Udacity Capstone Project Report

Yifei Tong

August 2020

1 Project Introduction

Predicting the stock price has been a problem that many mathematicians and traders try to solve. Although the mathematical model used to describe the stock market is a random stochastic model, news and events that influence stock traders' behavior can still cause movement of one stock. This main focus of this project is to investigate how a financial news article can affect its related stocks.

Since I currently do not have ways to smartly find company names and match their stock symbols in a news article, I decide to narrow down the scope of this project a bit for the purpose of meeting the deadline of this course. Instead of scanning through all financial news articles in the data set, this project will select only the news articles mentioning four technology companies, namely Google, Amazon, Facebook, and Microsoft. Then the project will perform regression analysis using Machine Learning models.

2 Problem Statement

The goal of this project is to develop linear regressions model respectively for four different technologies companies, which are Google, Amazon, Facebook, and Microsoft. Detailed steps include:

1. Download data set from Kaggle - US Financial News Articles.
2. Extract useful information from the downloaded data set such as news article, and published date.
3. Select only the news articles mentioning the four technology companies.
4. Process and prepare news articles for training.
5. Develop customized PyTorch LSTM regression models
6. Train, deploy and evaluate PyTorch LSTM regression models on 4 technology stocks

7. Train, deploy and evaluate linear regression models on 4 technology stocks The final results of the project will be the 4 better regression models, one for each of the 4 technology stocks.

3 Evaluation Metric

In total, two evaluation metrics are used in the process of development: mean squared error (MSE) and mean absolute error (MAE).

MSE is used during the training process as the loss function for the gradient descent algorithms of neural network and linear regression. Given the true value y, the predicted value $\hat{y}$ by the regression model, and n number of training examples, MSE is defined by the equation below:

$$MSE = \frac{1}{n}\sum_{i=0}^{n}(y - \hat{y})^2$$

This metric is used during the training process because the square nature of the metric amplifies the effect of extremely large errors which are then punished more by the optimization algorithm.

When evaluating the models, MAE is used. MAE is defined as:

$$MAE = \frac{1}{n}\sum_{i=0}^{n}|y - \hat{y}|$$

MAE gives us a more straight forward representation of the error made in the percentage change of stock prices.

4 Data Collection and Exploration

4.1 Data Collection

The data set I work on for this project is downloaded from a Kaggle data page titled "US Financial News Articles". From this data page, I was able to download more than 300,000 US financial news articles. From these articles, I found about 5000 articles mentioning Google, 6000 articles mentioning Amazon, 11000 articles mentioning Facebook, and 2000 articles mentioning Microsoft.

I put these articles into a dictionary along with their respective published dates. Based on these dates, I used pandas-datareader package to read historical stock data and calculate the percentage price change of each company's stock on the date a related news articles was published. These data sets are saved in a pickle file and uploaded to a S3 bucket on my AWS account.

4.2 Data Exploration and Analysis

It is essential to first understand the data we are modelling with. In this case, we want to understand the distribution of our y variable and length of our articles.

4.2.1 Distribution of y variable

Historically, the distribution of stock returns is close to but more fat-tailed than a normal distribution. The key feature is that the peak of the distribution will be higher than that of the normal distribution. In order to verify that our sampled stock returns form a similar distribution, we plot a histogram for each of the stock returns distribution.

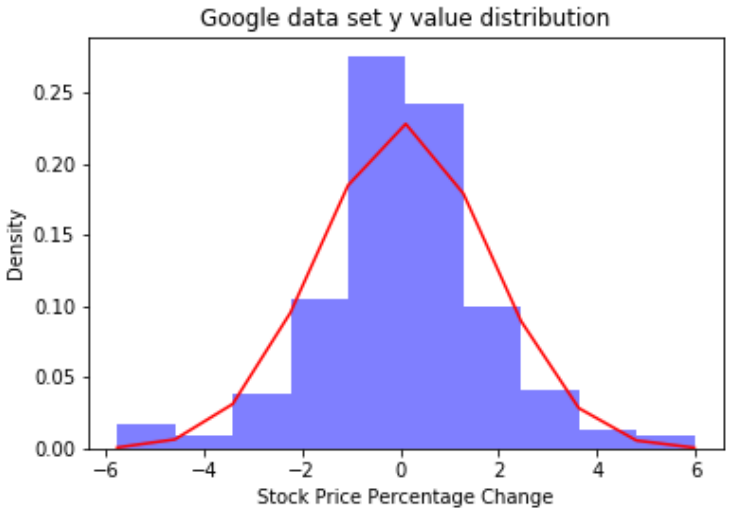

Figure 1: Distribution of y variable (Google)

