Stock Price Prediction With RNN's and NLP

By: Eduardo Osorio

Objective

Objective:

The objective of this study is to see how user **sentiment** towards a particular stock affects its future price. We will be using data from **twitter/reddit** for user **sentiment** and **Yahoo Finance** for historical stock price data.

Data

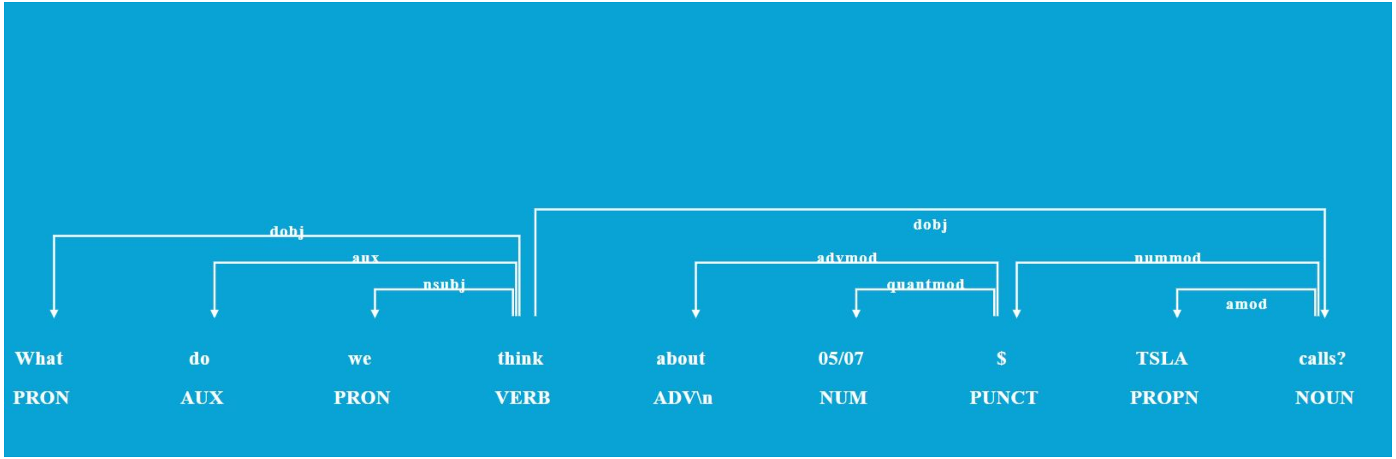
Data:

The data used consisted of **24 months** worth of historical **Tesla** stock price data which accounted for **505 dates** (since the stock market does not work on weekends) and about **7k+ posts** from reddit and twitter.

EDA

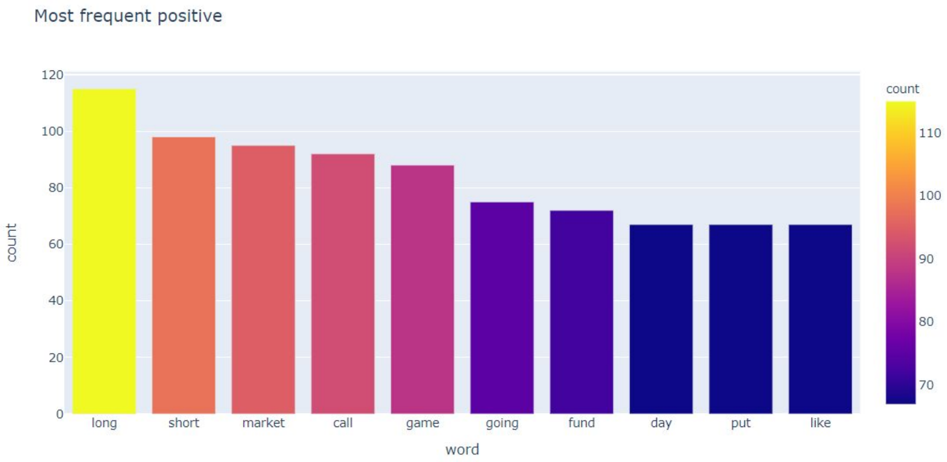
EDA:

