Nicholas Mohan, Nathan Fenske, Nick Landry, Isaiah Plummer
Syracuse University
May 15, 2021
CIS 400 - Prof. Edmund Yu

Cryptocurrency Projections From Tweet Sentiment Analysis

Abstract

This report will detail our group's efforts in developing a tweet-based prediction model for the hourly percent change in three of the largest cryptocurrencies: Bitcoin, Dogecoin, and Ethereum. All three of these relatively new investment opportunities have risen to extreme market caps in excess of hundreds of millions of dollars, despite suffering massive minute-by-minute booms and busts on a daily basis. Public sentiment via social media has been the main driver behind this extreme volatility -- in this month of May '21 Tesla CEO Elon Musk was able to cause the trillion dollar cryptocurrency Bitcoin to plummet in price by 16% in a matter of hours with a single Tweet [1]. This susceptibility to public discourse is what has driven our team to determine if there exists some reliable form of price prediction in these markets. Using a combination of 'crypto' market data and information from Twitter's Streaming API, we were able to develop a classifier algorithm which can predict the trajectory of a cryptocurrency's price with 71% accuracy.

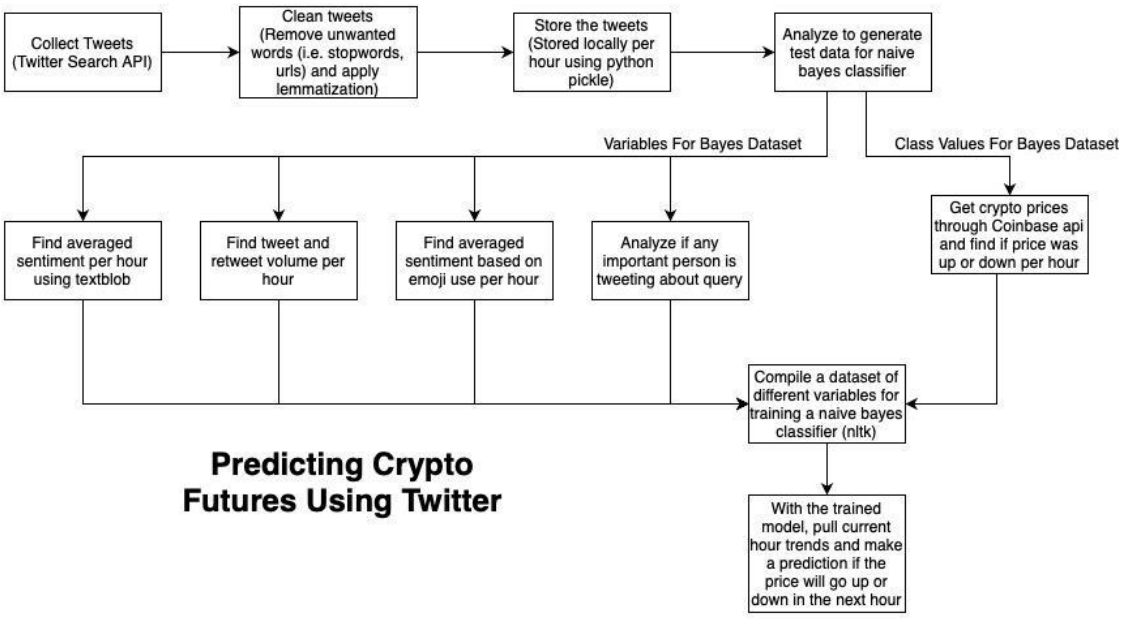
Introduction

As the world continues to become more digitized each and every year, it seems as if no industry is safe from cutting edge disruptive technologies. This is especially true recently within the finance world. Since the introduction of computers, banks and other financial institutions were slow to adopt technologies into their business models. At the time it made very little sense for them to change as the underlying securities were not changing. Based on this it made very little sense for these institutions to change the systems they have relied on for years. This all changed in the last decade as a new investing instrument was introduced into the world. Cryptocurrency broke onto the scene in January 2009 when Bitcoin was first introduced to the public. This new decentralized, blockchain backed internet currency made it easy for everyone, not just large financial institutions to keep their money secure online. While in the first few years there was not much interest outside of hardcore followers, in recent years the popularity around Bitcoin and other cryptocurrencies has skyrocketed. With this popularity spike, came a large demand, and thus price spike for these internet currencies. Unlike traditional stocks and financial investment opportunities, cryptocurrencies were easily accessible by anyone with access to the internet. This made cryptocurrencies a popular investment tool for the average investor. As the popularity and each of cryptocurrencies increased, so did the visibility of many qwerks within this new system. With the main difference between traditional investing being the massive amounts of volatility within the market. With the sheer volume of trades of cryptocurrencies a

day, as well as trading being open twenty four hours a day seven days a week, volatility of cryptocurrencies is very large which creates the opportunity to make money, as well as lose money. This led our team to the idea of trying to create a program that can accurately predict cryptocurrency futures, and thus inform the user of the program on what they should do with their current cryptocurrency portfolio. With cryptocurrencies being popular with everyday average financial traders, we felt an immediate link to another online platform that is known for being tied to this same community; Twitter. With many of the people who buy and sell cryptocurrencies discussing their current thoughts, we felt that by running some kind of sentiment analysis we could determine what the cryptocurrency market would do in the future. So for the project, we decided to create a program that first collects many tweets centered around a certain cryptocurrency. Afterwards, we will analyze these tweets using sentiment analysis to get the current feelings about whatever cryptocurrency we are currently analyzing. By comparing these current feelings to the price of the cryptocurrency, we hoped to be able to find past trends and then accurately predict the future prices of the cryptocurrency. Below we will explain all of the steps and decisions we made in this project. We will then talk about the results that we got and our group's reflections on these results.

