Real-Time Sentiment Analysis on Twitter to Predict the Cryptocurrency Price

ELEN E6889: Large-Scale Stream Processing - Final Report

May 2023

Tong Wu, tw2906
Eric Chang, cc4900
Yu Wu, yw3748
Yiqing Cao, yc4138

1. Introduction

In a market economy, goods and services have prices because resources are limited, and the price of a given good or service depends on demand and supply. In other words, price is the visualization of the demand-supply relationship model. These are the two fundamental principles of economics. Many factors can influence demand and supply, from government policies to individual buying and selling behaviour. This information can be summarized as expressing sentiment toward a good or service. Intuitively, people use the information they can gather to analyze sentiment to predict prices. And now, with the rapid growth of the Internet, we can get news and people's opinions from social media, which makes gathering information quick. However, we need the help of an algorithm to quickly analyze the sentiment on social media to determine the likely price trend. That's why we chose this topic to help predict cryptocurrency prices by analyzing sentiment in real-time.

We divided the project pipeline into four main parts to analyze sentiment to predict cryptocurrency prices: data streaming, sentiment analysis, prediction model and front-end presentation. Our final goal is to provide an online web page to present our model results, including predicted prices and price trend graphs.

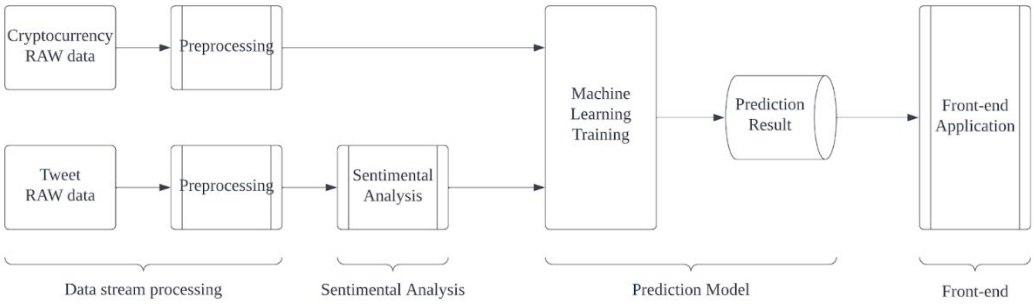

Figure 1 Flow chart of the project pipeline

2. Stream Processing

In the stream preprocessing, it can be roughly divided into five parts, and each part is all done in spark:

Part one: fetching cryptocurrency data

We collected data on five cryptocurrencies: Cardano, Ripple, Dogecoin, Bitcoin, and Ethereum. Using the Yahoo Finance API (yfinance) in Python, we obtained the closing prices online for 180 days. Thus, we have 180 days' worth of data for each of the five currencies, resulting in 900 records.

Part two: fetching Twitter data

To gather tweets related to the five currencies, we utilized Tweepy, Twitter's official API. However, due to limitations on free accounts, we could only collect tweets that were no more than ten days old. With the help of Tweepy's search function, we could filter tweets based on specific keywords related to the currencies of interest. We also utilized PySpark's built-in function to extract relevant information from the tweets, including the posting time, article content, number of followers for the account, and account ID. Once we gathered all data, we input it into a PySpark DataFrame and then converted it to an RDD for preprocessing.

Part three: cleaning Twitter data

To improve the accuracy of our sentiment analysis, we needed to preprocess the raw Twitter data. The original tweets contained unwanted elements such as emojis, tags, mentions, retweets, and URLs. We used PySpark's `regexp_replace` function to replace these elements with blank spaces. In addition, to delete suspected advertising accounts, we use filters to delete data with followers less than 10.

Part four: integrating sentiment analysis model into the pipeline

After cleaning the data, we input it into the sentiment analysis model to generate a sentiment score for each tweet. We then averaged the sentiment scores for each day to obtain a single sentiment value per day. This process resulted in a PySpark DataFrame that contained the sentiment values for each day over approximately ten days. It is worth noting that the sentiment values range between -1 and 1, where -1 represents a negative sentiment, and 1 represents a positive sentiment.

part five: steam optimization

To optimize the system performance and speed, we implemented stream optimization techniques. Specifically, we utilized two techniques taught in class: operator reordering and load shedding. In operator reordering, we identified the filter operator with the smallest number of followers as having the highest selectivity since it would remove a significant amount of data. As a result, we placed this operator at the beginning of the pipeline. We designed a mechanism to avoid discarding too much data in load shedding. Specifically, we randomly generated a number, and if the sender's ID matched that number, we discarded the data. This approach helped reduce the probability of data loss while still improving the overall system performance.