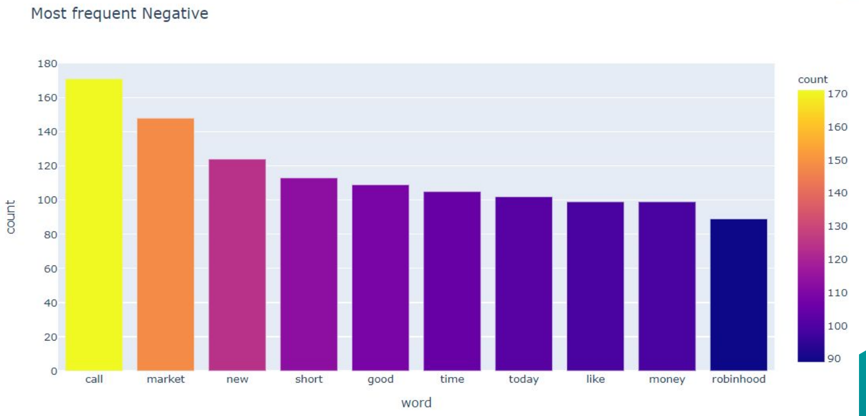
In order to detect user **sentiment** we have to first teach the model how essentially read. To do this, I used spaCy's Textblob pipeline component. With Textblob, we can **part of speech** tag and find **sentiment** analysis with textual data.



EDA Continued:

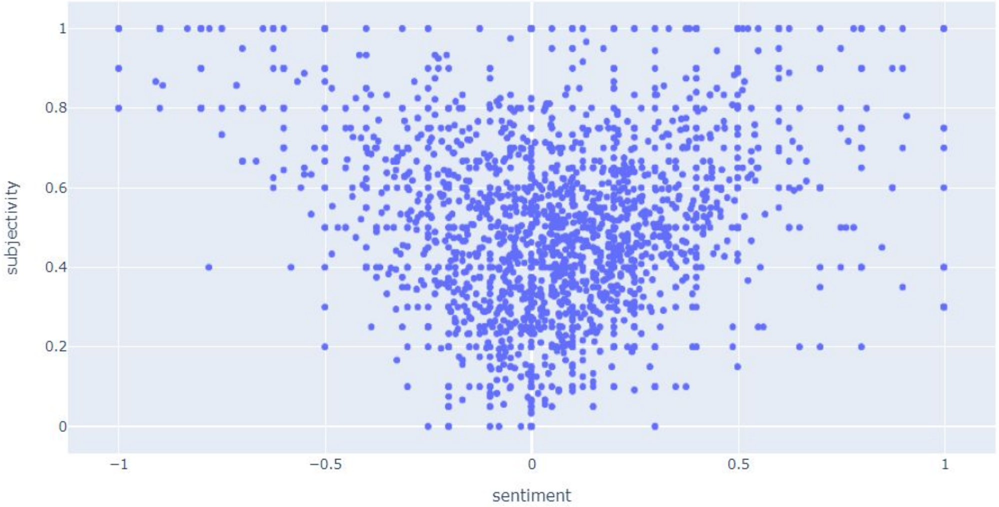
After the reddit data was **tokenized** and **tagged**, I was able to pull the sentiment and subjectivity from each post. With this information we are able to see if a post leans positive or negative.

