Data Science 2

Project A

Live Bitcoin Price and Sentiment Analysis

By:

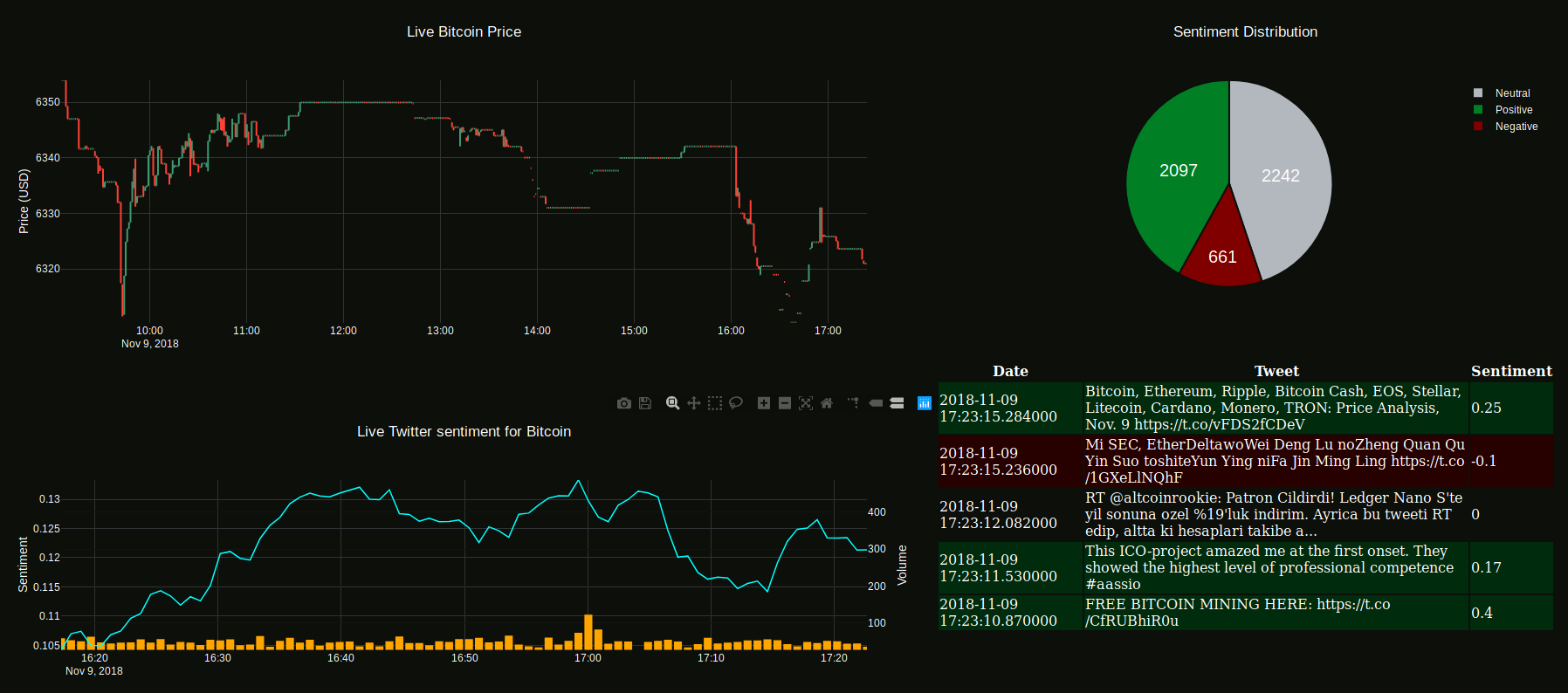
Rohit Kumar

Mona Mann

**Overview**

The dashboard is built on python dash and plotly library and has 4 major components, all of which update in real-time.

1. *Live Bitcoin Price:* Bitcoin price (Open, High, Low, Close) taken from [CryptoWatch](https://api.cryptowat.ch/markets/coinbase-pro/btcusd/ohlc). The website provides data for every minute.
2. *Live Twitter Sentiment:* Everyday 500 million tweets are generated making Twitter the most lucrative space to find the most recent talk about any subject. This graph shows how many tweets are talking about Bitcoin and the overall sentiment. Both factors are calculated on the data collected over a period of one second. The graph updates every second.
3. *Sentiment Distribution:* This pie-chart shows the overall distribution of sentiment on the entire data set collected. This graph updates every second.
4. *Live Tweets:* 5 of the most recent tweets with the time stamp and respective sentiment. This table updates every two seconds.