Figure 2: Distribution of y variable (Amazon)

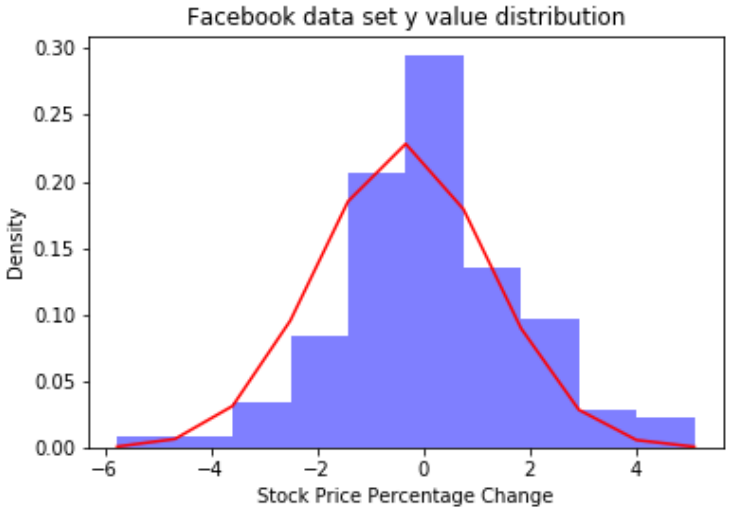

Figure 3: Distribution of y variable (Facebook)

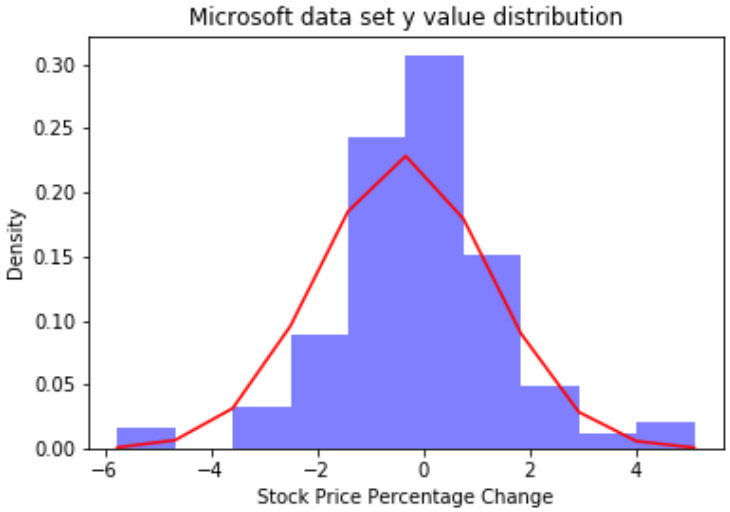

Figure 4: Distribution of y variable (Microsoft)

As we can see in the four figures above, the daily stock return distributions of all four stocks look similar and all have fatter tails and higher peaks than a normal distribution. This means that our daily stock return data sets are sampled appropriately and have similar distributions to historical stock return in general.

However, this may not be what we want for the project's purpose. Ideally, we want the stock return values to be more spread out into the positive and negative territory instead of being centered around 0 due to the effect of financial news articles. The distributions suggest that such effects are limited. The reason may be that the effects of financial news on stock market are relatively short-term. Taking the daily stock return gives stock market the space to self-adjust and return to a normal return level. A better way to collect the data would be to calculate the percentage change of stocks in the 5 to 30 minutes after the published time of news articles. Unfortunately, for this project, I don't have to resources to acquire stock data with that level of precision.

4.2.2 Exploration of Article Length

When exploring the length of news articles, I first compute the average length of news articles.

Company	Article Length (words)
Google	730.55
Amazon	638.17
Facebook	703.33
Microsoft	736.24

Based on these values, I chose a pad length of 500 when vectorizing the articles into numerical inputs for the machine learning models.

Another aspect about the article length I investigated was its value as a feature. A simple correlation test reveals that article lengths are not highly correlated with stock price movement. In fact, the correlations are very weak.

Company	Correlation Coefficient between Article Length and Price Movement
Google	0.0159
Amazon	-0.002
Facebook	-0.0186
Microsoft	0.0232

As shown in the table above, the correlation coefficients are close to 0 for all companies, indicating a weak linear relationship between the article length and price movement. Therefore, when modelling with regular linear regression, the article length will not be considered as a feature.