Project Structure

For this project, we decided to try and keep the flow of the project as modulated as possible. There were a few key reasons for this. The first being by working in modules each group member could easily work on each section of the project. Once the modules were done it made it much easier to piece the entire project together. The second reason the project was designed like this was for future improvements. If in the future someone wanted to reuse or redesign a section on this project, they would only have to replace a small section and code and



leave the rest of the code alone. This structure made the project easy to design and build out as a team. With this project structure also made our project flow very simple and easy to understand. Below is a flow chart of how our project functions as well as a brief explanation about each part of the project.

The first module of this project has to do with the tweet collection and creation of the initial dataset that we will base our sentiment analysis around. This section of the project is centered around interactions with the Twitter api and the collection of tweets for analysis. For this module of the project there are many different decisions that need to be made, including what tweets to collect, how to collect them, and when they are collected how to store them. Collecting the tweets is an important process as the rest of the project will only be as good as the data that is collected. These decisions should be considered thoroughly and the design decisions we took in this project will be talked about in detail in the collecting tweets section of this paper. After collecting the tweets, the next part of the project comes with de-noising and cleaning this data for processing.

In the tweet cleaning and de-noising section of this project, there are a few key steps that need to be taken to ensure the quality of our prediction at the end of the program. For one, Twitter is notorious for being a quantity heavy platform with poor editorial quality. The short form factor and speed of creation of tweets leads to a very noisy collection of content. Typos and shorthand abbreviations are commonplace, with emojis and nonsensical statements making appearances as well. This means that there is a lot of filtering that needs to be done on our part. In this section there are some decisions that need to be made as well. It is important to establish what types of tokens are useless, while identifying the useful tokens and storing them. For our use case, there were certain tokens that we did not think were relevant to our sentiment analysis, mainly hyperlinks and words that were under two characters. However we did feel that emojis were relevant so we decided to leave them as tokens within our filtered dataset. Other pieces that were filtered out were stopwords, which would interfere with our sentiment analysis, as well as punctuation. The last part about cleaning tweets that should be thought about is bringing all of the tokens to a normal form. This includes changing misspelled tokens and synonyms to words that represent the meaning of the words. This is a process called lemmatization and should be considered when doing any type of sentiment analysis. Once we have this filtered dataset, it is time to start processing the data and generating a final price movement prediction.