**Design Thinking**

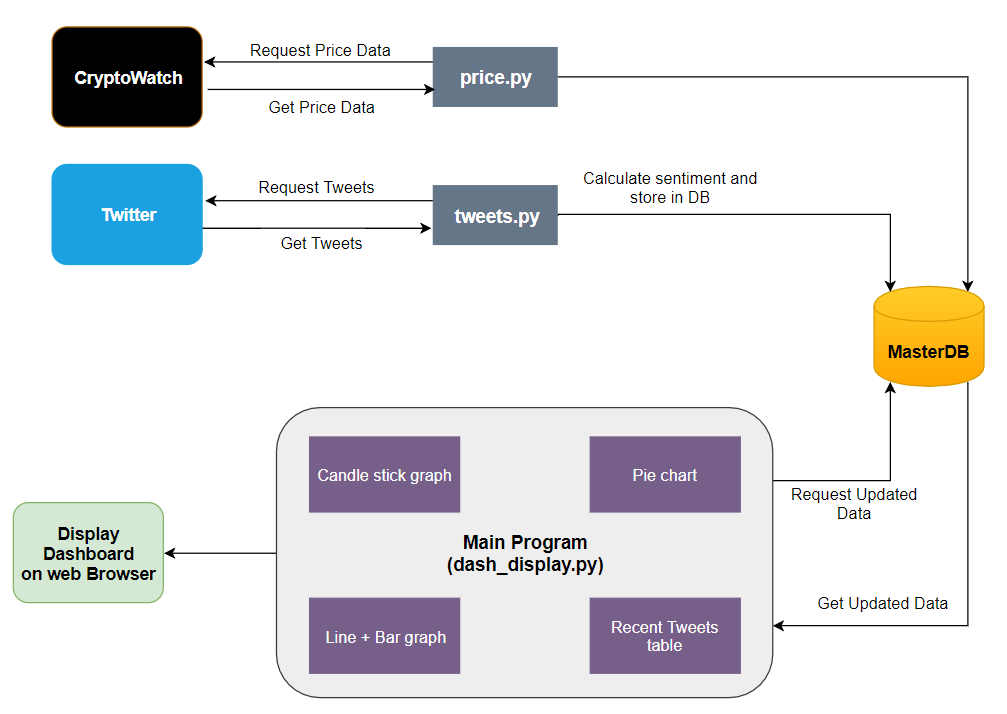
The initial plan was to create a single program that collects price from CryptoWatch, tweets from Twitter and stores it a database for further processing. However due to the design complexity, multiple sources, and keeping simplicity in mind the code was broken down into three major chunks i.e., get price, get tweets and calculate sentiment, and display data.

CryptoWatch provides data for every minute and punishes with a temporary access block if too many call requests are made. Hence, since the data is updated every minute on the website, it made logical sense to make the program sleep for 60 seconds after every request.

On the other hand, Twitter punishes if disconnection happens multiple times in a small period of time. Twitter API ‘streams’ data and streams are meant for continuous flow.

The MasterDB has two tables – masterData (to save CryptoWatch data) and sentiment (to save tweets and sentiment).

A detailed discussion on sentiment is made in the next section.



**Sentiment**

***Plan A***

To calculate sentiment, the initial design thinking required us to train 5 classifiers – Naïve Bayes, Multinomial Naïve Bayes, Bernoulli Naïve Bayes, Logistic Regression, and Linear SVC. The training was done on movies reviews data of 10000 movies, found online.

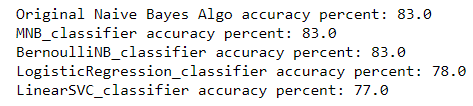
The program (old\_sentiment.py) performed, as expected, when working on Windows system.

Two fictional reviews were fed to the program:

*“This movie was awesome! The acting was great, plot was wonderful, and there were pythons...so yea!”*

*“This movie was utter junk. There were absolutely 0 pythons. I don't see what the point was at all. Horrible movie, 0/10"*

**Accuracy after training and testing the classifiers (Windows:**





**Sentiment result for the two reviews (Windows):**



Unfortunately, after multiple attempts, for days, the same code failed to provide the same output in Ubuntu, even though the accuracy was almost same.

**Sentiment result for the two reviews (Ubuntu):**



We made multiple attempts to train/retrain the classifier on different dataset (nltk has a movie\_review dataset) and by going ahead with just one classifier or changing the number of classifiers, but unfortunately, we ended up with the same result.

Another reason to drop this Plan was that training classifiers consumed a large amount of memory and time (~8 minutes). To save time we tried using the pickle library so that we could saved the trained classifiers for future use. This did help but still every it required approximately 3-4 minutes to calculate the sentiment of a single line.

***Plan B***

With Plan A failing to work on Ubuntu, we moved on to the other choices available – vaderSentimentAnalyzer and TextBlob, both of which provide quick methods to calculate sentiment. After careful review, we chose to go ahead with TextBlob due to it’s higher accuracy.

For bitcoin, many tweets were received every second and hence, for simplicity and graphical representation purposes, we applied rolling mean method to the Sentiment values to smoothen it out. The Live Twitter sentiment chart incorporates smoothened sentiment values and the pie chart uses the raw values.

Twitter provides data every second and CryptoWatch, every minute. Due to this fact, the x-axes of the graphs have not been kept in sync.

**Libraries**

The following libraries were used to build the dashboard:

* **tweepy –** Standard python library to interact with Twitter API
* **json –** used to read the json data collected from CryptoWatch
* **unidecode -** used to decode the Unicode characters in tweets
* **requests -** used to send a get request to an URL
* **textblob -** Standard python library for sentiment analysis
* **nltk -** Standard python library for natural language processing
* **sklearn -** Standard python library that contains various classifiers
* **statistics -** Standard python library containing various statistical functions
* **dash –** used for interactive plotting and hosting it on a local server
* **plotly –** used for interactive plotting
* **pandas –** used for reading data from database and manipulating it
* **sqlite3 –** used for creating a connection with a database where data is stored and read from
* **pickle –** used to save trained classifiers for future use
* **vaderSentimentAnalyzer -** Standard python library for sentiment analysis
* **time –** Used to make the pricy.py code sleep for 60 seconds

**Steps to run the code:**

We need three terminals active to successfully get the output:

1. In the first terminal open and run tweets.py to populate the sentiment table
2. In the second terminal, open and run price.py to populate the masterData table
3. In the third terminal, open and run dash\_display.py to activate the dashboard
4. Visit 127.0.0.1:8050 on your web browser to view the live dashboard.