3. Sentimental Analysis

Naive Bayes:

The idea of Naive Bayes is that the learned model calculates the posterior probability distribution for a given sample to be classified. That is, the probability of each label occurring under the condition that this sample appears, and the class with the highest posterior probability is the class to which the sample belongs. The posterior probability is calculated according to Bayes' theorem.

In sentiment analysis, for document d and all categories $c \in C$, the classifier returns the category $\hat{c}$ with the maximum posterior probability for the given document.

$$\hat{c} = arg\ max\ P(c|d) = arg\ max\ P(d|c)P(c)$$

Without loss of generalization, we can represent the document d as a set of features $f_1, f_2 \ldots$

$$\hat{c} = arg\ maxP(f_1, f_2 \ldots |c)P(c) = arg\ max\ P(c)\prod_f P(f|c)$$

But since the Naive Bayes multiplies all the characteristic probabilities, a zero probability in the likelihood term of any class results in a zero probability of the result. The simplest solution is Laplace smoothing.

$$\hat{P}(w_i|c) = \frac{count(w_i, c) + 1}{\sum_{w \in V} count(w_i, c) + |V|}$$

Also, the probability is calculated in log space to simplify the calculation. So we only need to do summation instead of multiplication.

We also implement two optimization methods to improve the performance of the Naive Bayes model. The first method concerns words that appear in our test data but not in our vocabulary. For another category of words: stop words, the solution is to ignore them - to remove them from the test file and not to include any possibility of them. The second optimization method is about the negation of words. A simple baseline commonly used when dealing with negation in sentiment is to prefix each word after the logical negation symbol (n't, NOT, no, never) with the NOT prefix during text normalization. Thus, newly formed words like NOT_like, and NOT_recommend appear more frequently in negative documents and serve as cues to negative emotions, while words like NOT_bored, NOT_reject gain positive associations.

Lexicon

Take WordNet Lexicon as an example. The sentiment analysis workflow includes data collection, text preprocessing, sentiment words popularity calculation, sentiment words score calculation and return of the

sentiment results. Usually, a lexicon for sentiment analysis contains three classes of words: emotional words(positive, negative and neutral emotion), degree words(e.g. a little, very, a bit of) and negative words(e.g. nobody, not, none).

The three very common lexicons are SetiWordNet, Setiwords and VADER. SetiWordNet assigns scores to WordNet synsets instead of words. Each synset has a positivity and a negativity score between 0 and 1. Unlike SentiWordNet, SentiWords assigns scores directly to words, allowing us to use this dictionary without disambiguating the input text first. VADER is a lexicon and a rule-based sentiment analysis tool for social media text. It also covers emojis and abbreviations.

Table 1 Compare Naive Bayes and Lexicon

Approach	**Naive Bayes**	**Lexicon**
Training Effort	Model is fast to train	Pretrained models are easy and quick to implement
Strength	Computation is fast Custom models	Less resource and computationally intensive It contains linguistic rules beyond what is captured in a typical document-term-matrix model.
Weakness	rely on a complete and representative dataset	Susceptible to misspellings, nomenclatures, sarcasm, irony, jargon, and grammatical mistakes as the Lexicon does not recognize them
Accuracy	54.23%	67.43%

Lexicon has higher accuracy in the test dataset. Also, the Naive Bayes model returns a label from 'positive', 'negative' and 'neutral' while the Lexicon model returns a sentiment score between -1 and 1. The Lexicon model could reflect a more precise degree of emotion analysis considering the future prediction model. So we choose the Lexicon model as our final sentiment prediction model.

4. Prediction Model

Flow chart

After stream processing and sentimental analysis, the pipeline of the project input two processed datasets here, and the objective of the prediction model is to find and conclude the potential relationship between price and sentimental value, even if it is obscure.

The right side graph shows the pipeline of the function that constructs the model. We merge two datasets into one dataset and normalize to ensure it has a fixed change range. After that, split the dataset into a training set with 80% and a test set with 20%. Finally, we input the hyperparameter set and training data into the model we construct.

