The Efficacy of Different Timesteps in Data when Predicting Cryptocurrency Prices

Andrew Murwin
Directed by Dr. Chris Bryan, Dr. Samira Ghayekhloo
School of Computing and Augmented Intelligence at Arizona State University
Barrett, the Honors College at Arizona State University

Abstract

This thesis serves as an experimental investigation into the potential of machine learning through attempting to predict the future price of a cryptocurrency. To begin the thesis, I first had to decide which cryptocurrency to predict. Ultimately, Dogecoin was chosen as the target as it was a large enough cryptocurrency that it had available data I could gather. This was due to its relative stability compared to Bitcoin. At the time of the data collection, Bitcoin became a much more frequent topic in the media, and had more significant fluctuations due to it. After that, the next step was to collect the data. I used Robinhood as my source of information for Dogecoin's prices. After the data was collected, a four day time period was selected, and it was processed into a consistent timestep. From there the data was processed into three separate timesteps: one minute, five minutes, and ten minutes, with each dataset having a linear regression trained on it with a 75/25 train-test split. This process was then repeated ten times. The models generated only achieved one of the two goals. The models were able to accurately predict test data given all the preceding test data, but were unable to autoregressively predict future data given only the first set of test data points. Ultimately, this project helps illustrate the complexities of extended future price prediction when using simple models like linear regression.

Introduction

With a surge in the popularity over the past few years, the relevance of cryptocurrencies has become more relevant than ever. Cryptocurrencies are a form of decentralized, digital currency that are made up of three main components: wallets, transactions and a ledger. The wallets serve as endpoints for transactions, as well as a way to store cryptocurrencies. Inversely, transactions serve as a way to transfer cryptocurrency from one wallet to another. Finally, the ledger serves as a record of all transactions. On the technical side, cryptocurrencies get much more complicated. One reason cryptocurrencies have become more popular is that they inherently reduce the possibility of double spending using a technique called triple bookkeeping ("Triple-Entry Accounting", 2020). Triple bookkeeping works by using a third party to validate transactions. In the case of a proof-of-work system, such as Bitcoin or Dogecoin, crypto miners process and validate transactions in exchange for the possibility of being rewarded with their respective cryptocurrency. Proof-of-stake works in a similar manner, but rather than having many different miners competing with each other, a validator is randomly chosen out of a pool, with each validator's odds being reliant on the amount of cryptocurrency they "stake" ("What is 'Proof of Work'", 2022). Once validated, the transaction occurs and is added to the ledger for all computers connected to the blockchain to see.

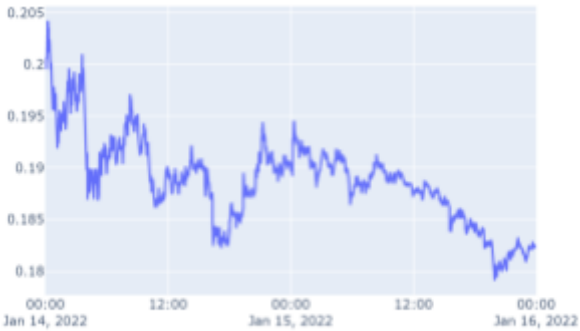

Figure 1: Continuous Dogecoin Data

While these are mostly technical details, it is important to help indicate why cryptocurrencies are such an interesting topic, and what makes

them unique. One of the larger points, and my reason for conducting this predictive experiment as opposed to one composed of traditional stocks, is that the cryptocurrency market never closes. Transactions occur 24/7, giving a stream of non-stop, continuous data to work with. Another aspect is their decentralized nature. While a decentralized currency isn't under the control of any one body, it also means that there are no regulatory bodies either. Cryptocurrency prices can be manipulated by individuals or groups with enough financial power through exploiting theories of supply and demand. They may also just be responsive to public sentiment. In either case, it can lead to exponential rises and sudden falls in prices with no preventative measures. With the market cap for all listed cryptocurrencies at 962.8 billion USD as of October 22, 2022 ("Top 50 cryptocurrency prices", 2022), cryptocurrencies serve as a relatively new market worth exploring.

In this thesis, I developed multiple linear regression models to predict the future price of cryptocurrency. To do this, I had to create the cryptocurrency dataset, which was accomplished by collecting the price of Dogecoin at extremely frequent intervals, and processed into different timesteps. The collected data was used as training and testing data for creating and comparing several machine learning models, differentiated by the timestep used. I compared the efficacy in autoregressively generating future prices, both in terms of how far into the future the price was predicted and how accurate those predictions were on average. Ultimately, the takeaway from this project is that although predicting a short time into the future is relatively simple, further out predictions require more sophisticated systems than were tested.

