Predicting the Price of Bitcoin Using Twitter Sentiment Analysis

Akshay Gupta

Chengzi Ni

Li Chang

To understand how the cryptocurrency market is dependent on the sentiment of the tweets accumulated over a period of three years. The coin we are considering here is Bitcoin since it is the coin which actually influences the rest of the coins in the Cryptocurrency market. We also generated the sentiment labels for all the tweets using Neural Network and Google's NLTK platform.

I. Project introduction

With the sudden boom in the cryptocurrency market, it has been a surprise for a lot of people. Stock markets have a typical trend to them and can be predicted depending upon the actual industry scenario. This is not the case with cryptocurrencies. The only way crypto trends can be analyzed or at least thought of being analyzed is through the use of public platforms where people share their feelings or give their insights. Twitter, Facebook, News articles etc., can help us in understanding certain trends.

II. Business understanding

The use of this project can be many fields, right from finance to the field of sales wherein there are no particular trends for the company to do seasonality forecasting. For now, this project helps the author in understanding how to use time series and helps the reader understand the same.

III. Data gathering

A. Gather twitter data

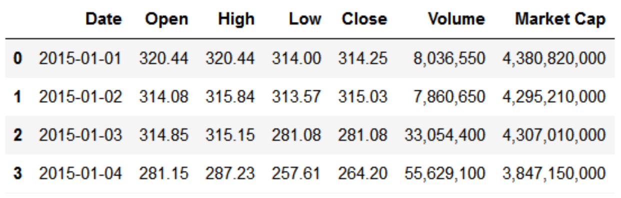
Used selenium, beautiful soup, MongoDB to gather twitter data. The URL [1] can be used to specify the word or the subject we are searching for, the from date and the to date. Once we have our browser reach that page using Selenium, we use Beautiful to scrap the data. We then used JavaScript to scroll to the end of the page. Initially there were a lot of issues when scraping the data since the browser we were working on used to crash and we used to end up losing all the data we scraped. For this, we used MongoDB and made our scraper only scrape the data for two days at a time. This helped us immensely. Even if the browser crashed now, we had our data safely stored in MongoDB. We then extracted the JSON file from MongoDB.

B. Gather price data

We used Selenium and BeautifulSoup for scraping the prices from the web site www.coinmarketcap.com. For this, we first used Selenium to reach the webpage and then used BeautifulSoup to scrape the table that was there on that webpage. The scraped data was subsequently stored in CSV file.

IV. Data Understanding & Preprocessing

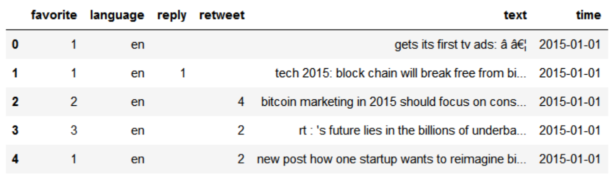
A. Data Understanding



	Date	Open	High	Low	Close	Volume	Market Cap
0	2015-01-01	320.44	320.44	314.00	314.25	8,036,550	4,380,820,000
1	2015-01-02	314.08	315.84	313.57	315.03	7,860,650	4,295,210,000
2	2015-01-03	314.85	315.15	281.08	281.08	33,054,400	4,307,010,000
3	2015-01-04	281.15	287.23	257.61	264.20	55,629,100	3,847,150,000

Figure 1 Price dataset

The first dataset that we have is the price dataset. It has all the information about Bitcoin from 1st January 2015 to 1st January 2018. For the purpose of this study we will only be considering the Close i.e. the closing price and the Volume



	favorite	language	reply	retweet	text	time
0	1	en			gets its first tv ads: â â€¦	2015-01-01
1	1	en	1		tech 2015: block chain will break free from bi...	2015-01-01
2	2	en		4	bitcoin marketing in 2015 should focus on cons...	2015-01-01
3	3	en		2	rt : 's future lies in the billions of underba...	2015-01-01
4	1	en		2	new post how one startup wants to reimagine bi...	2015-01-01

Figure 2 Twitter dataset

The second dataset that we have is the twitter dataset having the number of favorites, language, replies and retweets of the tweet in addition to the tweet itself. This is sorted by time using pandas.