The next module of the project is all about creating the variables and class for each entry into our naive bayes dataset. While doing analysis of some of our results, we came across the idea of not just using sentiment analysis to generate our prediction, but considering other factors as well. For example, each set in our naive bayes dataset includes many different variables, not just the sentiment analysis score, as well as an identifying class. These variables are the sentiment analysis variable, an emoji variable, an important person mentioned or involved variable, trade volume variable, and a tweet volume variable. Each of these variables is set with a class which is determined by if the near future cryptocurrency price from the time these variables were created went up or down. These variables are all collected in different ways. Many of them

come from a direct analysis of the filtered dataset that was created in the previous module. The sentiment analysis score comes from textblob, which is a sentiment analysis python library. Finally, the cryptocurrency prices are brought in using the Coinbase Api. From the data gotten from this api analysis can occur to assign each set of variables to the correct class. The selection of these variables was strategically calculated, and more of the thought process behind each variable can be read about in the corresponding section later in this paper. With this set of data, we then moved to creating current predictions using this naive bayes approach and the dataset that we developed.

With the final dataset ready for operation with the bayes classifier, this final module of the project is where the final prediction is created for the future price of the cryptocurrency we are currently exploring. With all of the seven days worth of previous data cleaned and processed for bayes, the last thing that needs to be done is the current data needs to be captured and appropriate variables pulled out of it. In collecting the previous seven days worth of tweets, we also collect all of tweets for the current day. Going back to the previous full hour of tweets, we run this hour through all of the filtering and variable extraction as above. However since we do not know the future price, we cannot label the class and thus this is where prediction takes place. By feeding the variables into the trained classifier, we can generate a percentage that the set of variables will be in the price goes up class or in the price goes down class. By taking the higher percentage we can get a prediction on what we believe the future price of that cryptocurrency will do. With that completes the flow of how the program and prediction process takes place.

Collecting Tweets

Collecting tweets is a crucial part of this project as without it we would not be able to determine the sentiment on the different cryptocurrencies we are trying to determine the futures for. While in the process of collecting the tweets we ran into many different decisions that had to be made. We will dive into the decisions we were faced with, and the reasoning as to why we made the choices that we did. The first decision we had to make was how we wanted to get the actual tweet data from Twitter. For this we had two main options, one being the Twitter search api, and the second being the Twitter streaming api. For this project we were trying to compare past trends to predict what exactly is going to happen in the future. As for this decision it only made sense to go with the Twitter search api. While doing our research, we did find that Twitter streaming api was a better option for collecting tweets that are real time. The reasons for this are the ease of use, as well as the “fire and forget” mentality that can be used for collecting tweets with the streaming api. However, for this project we are looking to find past tweets, and the best way to get these past tweets are through the Twitter search api. This api gives the programmer the option to pass both a date, as well as a max tweet id into the api call. Together these two things allow the programmer to receive all the tweet data from a certain day. This was perfect for our use case, so this api is what we decided to use to collect our tweets. However, this brought up

the next question of how much data do we want to collect, and how far back do we want to look for when we are making our final price increase or decrease prediction.

The next big decision we as a group had to work through was what data do we want to consider for our prediction. We had a few options to consider here. The first being to consider as much data as we could get our hands on. The second option was to consider just the previous day's tweets to make our prediction and only those tweets. The final option was to pick a number between the previous two and consider that many days of tweets. Each of these options have their advantages and disadvantages. The first option one will give us the most amount of data to pull from, and when making predictions like this, it is normally best to consider as much information as you can. However a downside is that collecting and storing all of this data will get expensive in terms of storage, as well as program run time will increase as we will have more data to clean, de-noise, and process. As for the second option, the prediction that is made will be made off of less data, however this will offer a faster prediction with less of a storage cost. This leaves us with the third option of using an in between amount of tweets. What we decided as a team to do was to consider the previous week, or 7 days worth of tweets. This gives us the best of both worlds from the first two options. We will get a larger dataset to make our prediction off of, however we will not have an extremely long runtime for our prediction. There did happen to be one more reason to only consider the previous seven days worth of tweets, and this has more to do with the psychology of crypto-currency traders than with the programmatic issues we had for the other two options. Upon doing research before starting this project, we as a group looked into other academic papers who have attempted similar projects to what we were trying to accomplish. Upon doing research we found many different papers including the paper “The predictive power of public Twitter sentiment for forecasting cryptocurrency prices” by Kraaijeveld and De Smedt, talk about how twitter sentiment is a cause of price fluctuations in cryptocurrencies and these fluctuations are normally caused by sentiment trailing by a short number of days beforehand. As we plan on comparing hourly twitter sentiment data to cryptocurrency prices, we felt that having seven days worth of tweets was a good in between where we would have data to make a prediction, but still would not be spending a large amount of time on processing all of the data. Once we figured out how we were getting the data, as well as how much of the data we were going to collect, we had to decide the queries that we were going to send to the Twitter search api.