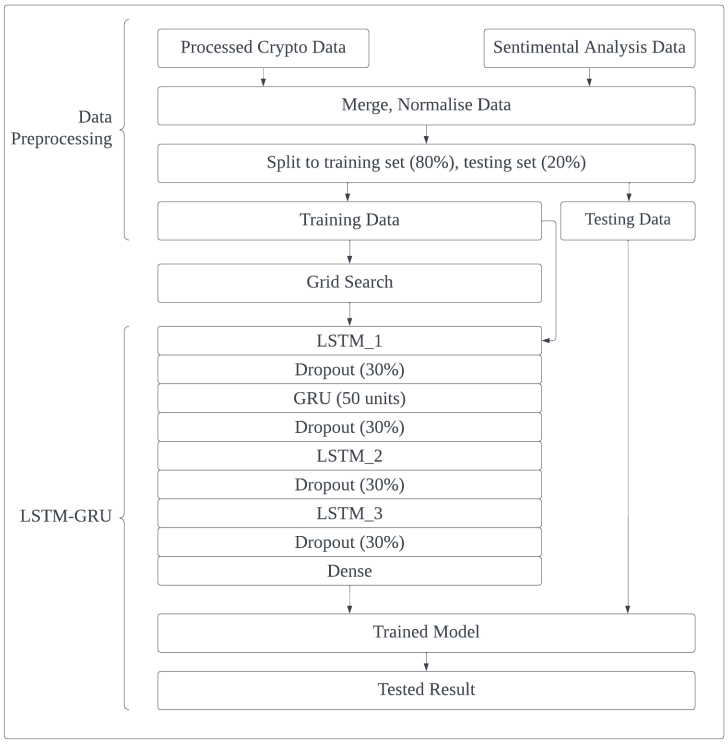

Figure 2 Flow chart of model training

Dataset

The size of the dataset we used to train the model has 180 days of crypto prices and 10 days of the tweet's sentimental value. We use 30 days of crypto prices and 10 days of sentimental tweet value for the test and prediction functions that should be mentioned afterwards. It is worth mentioning that we did the function to support if the fetched date of tweet data is smaller than the number of dates of crypto prices because of the previous statement that the free Twitter API account can only fetch 10 days of data. In this case, we normalize all data outside of the 10-day range to 0 (neutral) to discard the effect of sentimental value. This was a reluctant move. Although we tried using a public dataset containing cryptocurrency tweets for analysis and training, the analysis was much less effective than self-collection.

Model

We use LSTM-GRU hyper model to predict the price. The model mainly contains 3 LSTM layers and 1 GRU layer. We choose LSTM as the model because the crypto prices represent time dependencies, meaning that past prices influence future prices. LSTM is naturally suited for handling time series data. LSTM memory cell has input, forget, and output gates that control the flow of information in, retention, and out of the cell. The model can capture the relationship between past information and current input. Also, crypto prices can be affected by long-term trends and sudden events. LSTMs can capture these long-term dependencies, improving the accuracy of the predictions.

Optimizers

The optimizations are essential for the model accuracy and training time in the training model. We significantly increase the generalization ability and robustness of the model to capture the dataset's features and make predictions quickly.

Grid Search: The grid search can determine the model's accuracy by determining the Root Mean Square Error for each hyperparameter combination.

AMSGrad: AMSGrad retains the maximum of all past-second moment estimates to prevent sudden drops in the learning rate, resulting in better optimization performance. Also, it uses a correction factor to ensure the gradient

average is not underestimated during the iteration to get a better convergence performance. In general, AMSGrad has the best performance and robustness in the grid search results.

Dropout Layer: By adding dropout layers after each LSTM layer of the model, randomly discard the output of a certain percentage of neurons to ensure that the model does not rely excessively on specific neurons. Adding the dropout layer significantly increases the model's generalization ability and prevents overfitting. In this model, we set each LSTM layer to have 30% of the neurons' results discarded.

GRU Layer: GRU has a simpler structure than LSTM, in which GRU has only two gates, the update and reset. The different structure makes GRU has faster training time and can capture different features than LSTM.

Prediction Function

The prediction function loads the model from the local storage, fetches the data set from the pipeline's previous stage, and then outputs the predicted price and price trend graph to the front-end. The prediction function is separate from the training model, which can directly use the trained model to predict. In the actual user experience, the user usually expects a quick response from the website, so we let the prediction function join the project pipeline but not the whole function containing the training model.