B. Data Preprocessing

For cleaning tweets, we used the Python library ‘re’. It helped us in cleaning majority of the tweets having regular expression as can be see from the first tweet in Figure 2. We also used langdetect package to only keep the English language tweets in the datafile. A function was created around these two packages whose output was a cleaned JSON file in the same format as the one in figure 2.

For sentiment labeler we binarized the 5 sentiments (strong negative, negative, neutral, positive, strong positive) to values from 1 to 5.

C. Decide Sentiment Label

For deciding sentiment label, we decided to use google natural language processing for determining label. There are other famous sentiment analysis APIs such as IBM Watson natural language understanding or Amazon AWS comprehend. We thought the best sentiment analysis API is amazon AWS comprehend, while google provides $300 free for a new account, and it has a python library too. But the implementation of other API such as IBM Watson you need familiarity with its SDK and http request, which makes it difficult to use. By using google NLP, the total time to get 60000 twitters is between 4 and 5 hours. If you use some other API such as ParalelDots from small companies, the time may be tripled. There are also free model that we tried to get the sentiment score, such as **Vader**, the result of which would be shown also in this study.

D. Overfitting handling

We mainly used 4 methods in overfitting handling. Set validation set in all my model we introduced. Add dropout layers (train many separate networks, select part of neuron for training). Shuffling (mainly using recurrent network, change the sequence of input). And during the training process, the validation accuracy may decrease so adjusting the batch size and total number of epochs was also very important.

D. Exploratory Data Analysis & Feature Extraction

We first set out to find the relation between the price and the volume of bitcoin:

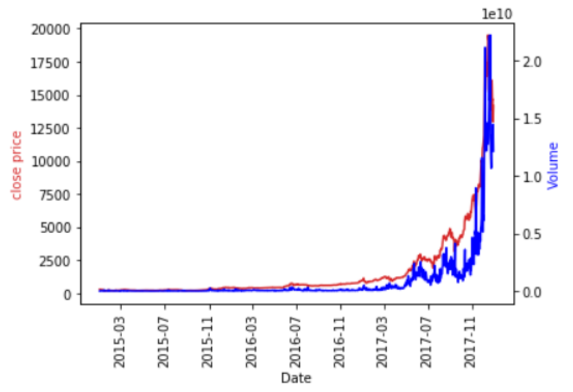


Figure 3 Volume and Closing price

We see from the above graph that they are closely related to each other but that was expected. We then tried to find how the sentiments were related to the closing price of the currency. For sentiments, we used Vader since it gave us a good information about the polarity of the tweets. The graph that we generated was not so useful. We decided that using the exposure (favorites, retweets, likes) would give us some insights. We added the favorites, retweets and likes and multiplied them by the sentiment score. Then we took the mean of the sentiment score for each day. Finally, the standard deviation was calculated for the exposure sentiments. Following was the graph that we generated of sentiments against price.

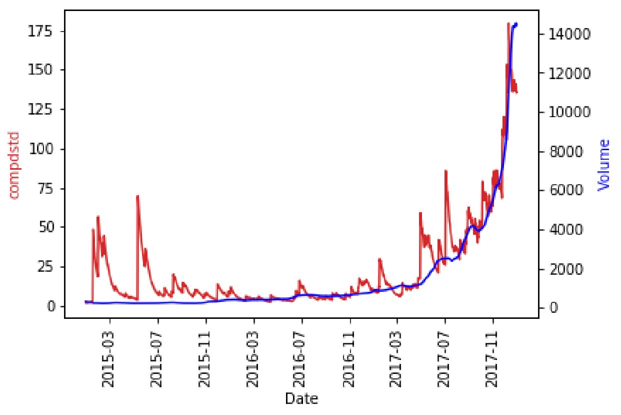


Figure 4 Sentiment vs Price

The sentiment score taken here is the compound score not just the negative or positive or neutral values but a summation of all three normalized. More info about it can be found on the source code of Vader. [2]

We had to clean a lot of robot generated tweets as well, since they had a high exposure and were messing up our analysis.