Right off the bat, there were some agreed upon parameters we wanted for our data. The first of these being that we wanted to work with english tweets. All of our team members decided that it would be easier to just work in english, a language we are all comfortable with, and so we would filter out any tweets that were not in english. The second agreement between the team was that we wanted to have the ability to not only predict one type of cryptocurrency. After some deliberation, we decided that we will work with three different currently popular cryptocurrencies. These being Bitcoin, Dogecoin, and Ethereum. Once we had those things established, we had to come up with the search terms that we wanted to use to get the tweets centered around these three topics. After some deliberation and playing around with the Twitter

api, we found that keeping the search term as simple as possible would give us the best results. Knowing this we decided to just use the search terms “Bitcoin”, “Dogecoin”, and “Ethereum” respectively. These query terms give us all tweets that include these search terms, and nothing more or else. We found that this is what works best for us. Now that we have made the decision as to what we are going to feed as a query term into the twitter API, we have to make the decision about what data we plan on saving once we get all of the results back.

The next decision that we have to make is about what data we will store once we do have the tweet objects from twitter. When receiving data from the Twitter api we receive back tweet objects. These objects are large JSON like structures that house a multitude of information about the tweet. While in a perfect world we would be able to store all of this information, we have storage restrictions that would limit us from keeping all of this data. Even if we did keep all of this information, it is much more than we need for our study. This leads us to having to decide what information is important for each tweet for this project. The first few obvious fields that we wanted to collect were tweet text, username, as well as time of the tweet. The tweet text is obviously an important field as this is where we will be able to determine the sentiment of the tweet and thus make a final prediction. The username is important because we found that certain users have more impactful tweets then others. We thought that this information may be useful in creating the prediction. The time is an important field as this will allow us to pinpoint exactly when the tweet was created so that we can compare it to the price of the crypto precisely at that time. After we established these fields, we had to decide what other information we wanted from these tweet objects for our prediction. We decided to store two other variables. The first of these variables was the tweet id. This id is important as we need to be able to tell the Twitter api what the max tweet id we want back in our api call, as we will be making many calls to the api. The other variable that we decided to store was the retweet count of the tweet. When collecting tweets we will be filtering out any tweet that is a retweet. Keeping retweets in our dataset would skew our sentiment analysis results as certain tweets will be counted multiple times, so we decided to filter them out. However, we felt that retweet volume would be a helpful metric in our final prediction, so by tracking the number of times that a specific tweet was retweeted we could find this metric out. Now that we figured out the data that we were going to store for each tweet, we had to figure out how we wanted our data to be stored for our tweet dataset.

The final large decision we had to make was how we wanted to store our tweets once we got them from the Twitter api and pulled the information that we wanted from the Twitter tweet objects. Earlier in this project we decided to store our information per hour as cryptocurrencies are notorious for being extremely volatile that can sharply spike or dip at any time. Using an hour by hour approach we can have some granularity on creating that final prediction, while not having to focus on every single tweet. Since this is how the project is set up, we decided to store our data on an hour by hour basis. This will make pulling data for the dataset more logical and easier later on when we are processing this data. As we wanted to do an hour by hour approach and store the data per hour, we had a few options for storing the data. We could set up a database, or we could store the data in files locally. As our team did not have much experience in setting

up databases, we decided to store the information locally. In this class we were all familiar with using python pickle, we decided that each hour we would pickle the python object of all of the tweets from the last hour and dumb it into a pickle file. Every day would have its own folder, with the folder filled with 24 pickle files for each hour of the day. This was a very technologically basic solution, however it was perfect for our use case, very easy to set up, and made interfacing with this collected dataset very simple. With all of these decisions made, we can now talk about how the actual tweet collection works, and how to operate it.