5 Development Stage

5.1 Data Processing

In order to train machine learning models using our articles, we need to first convert our textual data into sequences of numerical data. The method I adopt in this project is to create a dictionary of significant vocabularies in the articles and encode every word in articles using its index in the dictionary.

5.1.1 Lower Case

Before creating this dictionary, I need to ensure that same words or words with the same meaning have the same representations across all articles. In order to achieve this, I first converted all words to lower case so that same words in upper case and lower case look the same.

5.1.2 Removing Stop Words

After converting all words to lower case, I removed all the stop words from the articles. Stop words are words that are so common in articles that they convey no significant meanings. Examples of English stop words include "i", "you", and "this". Removing these stop words increases the chance that the words in the dictionary we create provide significant information for the model to learn whether an article is positive or negative.

5.1.3 Stemming Words

Another step I took in the data processing stage is stemming the words in the articles. Stemming the words allows the adjective, verb, and noun of the same stem to be recognized as the same word. Grouping these words together provides more space in the dictionary for words of other stem.

5.1.4 Train/Test Data Split

Splitting the data set into a training set and a test set is important for developing a machine learning model. I utilized sklearn's train_test_split function to handle the splitting. I also further split the training data set into a training data set and a validation data set. The validation data set will be used during LSTM modelling to tune hyperparameters.

5.1.5 Building Dictionary

In building a dictionary for a training data set, I hope the dictionary would capture the words that provide most valuable information in the finance realm. After removing stop words from the corpus, the words that appear most frequently should be the most used vocabulary in typical financial news articles.

Implementation: I selected the 5000 most frequently used words in the news articles and encoded them with indices 2-5001 (0 is saved for NOWORD and 1 is saved for words not included in the dictionaries). These indices then become our numerical input in the article vectors. One dictionary is built for one stock in order to make sure the dictionary is more specialized for the company it is constructed for. Code:

5.1.6 Vectorize Articles

Once the dictionary is constructed, vectorizing articles is as simple as iterating through each word in the articles and replace an actual word with its dictionary encoding.

5.1.7 Save Data

Finally, we vectorized all sentences in train, test, and validation data. Three types of data are saved separately in three csv files for each technology stock. With the previously saved pickle files for dictionaries, 16 files in total are saved for later used.

5.2 LSTM Modeling

Two main steps of my LSTM modeling process are development of model and training codes as well as hyperparameter tuning.

5.2.1 Model Definition

Key components of the LSTM regression model:

1. Embeddings of a fixed dictionary
2. A multi-layer LSTM
3. A linear layer

Code:

```
import torch.nn as nn

class LSTMRegressor(nn.Module):
    """
    LSTM network that we use to perform regression on financial news data
    """

    def __init__(self, embedding_dim, hidden_dim, vocab_size, num_layers=1):
        """
```

```
        Model initialization: initializing layers
        """
        super(LSTMRegressor, self).__init__()

        self.embedding = nn.Embedding(vocab_size, embedding_dim, padding_idx = 0
        self.lstm = nn.LSTM(input_size=embedding_dim,
            hidden_size=hidden_dim, num_layers=num_layers)
        self.linear = nn.Linear(in_features = hidden_dim, out_features = 1)

        self.word_dict = None

    def forward(self, x):
        """
        perform a forward propagation with this LSTM network
        """
        x = x.t()
        lengths = x[0,:]
        reviews = x[1:,:]
        embeds = self.embedding(reviews)
        lstm_out, _ = self.lstm(embeds)
        output = self.linear(lstm_out)
        output = output[lengths-1, range(len(lengths))]

        return output.squeeze()
```

The number of layers of LSTM is made as a configurable hyperparameter to be tuned later.

5.2.2 Training Code

When training an LSTM regressor, I choose an Adam optimizer and the MSE loss function. The general process of training is forward propagation and backward propagation in batches.

Code:

```
optimizer = optim.Adam(model.parameters(), lr=args.learning_rate)
loss_fn = nn.MSELoss()
def train(model, train_loader, valid_loader, epochs, optimizer, loss_fn, device)
    """
    This function trains the LSTM with a training data loader from batches of tr
    """
    total_loss = 0
    for epoch in range(1, epochs + 1):
```

```
        model.train()
        total_loss = 0

        for batch in train_loader:

            batch_X, batch_y = batch
            batch_X = batch_X.to(device)
            batch_y = batch_y.to(device)

            output = model(batch_X)
            loss = loss_fn(output, batch_y)
            loss.backward()
            optimizer.step()

            total_loss += loss.data.item()
        print("Epoch: {}, Loss: {}".format(epoch, total_loss / len(train_loader)
    print("Loss: {}".format(total_loss / len(train_loader)))
    #calculate validation
    total_valid_loss = 0
    for valid_batch in valid_loader:
        valid_batch_X, valid_batch_y = valid_batch
        valid_batch_X = valid_batch_X.to(device)
        valid_batch_y = valid_batch_y.to(device)

        valid_output = model(valid_batch_X)
        valid_loss = loss_fn(valid_output, valid_batch_y)
        total_valid_loss += valid_loss.data.item()

    print("Validation Loss: {}".format(total_valid_loss / len(valid_loader)))
```