We tried alternative approaches as well before this by generating a super tweet for each day i.e. aggregating all the tweets for a single day and then assessing the sentiment score, but this did not account for the exposure and hence it was deemed unfit for our model. The result of this can be seen in the python notebook.

Now, since this is a time series analysis, immense time was spent in understanding the trend and **seasonality** and how we can remove them so that they don't affect our model. We first had to find out what kind of autocorrelation did our price dataset had. Autocorrelation plot shows how correlated the variable is to itself with a time lag. Following plot was generated for finding the autocorrelation of price.

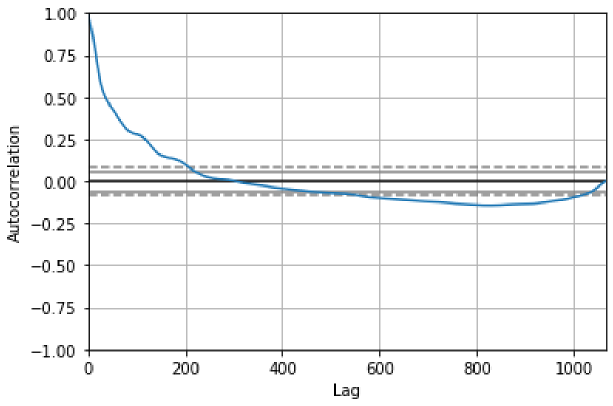


Figure 5 Autocorrelation of price

We see from the above graph that price is significantly autocorrelated with itself up until the 30 days mark after which it is below the 0.5 value. We decided that removing this autocorrelation will also help us in removing the gradual uptrend that we see in the price. To an extent we were successful in doing that. Following was the graph we generated for the price after removing seasonality. [3]

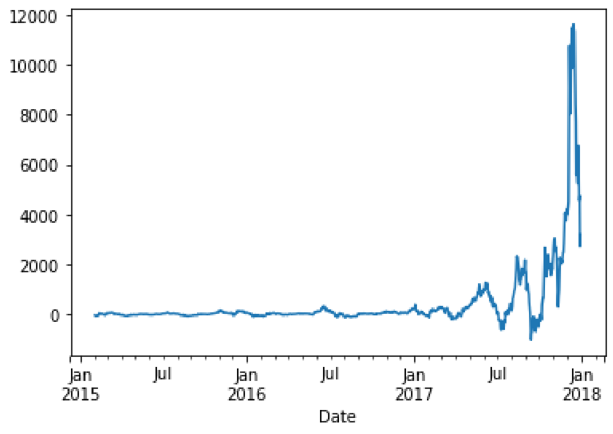


Figure 6 After removing Seasonality

There is still some seasonality in the graph, but we will keep that in the future scope of this project.

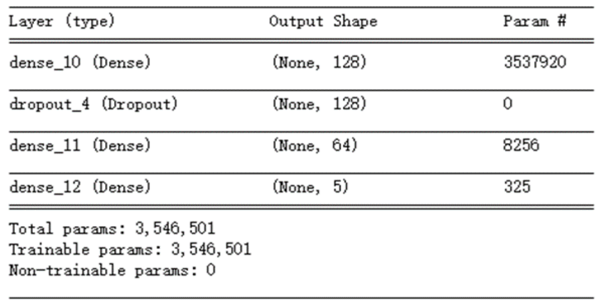
The second approach we used to generate sentiments was using Google's NLTK. For this we first made a model that was able to capture the unigrams, bigrams and trigrams. And subsequently, train our model using Google's NLTK cloud platform.