While designing this part of the project, we decided to keep it separate from the prediction program. We want users to be able to first run the data collection, and then once it is finished be able to run the prediction software as both of these operations take a significant amount of time, and separating them felt like the right decision. In the construction of the tweet collection program we first started by utilizing the twitter cookbook that we were given in class. Specifically we used the make_twitter_request() function so that we had the all important rate limiting error detection. The limit for the Twitter search api is 180 api calls in a 15 minute window. Each api call can return at most 100 tweet objects, so this means we can get 18,000 tweets every 15 minutes. We are collecting much more than 18,000 tweets, so it was important to have this rate limiting in the project. With this cookbook function we then created our own function getAllTweetsOnDay(). This function takes in a date, as well as a search query, and then gets all of the tweets for that query from the day that was inputted, and stores these tweets as pickled files as talked about earlier. In the main function of this program, the computer first finds the current day, and creates a list of the previous seven days. It then runs the getAllTweetsOnDay() function for each of these dates. This entire process will run three times for the three different queries for the three different crypto currencies that we are creating predictions for. This tweet collection takes some time as the twitter rate limiting becomes a bottleneck for the program. After this program is run, the user can now go ahead and run the de-noising and prediction parts of the project.

Removing Noise From Dataset

In our python file named ‘bayesV4.py’, there are two places where noise removal is utilized. It’s first implemented in the function ‘tokenFeaturesHour(day,hour)’ where we initially compile and tokenize a specific hour of tweets which will be used in our Bayes classifier as our “token feature set”, which will be discussed in the Bayes section of the report. The second place its used is in the ‘compileTweetTokenFeats(t, f)’ function, which tokenizes and compiles features of a specific hour of tweets which will later be used in comparison to our initial most common tokens that were generated by the ‘tokenFeaturesHour()’ function (also more detailed in the Bayes section).

Noise removal is a crucial part to improving the accuracy of our results because it removes unwanted and, most likely, useless data that could influence the outcome of our program. Specifically, we were looking for words that were not stop words or named entities,

emojis, references to other people's Twitter handles (@), and hashtags to trends (#). Keeping words that could produce a certain sentiment on cryptocurrency is a crucial part of our project. However stop words produce little sentiment and keeping them within our dataset would not be beneficial at all. Emojis are very important when it concerns twitter, almost acting as another language. Due to the unique ways people can communicate over Twitter with emojis, we decided to keep them in our dataset since emojis can potentially convey sentimental sentences by using only a single character. Lastly, '@' and '#' strings will be kept due to the way Twitter uses strings that start with these symbols as a way to reference other users and trends.

The first items that needed to be removed were links and hyperlinks, which by far are the easiest to detect due to a lower density in amount and the tokenized string always starts with 'http'. Usually, tweets that contain these links are very specific in subject and gathering these links would be useless in order to determine the overall sentiment value of a tweet. Once these were removed, we could proceed to filtering out more of the tokens.

To remove stop words, we imported an already existing list of stop words from nltk.corpus and checked the lower case version of tokens to that list. In the same check, we used 'string.punctuation' to remove any punctuation that could be detected. We also clean out any string that has under 2 characters that are not emojis, since these strings are primarily just letters that could cloud up our results in the end. Once these are filtered out of our tokens, we check the list for the types of strings that we want in our dataset. First, the program checks if the string has an '@' or a '#' at the beginning of the string and adds them to our filtered list accordingly. If the token isn't one of these two, it moves on to check if the string is an emoji. In order to perform this check, we imported the package 'emoji' which has a database that contains the english unicodes for all emojis that are used, called 'UNICODE_EMOJI_ENGLISH'. If an emoji in our token list is detected in the list from the 'emoji' package, it will add the token to our filtered list. If the token isn't any of the previously mentioned tokens that we wanted, we check if the token is an actual english word by seeing if it's contained in a list of words from nltk.corpus. If it is, it gets added to our filtered tokens.

