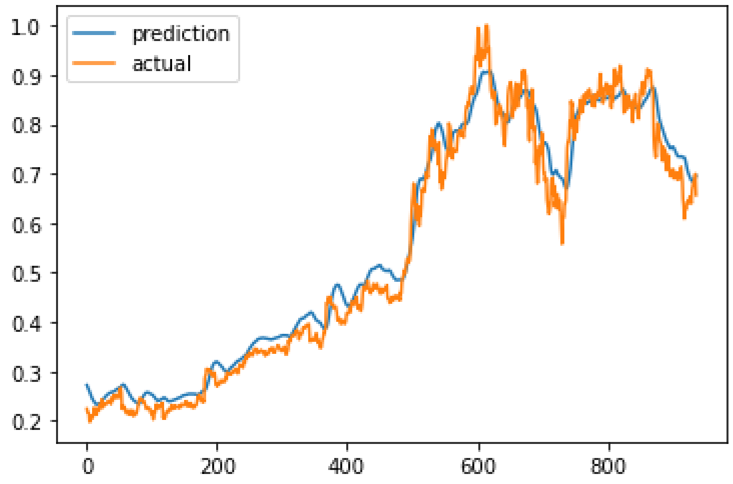
```
#Dataloader
loader = DataLoader(dataset, batch_size=32)
#Optimization
optim = Adam(params=model.parameters(), lr=0.0001)
```

```
for epoch in range(200):
    iterator = tqdm.tqdm(loader)
    for data, label in iterator:
        optim.zero_grad()

        h0 = torch.zeros(5, data.shape[0], 8).to(device)
        pred = model(data.type(torch.FloatTensor).to(device), h0)
        loss = nn.MSELoss()(pred, label.type(torch.FloatTensor).to(device))

        loss.backward()
        optim.step()

        iterator.set_description(f"epoch{epoch}  loss:{loss.item()}")

torch.save(model.state_dict(), "./rnn.pth")
```

This model is a regression problem because it predicts the closing price for the next day. Therefore, we used the mean square error as a loss function. The batch size for learning the model was specified as 32, and the learning rate was set to 0.0001. The epoch was set to 200. The initial hidden state was set to a tensor with all elements consisting of zero. The shape of the hidden layer is (the number of layers, batch size, and output dimensions of the RNN). We defined the input tensor and the initial hidden state and then entered it into the model to receive the results.

We compared the actual data with the model predictions and calculated the error. After calculating the error, the optimization was carried out and the learning was completed.

Analysis



We tested the mean squared error (MSE) between the predicted values of our model and the actual values. The output MSE is 0.0016, which is a relatively small value, indicating that our model performed well in predicting the stock prices. In fact, since we limited the data between \$100 and \$400 to be between zero and one, so even if the average loss is increased by 100 times, we will approximately get an error of only \$1. This demonstrates the model's high accuracy and predictive power on Netflix data.

Conclusion

Our study aimed to explore the use of RNN for predicting stock prices of Netflix, with a focus on predicting the closing price for the next day. To achieve this goal, we used multiple input features, including the opening price, highest price, and lowest price, and grouped them together in the input data. We then treated the closing price as a single point in the closing data. We used 30 days of data as batch units and normalized the data to be between 0 and 1.We built our RNN model with 5 layers, with batch_first set to True. We used MLP layers to predict the stock prices using features extracted from the RNN layer, with ReLU used as the activation function. The model was optimized using mean square error as the loss function, with a batch size of 32, a learning rate of 0.0001, and an epoch of 200. We initialized the hidden state to zero and trained the model by comparing actual data with model predictions and optimizing for error.

After optimizing the model and training the hyperparameters, we tested the MSE between the predicted values of our model and the actual values to evaluate the performance of our model. The results indicate that the model performs well with high accuracy and predictive power.

Overall, the results of this study demonstrate the effectiveness of RNN for predicting stock prices, and provide a foundation for future research in this area.

Future Work

For future work, we plan to generalize our model since it performed well on the Netflix dataset. First, we will collect data from more companies to expand our dataset. Given that the factors that affect a company's stock prices differ across different industries, we will conduct cross-industry validation to test whether our model is still effective in different industries. Additionally, as different features are relevant for different industries or companies, we will conduct research to identify key variables that could improve the predictive power of the model and then select different feature sets for different industries or companies. Finally, we can incorporate external data sources into the model, such as news articles, financial reports, or social media sentiment, which may have an impact on stock prices.