5.2.3 Hyperparameter Tuning

I utilitzed the AWS SageMaker's hyperparameter tuning functionalities to perform my hyperparameter tuning for LSTM models. The hyperparameters I chose to tune include number of epochs of training (range 10-20), number of LSTM layers (range 1-4), and learning rate (range 0.001-0.01). The best training jobs are displayed in the table below:

Company	Epoch	LSTM Layer	Learning Rate	Validation Loss
Google	20	4	0.0017	3.0656
Amazon	20	4	0.0013	3.1768
Facebook	20	4	0.0031	2.7916
Microsoft	20	4	0.001	2.2336

As we can see from the table above, all best training jobs share several commonalities. Firstly, the number of epochs are large. Secondly, all use a very deep LSTM (4 layers of LSTM). Finally, the learninga rate are all small and in the order of 0.001. Due the limitation of resources, I was not able to tune hyperparameters beyond these ranges but the hyperparameters displayed here indicate that this is a rather complicated problem to model and requires a large number of epochs and a deep LSTM network to achieve the best results. Since epochs and LSTM layers are all at the max value they can achieve, increasing these two parameters may achieve even lower validation loss values.

5.3 Linear Regression Modeling

For linear regression modeling, I use the AWS SageMaker built-in linear learner to train the models. Key hyperparameters are set as the following:

Hyperparameter	Value
predictor_type	regressor
loss	squared_loss
epochs	15

Example linear learner setup code:

```
google_linear = LinearLearner(role=role,
                              train_instance_count=1,
                              train_instance_type='ml.c4.xlarge',
                              predictor_type='regressor',
                              loss='squared_loss',
                              output_path='s3://{}/{}/{}'.format(
                              sagemaker_session.default_bucket(),
                              prefix, model_name),
                              sagemaker_session=sagemaker_session,
                              epochs=15)
```

6 Evaluation

In evaluation phase, I used mean absolute error (MAE) to compare the predicted stock price change with the actual ones. Results are shown below:

Model	MAE
Google - LSTM	1.3031
Amazon - LSTM	1.3594
Facebook - LSTM	1.7246
Microsoft - LSTM	1.1214
Google - Linear Regression	1.4336
Amazon - Linear Regression	1.4095
Facebook - Linear Regression	1.3688
Microsoft - Linear Regression	1.4508

Except for Facebook, all other LSTM models produced a better performance than the linear regression models. This also meets our expectations since a complex task like textual sentiment analysis requires a more sophisticated algorithm to model. However, an error of more than 1% is still very large when predicting a stock market. In the next section, I will discuss possible causes for such inaccuracy and some ideas for improvement.

7 Conclusion

As shown in the previous section, the sentiment analysis based regression modeling on financial news articles does not provide a very accurate prediction of stock movement. I can think of a few possible reasons:

- Data is not collected appropriately. As I have mentioned in the data exploration section, the effect of a financial news article on the stock market can be very short. Collecting daily stock return as the y variable may not accurately reflect such effect since the noise brought by the stock market's self-adjustment can be overwhelming.
- What's more, news mentioning the company name of a technology stocks may not always be news about that company. For example, Facebook can be mentioned in a news articles as a source of information. Including these news articles may hinder the learning process for machine learning models.
- When tuning hyperparameters, I limited the search range due to the restraint of resources. If I could expand the search range of different paramters, I could potentially get to a better model.

In response to the above problems, here are some ideas for future improvement of this model.

- Find a way to collect historical stock data with finer time scale, such as minutes, even seconds. Instead of collecting daily stock return as y variable, compute the percentage change of stock in 5 minutes or 30 minutes after the financial news is published.
- Collect financial news from a more credible source where news can be categorized into different companies.

8 Reference

US Financial News Articles - https://www.kaggle.com/jeet2016/us-financial-news-articles

Textual data preprocessing for LSTM with help from https://github.com/udacity/sagemaker-deployment