After all of these checks, we are left with unique characters and misspelled words in our unfiltered list of tokens. To correct this, we imported a package called 'autocorrect' which has a function called 'Speller' that finds an english word that has the shortest edit distance to the misspelled word. Once it autocorrects the misspelled word, the program adds it to our filtered list. However, there are some inherent flaws that are presented when using the 'Speller' function, especially when it comes to other languages. If the word is a non english word, the 'Speller' will autocorrect it to the closest english word, which affects the tokens that are filtered out. However, we decided to keep it since there are significantly more english words contained in the token list. To top off our filtered list, we remove any named entities before returning it.

Determining Our Data Points

Now that the filtered dataset is created, we have to decide what data we want to use for our project. On initial inspection one could easily just take the sentiment scores from a particular time and classify it using the crypto price at the time slightly after that sentiment score was generated. However when we did initial research for this project we quickly realized that this method was not very effective and that we have a wealth of other data that we will also be collecting that we can use. One paper specifically written by Abraham, Higdon, etc. found that tweet volume has a large correlation with the near future price of cryptocurrencies. With this information we set out looking for different data points that we could use as inputs to positively impact the final prediction. After more research and thinking we came up with many different variables that we settled on using. These variables are sentiment analysis variable, an emoji variable, an important person mentioned or involved variable, trade volume variable, and a tweet volume variable. We will go through and talk about the reasoning behind including each of these variables.

The first variable that we decided for input into the bayes was the sentiment analysis score. After feeding the text into textblob, which is the python sentiment analysis tool we are using, we receive a polarity score. From this polarity score we can determine if the sentiment is positive or negative, and use it as a variable for our bayes. The next data point we decided to use was an emoji/repeated emoji use variable. When visually inspecting tweets talking about cryptocurrencies, we found that many of them include an excessive amount of emoji. Whether it be the rocket ship, the money bag emoji, or simply the smiley face emoji, we felt leaving emoji out of the final prediction would be losing some of the sentiment. That is why we decided not to filter out emoji in our cleaning module, and why we included it as a final data point for our prediction. The next datapoint we are using is an influencer or special influencer tagged datapoint. In cryptocurrency twitter predictions, there are certain people that when they discuss the topic on Twitter, a visible change in the price occurs. A perfect example of this is with Twitter user Elon Musk. Upon visual inspection, everytime Elon Musk tweets about dogecoin, there is an almost immediate spike in price. We wanted to capture this phenomena. When a predetermined high influencer user either tweets or is tagged in a tweet, we created a datapoint to track this and play a role in our final prediction. The next two data points that we decided to include both involve volume. These are the tweet volume, and the trade volume. While pulling data from the Coinbase Api we are able to get cryptocurrency trade volumes, and through our dataset we can get twitter tweet volumes. Going back to the paper mentioned above, volume has found in the past to directly correlate to a future increase in price. Including both of these volumes is advantageous knowledge for our bayes classifier to have to make a more accurate prediction. Once we decided on the data points that we wanted to use, we then were able to get started on creating the bayes classifier that would create our final prediction.