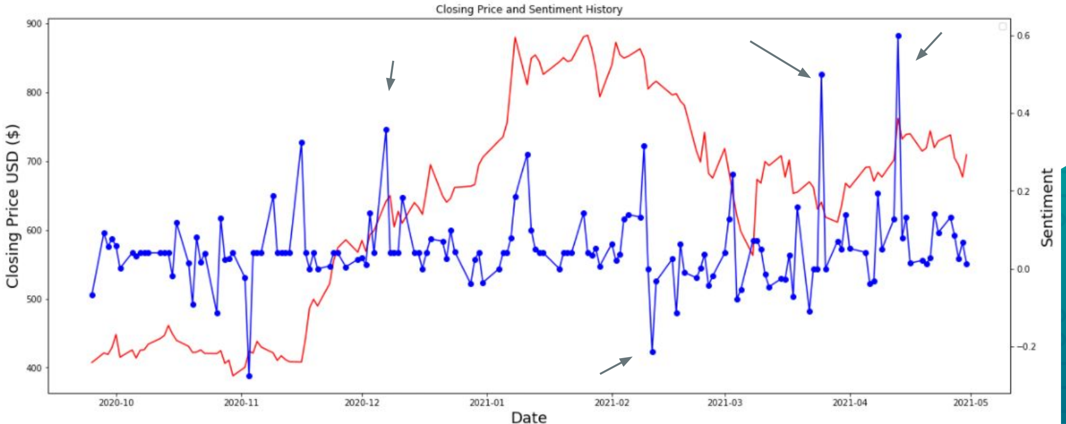




EDA: Continued:

Here we can see how **sentiment** compares to (**subjectivity**) which is another feature used to determine if the post is opinion or factual. The model is determining the post to be (1 = more **subjective**, 0 = less **subjective**). We can see that the **less subjective** the tweets, the **more sentiment neutral**. Below is sentiment to closing price. We can see that major spikes in **sentiment**(blue line) seem to indicate if the price going up or down in the near future.





Model

Base Model:

The data used for the model was a combination of both the reddit/twitter post data and historic stock price. To make dataframe applicable to the model, I first had to **group** all the post by the date and extract the **average value** for **subjectivity** and **sentiment**. This created a dataframe with just **closing price**, **sentiment** and **subjectivity**. Then I needed to create the **target** column which in this case was tomorrows closing price. After having the data in a serviceable manner, I could then feed it into the **LSTM RNN** model(**Long-Short-Term-Memory Recurrent Neural Network**). The main difference between an **LSTM** and a regular **RNN** is that, **LSTM remember** the important data and pass it on down the sequence to make predictions. **RNN**'s, however tend to **forget** if a sequence is long enough due to the vanishing gradient problem.

Base Model Continued:

For the base model, I used a simple architecture with 2 **LSTM** layers and 1 **Dense** layer. The base model was fitted on a **batch size** of 1 and with 100 **epochs**. The results were as followed:

```
13/13 [==============================] - 0s 3ms/step - loss: 5.7300e-04 - mean_absolute_percentage_error: 13011.9111
Training Loss: 0.000573
Training MAPE: 1.3e+04
----------
4/4 [==============================] - 0s 3ms/step - loss: 0.0546 - mean_absolute_percentage_error: 15.3176
Test Loss: 0.0546
Test MAPE: 15.3
RMSE: 141.54787328211893
```

Final Model:

For the final model, I tried different combinations of **layers**, including **dropout layers**, **L2 regularization** and **early stopping**. I also increased/decreased the number of **epochs** and nothing seemed to make a significant difference. The only parameter that made a noticeable difference was increasing the **batch size**.

Final Model Countinued:

For the fina; model, I used a simple architecture with 2 **LSTM** layers, 3 **Dense** layers, **batch size** of 20 and 100 **epochs**. The results were as followed:

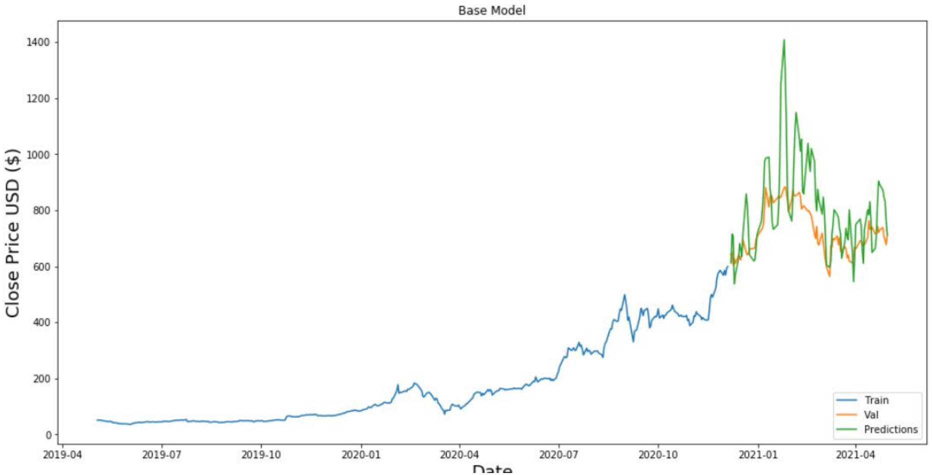
```
13/13 [==============================] - 0s 4ms/step - loss: 7.0885e-04 - mean_absolute_percentage_error: 2057.1589
Training Loss: 0.000709
Training MAPE: 2.06e+03
----------
4/4 [==============================] - 0s 4ms/step - loss: 0.0064 - mean_absolute_percentage_error: 5.6810
Test Loss: 0.00638
Test MAPE: 5.68
RMSE: 48.41230817504435
```

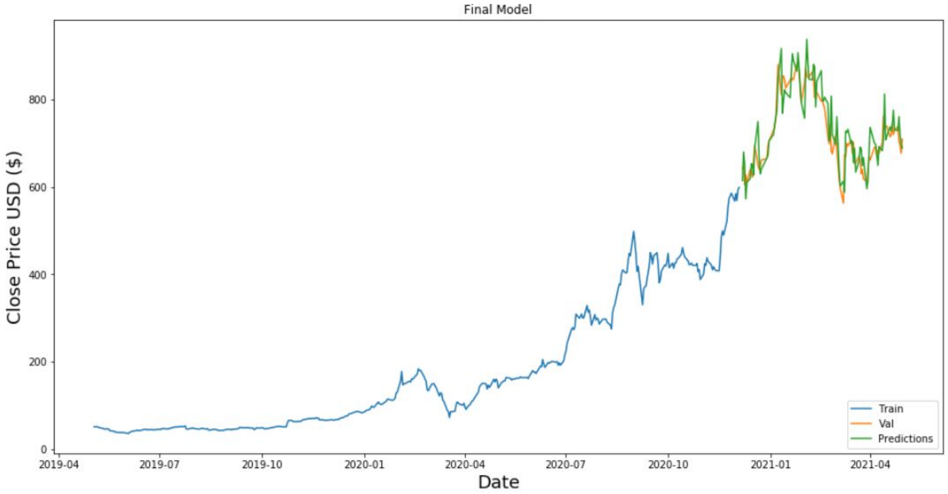
Results

Results:

Compared to the base model, I was able to lower the **Test Mean absolute squared error** rate by **250**%. **Test RMSE** dropped from **141.54** to **48.41**. The **training MAPE** also seems to be astronomically higher then the **test MAPE** but that is due to the fact the model is trained to predict future values .

Results Continued:





Summary/ Conclusion:

- **LSTM**'s work very well on forecasting time series data.
- Sentiment can signal if there's going to be a significant decrease or increase in the near future.

SWOT ANALYSIS

STRENGTHS

The model is able to remember better then a regular RNN and learn more from the data .

WEAKNESSES

Lack of data due to API limitations

S W O T

Functionalizing will make the model run more efficiently. Also increasing data will increase the accuracy

OPPORTUNITIES

Different languages might trip up the sentiment analysis

THREATS

Next Steps:

1. Finish Functionalizing to make more user friendly
2. Add more data sources to the web scraper
3. Create a frontend
4. Address the language problem from web scraping
5. Incorporate more data
6. Add crypto currency functionality

Thank you

Thank you for sitting through my presentation!

- Sources Used:
  - Presentation template by SlidesCarnival
  - https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21
  - APmonitor.com's youtube channel
  - spaCy.io