For feature selection, we built two single word2vec model with stopwords or without stopwords. For getting n-gram words, convolution neural network use conv1d, and if you specify the kernel size from 1 to 3, the model will consider the words from unigram, bigram, trigram.

V. Modelling

We will first build a model to get the proper sentiments from the Google's NLTK platform.

For fully connected neural network with tf-idf vector input the model summary is shown below:



Layer (type)	Output Shape	Param #
dense_10 (Dense)	(None, 128)	3537920
dropout_4 (Dropout)	(None, 128)	0
dense_11 (Dense)	(None, 64)	8256
dense_12 (Dense)	(None, 5)	325

```
Total params: 3,546,501
Trainable params: 3,546,501
Non-trainable params: 0
```

Figure 7 Fully connected neural network Model Summary

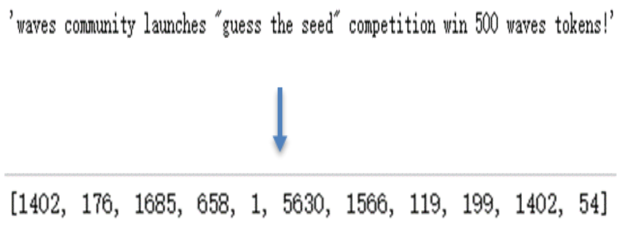
There is only one hidden layer. Because there are not so much input nodes, the total number word is 27,000. We examined our model with 60,000 tweets only. Some big research in this field have their data with millions of unique words, which will need to more hidden layers or convolution layers. Our optimizer and loss function are rmsprop algorithm and binary cross entropy (this is the result by trying other algorithms and we use the same optimizer and loss function in my following models). The test accuracy for this model is 85%. I also use a tf-idf vector without stop words as input, the result is 84.5%.

A. Convolution neural network

The convolution network is used to classify the picture. With invention of word2vec, people can transform the text into “image”. As we all know, a picture is composed by pixels, people use convolution layer to get feature from pixel matrix. By using word vector to stand for word in a text, a text can transform into matrix.

Step1: Tokenize

Use keras tokenizer to replace word by its index number:

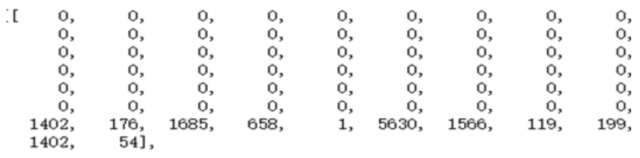


```
'waves community launches "guess the seed" competition win 500 waves tokens!'
```

```
[1402, 176, 1685, 658, 1, 5630, 1566, 119, 199, 1402, 54]
```

Figure 8 Tokenizing using Keras

The longest tweet in our dataset was 61 words, so we padded o in each sentence by 65, which will make every twitter’s length equal to 65.

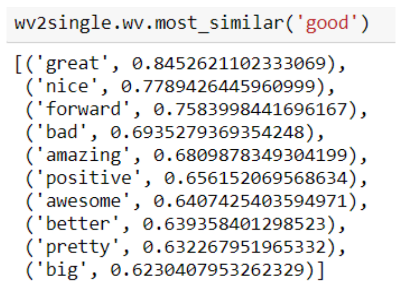


```
:[     0,      0,      0,      0,      0,      0,      0,      0,      0,
       0,      0,      0,      0,      0,      0,      0,      0,      0,
       0,      0,      0,      0,      0,      0,      0,      0,      0,
       0,      0,      0,      0,      0,      0,      0,      0,      0,
       0,      0,      0,      0,      0,      0,      0,      0,      0,
       0,      0,      0,      0,      0,      0,      0,      0,      0,
    1402,    176,   1685,    658,      1,   5630,   1566,    119,    199,
    1402,     54],
```