Bayes

Given our team's relative lack of familiarity with the world of machine learning, we felt it best to ensure that we could develop a reasonably sufficient iteration of a classifier before proceeding too far with our project. During lecture, we were given an example of how to use NLTK's naive Bayes classifier to classify textual inputs based on the presence of common tokens. We took this example and morphed into the first version of our classifier, where just as in the provided example, we chose to seek out the presence of highly influential tokens as our initial features. To do so, we compiled and tokenized an entire day's worth of tweets from April 22nd ,which would be used to construct the main "token feature set" (i.e. each token from this list is turned into a variable of the form 'contains(word)'). Values for this feature set were then compiled from another day's worth of tweets, this time from April 27th to make sure the features are applicable to days beyond the 22nd. We acknowledge that a single day is not representative of typical trading, but the purpose of this first iteration was simply to prove that the classifier was successfully operating on a small scale. Despite not having any price-related or sentiment analysis metrics involved in this build, we were able to produce an accuracy level of about 57%. This indicated to us that our hypothesis had potential, and that we were ready to begin development on a more refined classifier.

```
Training...

Accuracy:
0.5562
Most Informative Features
         contains(Black) = True              rise : fall   =     10.8 : 1.0
          contains(Hunt) = True              fall : rise   =      9.0 : 1.0
 contains(coinhuntworld) = True              fall : rise   =      9.0 : 1.0
         contains(vault) = True              fall : rise   =      9.0 : 1.0
          contains(Coin) = True              fall : rise   =      8.5 : 1.0
            contains(BN) = True              fall : rise   =      7.5 : 1.0
   contains(environment) = True              rise : fall   =      7.3 : 1.0
          contains(sir) = True              rise : fall   =      6.5 : 1.0
```

(Above: Output from Build 1 of Bayes)

Our next two iterations revolved around incorporating features which actually provided insight into public sentiment. The token features 'contains(word)' from before were certainly effective in the first attempt at Bayes, but they inform us of any market behaviors besides the subject matter of investor discourse.

First and foremost, we had to improve our tokenization/lemmatization so that our token feature set could include words with maximal emotional indicators. The Python libraries NLTK, autocorrect, emoji, and string were essential to carrying out this stage of development (more detail has been provided in the 'removing noise' section of this paper). Once this was completed, we implemented a small list of data points with more relevance to the price trajectory of a cryptocurrency. Specifically these were;

- Price change and price volume for the last hour
- Tweet volume in the current hour
- Emoji/repeated emoji used
- Influencer tagged/special influencer tagged
- Average sentiment and # retweets in the given tweet

We felt at this point that the token features were still a good inclusion to the classifier, as they had produced a decent accuracy in our initial build. Thus by including new features with more specificity towards market activity, we hoped that this would give us a considerable increase in accuracy from the first iteration. Curious to try out other predictions with the same training set, we also experimented with additional prediction classes during this time; we attempted to classify 5 separate price changes using different thresholds based on % change. This resulted in a dramatic drop in accuracy to about 61%. There were attempts made to resolve this decrease by sampling equally across the prediction classes for the training set, but it soon became apparent that the number of classes was the barrier to a better classifier. Sure enough, with our third build of the main Bayes classifier we were able to see an accuracy of up 67% when predicting two outcome classes.

```
Training...

Accuracy:
0.6483
Most Informative Features
contains(@Bitcoin_K_S_A) = True              C4 : C3     =     20.4 : 1.0
         contains(token) = True              C1 : C4     =     20.2 : 1.0
           contains(due) = True              C1 : C3     =     19.7 : 1.0
           contains(mum) = True              C1 : C3     =     19.7 : 1.0
contains(@dogecoin_rise) = True              C1 : C3     =     19.7 : 1.0
       contains(Capital) = True              C1 : C3     =     19.7 : 1.0
```

(Above: Output for Build 3 of Bayes)

The prior builds' accuracies were pleasing to see, but our team was still unsatisfied with the remaining uncertainty in our algorithm. We agreed that our feature set encapsulated all desired market predictors, so we set out to investigate how we might improve the creation of our training set to improve accuracy. Two main modifications quickly became apparent here. The first change needed to be made to the input weights of our features set. As previously mentioned, there existed 2000 token-based features for collection compared to the 8 non-token features, meaning the latter were being outweighed by a factor of around 250:1. Reducing the amount of token-based features to 500 improved accuracy to 68%. Following this improvement we tried boosting the weights of the non-token features to match that of the non-token ones, again boosting accuracy to 71% at this point.

```
Accuracy:
0.7132
Most Informative Features
      contains(SokuSwap) = True             fall : rise   =      8.9 : 1.0
contains(bestcryptocurrency) = True             fall : rise   =      8.9 : 1.0
          contains(SOKU) = True             fall : rise   =      7.8 : 1.0
          contains(pump) = True             rise : fall   =      4.9 : 1.0
     contains(financial) = True             fall : rise   =      4.7 : 1.0
     contains(beginning) = True             fall : rise   =      4.7 : 1.0
          contains(find) = True             fall : rise   =      4.7 : 1.0
        contains(little) = True             rise : fall   =      4.5 : 1.0
          contains(plan) = True             rise : fall   =      4.5 : 1.0
```