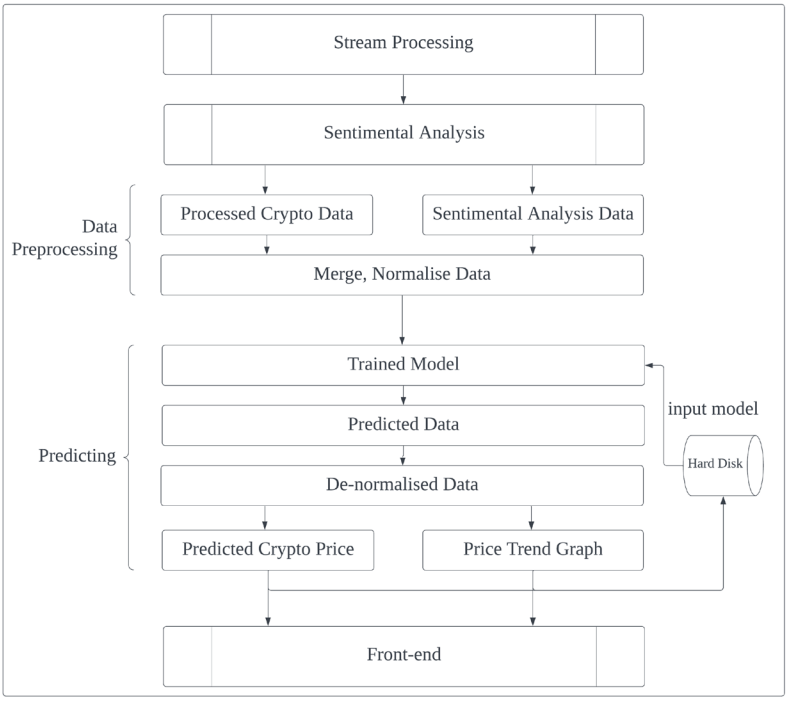

Figure 3 Flow chart of prediction function in the pipeline

5. Front-end

Visualization

The front-end of this project, developed using React, consists of six core components, including a HomePage and five individual cryptocurrency pages. The HomePage's primary function is to introduce users to the platform and assist them in choosing a cryptocurrency. Each cryptocurrency page offers users predictions tailored to the selected coin.

When users launch the app, they are directed to the HomePage, where they can select from five of the most popular cryptocurrencies. Once they choose a coin, the app navigates to the corresponding cryptocurrency page, where users can view price trend predictions for the next 1, 3, or 5 days. Additionally, users can easily return to the HomePage to select another cryptocurrency.

Figure 4 Visualization of the website

Interface

In the initial phase, Flask and Axios were employed as the interface between the back-end and front-end, enabling the back-end to generate trend charts and text for the front-end in real time. However, this approach had a significant limitation: users could only access the trend information on the front-end if the back-end remained running. To overcome this challenge, we altered our approach by directly saving the back-end's predicted trend results as PNG and TXT files in the front-end. Users can access the trends anytime by utilizing 'useEffect' to read the TXT data without needing the back-end to run continuously.

Deployment

A crontab was created to execute the training model function daily, fetching cryptocurrency prices and performing sentiment analysis. When users click the currency button, the prediction function runs, generating price and trend graphs that are subsequently displayed on the front-end. The project was successfully deployed on a website.

6. Summary

In this project, we create and execute a pipeline to fetch cryptocurrency prices and Twitter tweet data from API; sentiment analysis of the tweet data using the Lexicon Vader method and get high accuracy; train the prediction model using the LSTM-GRU hyper model and predict the future prices trend; plot the final results on the real-time update website.

For future work, we may refer to adding more sentimental source data into the analysis model, such as using a purchase account of Twitter API to fetch as much data as you can and also grab the news data from news websites in real-time to increase the accuracy of the analysis of sentiment. We may also refer to improving the prediction model, which now is LSTM-GRU hyper model, which can get high accuracy and avoid over-fitting. The newer and more complex models, such as GAN and WGAM-GP, can be used or added to the current model to capture more features from the dataset.

Overall, the prediction's accuracy can be maintained at a reasonable level to provide a reference to investors. However, the cryptocurrency market is extremely volatile and constantly changing, so the prediction provided in this project cannot be considered absolutely accurate or instructive. Also, we do not make recommendations on any particular cryptocurrency. When using the predicted results provided in this report to make investment decisions, investors should fully understand the risks involved and assume their own consequences and responsibilities arising from using the project's forecast results.