Figure 9 Padding Sentence

Step 2: build embedding layer

The most important part in convolution neural network sentiment classification is building embedding layer. In this project we used 3 ways to build embedding matrix. Firstly, in keras embedding layers API, we turn words indexes into dense vector of fixed size by not specifying the embedding matrix. These vectors don’t stand for their word meaning, but the test accuracy goes up to 84.49%. The second way is to use word2vec vector to stand for word meaning. We build 2 word2vec model, one is with stop words, another is without it. The no-stop word2vec shows 0.3% improvement in test accuracy while 4% lower in training accuracy. The highest accuracy achieved by our Word2Vec model was 82%. The model even ran well in finding the most similar word:



```
wv2single.wv.most_similar('good')
```

```
[('great', 0.8452621102333069),
 ('nice', 0.7789426445960999),
 ('forward', 0.7583998441696167),
 ('bad', 0.6935279369354248),
 ('amazing', 0.6809878349304199),
 ('positive', 0.656152069568634),
 ('awesome', 0.6407425403594971),
 ('better', 0.639358401298523),
 ('pretty', 0.632267951965332),
 ('big', 0.6230407953262329)]
```

Figure 10 Closest Meaning using Word2Vec

The third method we use is pretrained word vectors. Glove, published by Stanford nlp, have a pretrained word vector dataset which is trained by 2 billion twitters. The accuracy of the model increases to 84.61% by using Glove word vector model.

Step 3: Add convolution layers, pooling layers

In picture classification CNN, the convolution layers and pooling layers is designed to detect some feature in certain area in a picture and reduce the number of parameters training at the same time. In test classification, we can use convolution layer to detect the relationship between the words. There are two important parameters in convolution layers: kernel size and filters. If you specify kernel size = 2, it will combine two words vector with a list of filters, which means you want to find the relationship between two words. So, with kernel size = 3, you can find the relationship between three words. This means you don’t need to build a collocation word vector. Then by using max pooling layers it will keep the biggest number in every two numbers, this will reduce the training and fitting time greatly. After flattening the convolution and max pooling result, the rest is similar to what we introduced in fully connected neural network.

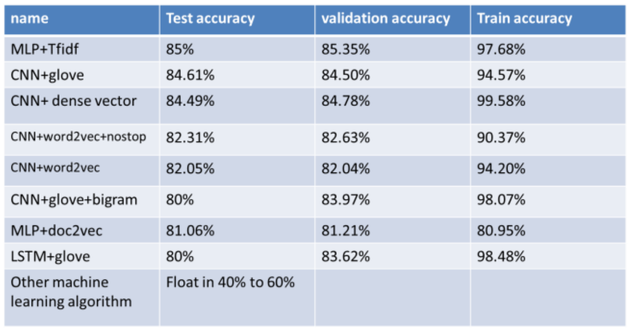
B. Recurrent Neural Network

Another way to handling embedding layers is using recurrent neural network. RNN is a network designed for sequence data. It will keep the parameter trained by last sample input in neural network. So, if you change the order of input, the result will be different. While in the process of RNN training, gradient vanish, and gradient explode happened due to keeping the parameter. People invent long-short term memory (LSTM) to handle this problem. The fitting time is longer than CNN and the accuracy was only 80%. But this doesn’t mean LSTM is bad, LSTM often uses huge datasets, often over 1 gigabyte data will consider use LSTM.

Word2vec and doc2vec model actually is a recurrent neural network auto-encoder. So, using shuffle training (change the input order randomly), the result will differ. By using 20 epoch shuffling, input in this doc vectors in fully connected network, the accuracy goes up to 81.06% compared to 80% without epoch training. We think if we scrape more tweets concerning cryptocurrency tweets, the result could be higher.