(Above: Output for Final Build of Bayes)

Unfortunately time constraints prevented us from making further improvements to our accuracy scores, but that did not prevent us from envisioning further ways that this classifier could be enhanced. The primary modification which we wished there was time for was the ability to expand the training/test sets to include data across a weeklong span, as a single day of trading and tweeting will never be representative of a 'normal' day. Our computers however were not powerful enough to complete this change before the deadline, as the algorithm takes quite a long time to execute, and we were beginning to encounter memory errors from the sheer size of the set. While we may have not been able to capture the accuracy and results for a full week set, the code which we have developed can be easily modified to operate such a classifier.

Results

The final iteration of our naive Bayes classifier reached a top recorded accuracy level of 71% when predicting hourly rise/falls in Bitcoin price data. Build over build, our team was able to continually improve the collection methods, analysis, and performance of our algorithm to the tune of a 15% increase in accuracy. This final score is still far from where we might have hoped it to be, but having read the findings of other academics in the field this is not unremarkable work. A Stanford paper posted similar results for hourly predictions, albeit receiving scores around five percent more than ourselves [4]. Considering the differences in academic experience and familiarity with machine learning, we are quite comfortable with this disparity. With the numbers that we did end up getting, we believe that there is a vast amount of improvement that can be made in the future. With the large amount of design decisions and user generated content on the internet, there are endless possibilities to bump up the accuracy while only making small changes to the program that we have created for this project. When accomplishing this the information learned while completing this project would be helpful in making these new design decisions.

Conclusion

Throughout this project, the main goal was to create as accurate a prediction system that we could make for different cryptocurrencies. Utilizing the large amount of free user generated real time content on Twitter, as well as sentiment analysis tools that we learned in class, the team had high hopes in creating a working system. Throughout the process there were many different decisions and obstacles along the way. From figuring out just what type of data we wanted to collect, to how we want to store the data, and finally how the data will be processed to create a prediction, there are endless possibilities and combinations for all of these decisions. Using the knowledge we learned not only in class, but throughout our undergraduate studies, we made the decisions that we believed would have the best outcome. In the end we were able to show a 71% accuracy of prediction rate on the training set that we compiled. In our eyes we view this as a complete success. As we discussed before, the extreme volatility and large daily trade volumes

of cryptocurrencies create a very difficult scenario for prediction, and being able to be right 71% of the time shows just how well thought out and executed our prediction plan was. While there are many improvements that can be made to this project that would hopefully further boost the prediction accuracy, the success that we had in the time allowed for this project shows that this area of study has a lot of potential in the near future.

Bibliography

1. Gandel, Stephen. “Why Elon Musk's Bitcoin U-Turn Caused a Crypto Crash.” *CBS News*, CBS Interactive, 13 May 2021, www.cbsnews.com/news/bitcoin-crypto-prices-elon-musk/.
2. Kraaijeveld, Olivier; De Smedt, Johannes; (2020) “The predictive power of public Twitter sentiment for forecasting cryptocurrency prices”, Journal of International Financial Markets, Institutions and Money, Volume 65, 101188, ISSN 1042-4431
3. Abraham, Jethin; Higdon, Daniel; Nelson, John; and Ibarra, Juan (2018) "Cryptocurrency Price Prediction Using Tweet Volumes and Sentiment Analysis," SMU Data Science Review: Vol. 1 : No. 3 , Article 1
4. Colianni, S., Rosales, S.M., Signorotti, M.: Algorithmic trading of cryptocurrency based on twitter sentiment analysis (2015)