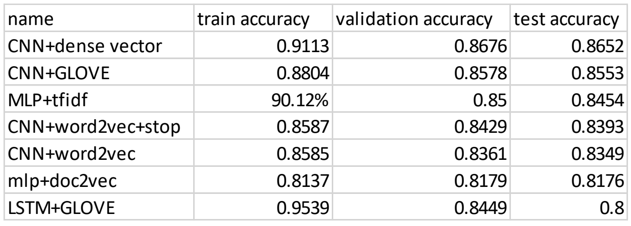
C. Model Summary for Sentiment Labels

This is the table in previous work, after considering new parameter and early stopping, result is different now.



name	Test accuracy	validation accuracy	Train accuracy
MLP+Tfidf	85%	85.35%	97.68%
CNN+glove	84.61%	84.50%	94.57%
CNN+ dense vector	84.49%	84.78%	99.58%
CNN+word2vec+nostop	82.31%	82.63%	90.37%
CNN+word2vec	82.05%	82.04%	94.20%
CNN+glove+bigram	80%	83.97%	98.07%
MLP+doc2vec	81.06%	81.21%	80.95%
LSTM+glove	80%	83.62%	98.48%
Other machine learning algorithm	Float in 40% to 60%		

Figure 11 Old Table



name	train accuracy	validation accuracy	test accuracy
CNN+dense vector	0.9113	0.8676	0.8652
CNN+GLOVE	0.8804	0.8578	0.8553
MLP+tfidf	90.12%	0.85	0.8454
CNN+word2vec+stop	0.8587	0.8429	0.8393
CNN+word2vec	0.8585	0.8361	0.8349
mlp+doc2vec	0.8137	0.8179	0.8176
LSTM+GLOVE	0.9539	0.8449	0.8

Figure 12 New Table

Due to limited data and small number of parameter training, CNN+dense vector performs the best result. Because there is no grid search in keras and for avoiding repeating code, so we just keep the parameter that show the best result we get after many combination of parameters trial and error. Add word2vec not increase the performance of this model much, but the model with word2vec and doc2vec have low overfitting problem. The list of tuned parameters attached below:

In neural network model:

Number of hidden layers, dropout rate, activation function, optimizer, loss function, accuracy metric, batch size, epochs, l1, l2 regularization and max-norm

In word2vec and doc2vec model:

Vector size, window, shuffling and learning rate

This is the confusion matrix from some of our models. Due to the unbalance data, most twitters are in the positive side, my models perform very bad in negative side, so we think that it must perform very good in balance data.

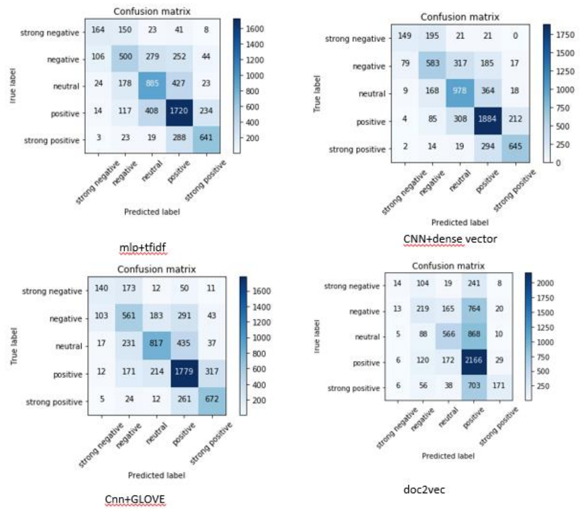
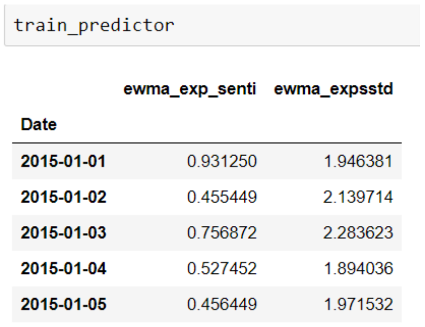


Figure 13 Confusion Matrix

D. Support Vector Machines (SVM):

After creating the model for generating sentiment labels, we will be using Vader for our sentiment analysis. We did not have enough number of tweets for us to use the above stated model. Hence, the decision to use Vader.

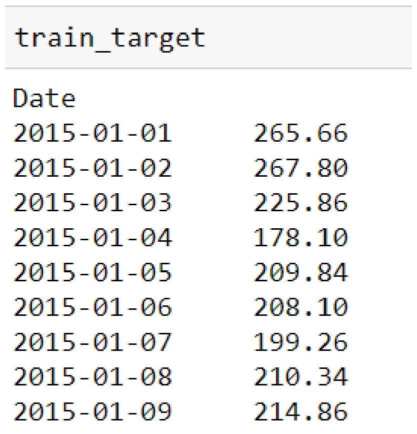
After we know the relation between exposure-sentiment and the market, we can try to build model based on the features. Because our data is a time series data, we split the full data set to training set (2015,1,1 to 2017,10,1) and test set (2017,10,2 to 2017,12,31), and from the EDA we realize that a 10-day lag would suit out model better.



```
train_predictor
```

	ewma_exp_senti	ewma_expsstd
Date		
2015-01-01	0.931250	1.946381
2015-01-02	0.455449	2.139714
2015-01-03	0.756872	2.283623
2015-01-04	0.527452	1.894036
2015-01-05	0.456449	1.971532

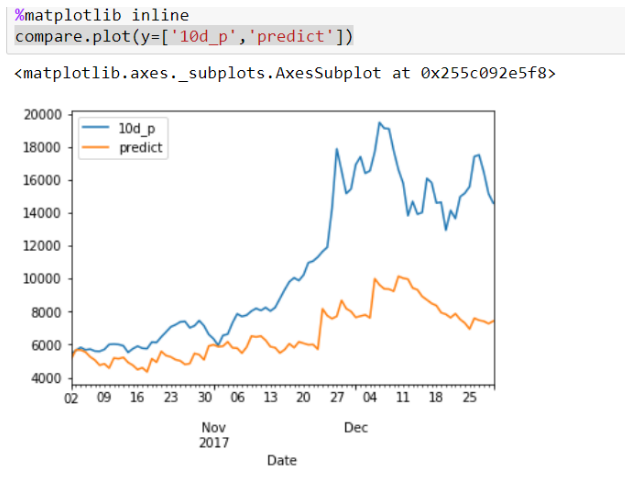
Figure 14 Predictor variable



```
train_target

Date
2015-01-01      265.66
2015-01-02      267.80
2015-01-03      225.86
2015-01-04      178.10
2015-01-05      209.84
2015-01-06      208.10
2015-01-07      199.26
2015-01-08      210.34
2015-01-09      214.86
```

Figure 15 Target Variable (Price)



```
%matplotlib inline
compare.plot(y=['10d_p','predict'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x255c092e5f8>
```

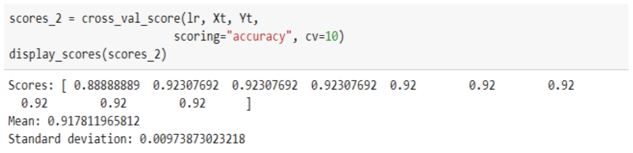
Figure 16 Actual vs Predicted

We see from the above graph that the model is able to predict whether the price is going up or down up but is not able to predict the price closely enough. Since, the data was limited, this is the best this model could do.

D. Logistic Regression

Since we were not able to predict the price as close to the actual price, we decided that we can tell the user of the model when the price is going up or down. For this we used Logistic Regression. For this model, the target was if the close price 10 days later is higher than current close price, it is P and if the close price 10 days later is lower than current close price, it is N.

Doing this, we got an accuracy of 91%. But the real-world use case of such a model is limited.



```
scores_2 = cross_val_score(lr, Xt, Yt,
                           scoring="accuracy", cv=10)
display_scores(scores_2)
```

```
Scores: [ 0.88888889  0.92307692  0.92307692  0.92307692  0.92        0.92        0.92
  0.92        0.92        0.92      ]
Mean: 0.917811965812
Standard deviation: 0.00973873023218
```

Figure 17 Cross-Validation Logistic Regression

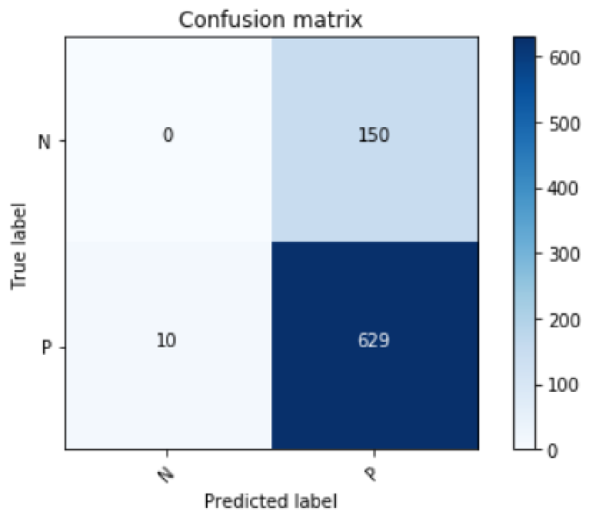


Figure 18 Confusion Matrix

Since this was an imbalanced class i.e. having more Positives than Negatives, a confusion matrix was necessary. We can see that the model is not predicting the N's at all. This is a problem with having less amount of data. For future scope, we would require more number of tweets to perform a successful logistic regression which has a real-world solution to a real-world problem.

VI. Conclusions

We can conclude by saying that, the in-depth EDA helped us in understanding what kind of features would be important for a study of this kind. The scope of this project was limited since the data we could scrape was limited. We were only able to scrape the top tweets and all of them. The challenge here was also to build our own data scraper without using the Twitter API. Although, it was basic, but it got the job done. In future, we would like to get more data i.e. instead of top tweets we would want to gather all the tweets. That also means using the twitter API. This study can be further continued by incorporating the sentiments generated by the Neural Networks since they also take into account word collocations (i.e. Word2Vev).

The one thing that was of immense learning through this project was understanding how time-series operate. We were able to understand through the EDA how the seasonality and autocorrelation have an impact on the final outcome.

Finally, we would want to continue working on this study if time permits. This is an area of growing interest and will have a lasting impact if anyone is able to generate a model with a high accuracy not only in terms of the monetary effects but also behind understanding what makes the cryptocurrency market tick.

References

[1] [Online]. Available: https://twitter.com/search?q=%22%23{}%22%20since%3A{}%20until%3A{}&src=typd.

[2] NLTK. [Online]. Available: https://www.nltk.org/_modules/nltk/sentiment/vader.html.

[3] J. Brownlee, "Machine Learning Mastery," 21 12 2016. [Online]. Available: https://machinelearningmastery.com/time-series-trends-in-python/.

[4] S. Raval, "Github," [Online]. Available: https://github.com/llSourcell/Stock_Market_Prediction/blob/master/Generating%20Different%20Models.ipynb.

[5] S. Maroju. [Online]. Available: https://www.researchgate.net/post/Which_models_are_best_suited_for_time_series_data.

[6] J. Brownlee, "Machine Learning Mastery," [Online]. Available: https://machinelearningmastery.com/autoregression-models-time-series-forecasting-python/.

[7] Abhineet, "Github," [Online]. Available: https://github.com/abhineet1991/Bitcoin-price-prediction-model.

[8] P. Triest, 20 August 2017. [Online]. Available: https://blog.patricktriest.com/analyzing-cryptocurrencies-python/.

[9] C. Bonfield, "Towards Datascience," 10 January 2017. [Online]. Available: https://towardsdatascience.com/a-foray-into-time-series-forecasting-using-ethereum-closing-prices-5ca69dbbd76a.