Background

With the rise of cryptocurrencies, it becomes even important to clarify what a cryptocurrency is should it become a topic in conversation. Cryptocurrencies as a whole serve as a form of decentralized currency, where the value and supply is not controlled by a single entity. There are many different forms of cryptocurrencies, with the main ones being Bitcoin, altcoins, and stablecoins. Bitcoin is the first and most valuable cryptocurrency, at least as of the time for writing this paper. Altcoins cover most cryptocurrencies that aren't Bitcoin, like Ethereum and Dogecoin. Functionally, they are similar to Bitcoin, they just exist as a separate currency, often on a separate blockchain. Stablecoins are cryptocurrencies that function similar to Bitcoin, but have their value "locked" to a specific fiat currency to guarantee value (Hertig, 2022). While previous similar ideas existed, the form of cryptocurrency that is well known today was created in the form of Bitcoin by Satoshi Nakamoto. Interestingly though, much like the cryptocurrencies he helped create, Nakamoto remains anonymous to the general public (Hayes, 2022). A few months after publishing a paper outlining the practical implementation of a decentralized system using a technology called blockchain, the original source code for Bitcoin was created and released by Nakamoto. Since then, hundreds of cryptocurrencies have branched off of this main code to create the variety of cryptocurrencies available today.

Before looking at the approach taken in this thesis, it is important to understand the technology stack used, the data analyzed, and the limitations of both. Unfortunately, data in the

smaller timesteps required for this project are not easily accessible, and needed to be collected. Through the use of an open source Robinhood API for Python, a Raspberry Pi was set up to collect and store the data. Robinhood was selected as the target data source due to the fact that it did not require a paid API key.

All data processing was done in Python using Pandas (McKinney, 2010). Python was selected as the main programming language for this thesis due to its popularity in both processing large datasets and machine learning. Pytorch was used as the main Python library for the machine learning portion of the project, with basic models and optimizers being available for use (Paszke, 2019).

The majority of the time spent on the project came from the machine learning portion of the thesis. Machine learning in general is a computer system that learns and adapts to achieve a purpose based on predefined algorithms, statistical models, and provided data (IBM, 2022). This process is done through the implementation of both statistical and probabilistic techniques to derive said information. One of the most common examples, and the model used in this project, is multiple linear regression. Linear regression uses the assumption of a linear relationship between the input(s) and output to approximate the weights needed to fulfill said relationship. The weights are approximated due not only to the fact that they can't always be directly calculated given enough data, but also because calculating the weights directly can potentially lead to the model overfitting on the training data, leading to a worse performance.

Linear regression models are computed through the procedural optimization of the coefficients using an optimization function, with the one employed in the project being stochastic gradient descent. During each run, the model looks at each data point, predicts what the output should be, calculates the mean squared error loss, and calculates and updates the gradients based on how close the prediction was.

Approach

The first step was to gather the data necessary for the prediction models. A Raspberry Pi was set up to request the cryptocurrency's current information from Robinhood, wait a second, and then repeat, for 24 hours, at which point the data would be saved to a local csv.

```
totp = pyotp.TOTP("twoFactorAuthentication").now()
login = robin.login("emailAddress", "password", mfa_code=totp)
tag = sys.argv[1]
csvName = datetime.now().strftime(tag + "_%m%d%Y.csv")
with open(csvName, 'w', newline='') as output:
    csvWriter = csv.writer(output)
    startTime = datetime.now()
    while datetime.now() < (startTime + timedelta(hours=24)):
        try:
            time.sleep(1)
            data = robin.crypto.get_crypto_quote(tag, info=None)
            temp = (data['ask_price'], data['bid_price'], data['mark_price'],
                    data['high_price'], data['low_price'], data['open_price'],
                    data['volume'], datetime.now())
            csvWriter.writerow(temp)
        except:
            print("Gateway error")
```

Figure 1: Data Collection Snippet

The script would then restart while a parallel process uploaded the file to Google Drive to prevent any gap in the data collection. The data was then downloaded and compiled into a joint dataframe. For the project, the data for the month of January 2022 was used. Overall, for 31 days, the raw data averaged out to about 51 points of data per minute, or 2.26 million data points for the month. Each data's CSV was approximately 6.5 megabytes, for a total of 198 megabytes of data for January.

	ask_price	bid_price	mark_price	high_price	low_price	open_price	volume	Time
0	0.171430	0.170450	0.170940	0.172730	0.169663	0.170040	0.0	2022-01-01 00:00:00.653622
1	0.171430	0.170450	0.170940	0.172730	0.169663	0.170040	0.0	2022-01-01 00:00:01.824734
2	0.171430	0.170450	0.170940	0.172675	0.169663	0.170040	0.0	2022-01-01 00:00:02.997146
3	0.171370	0.170364	0.170867	0.172730	0.169663	0.170040	0.0	2022-01-01 00:00:04.167257
4	0.171370	0.170364	0.170867	0.172730	0.169663	0.170040	0.0	2022-01-01 00:00:04.850969
...	...	...	...	...	...	...	...	...
2262618	0.142869	0.142220	0.142544	0.143340	0.142077	0.142975	0.0	2022-01-31 23:59:54.842962
2262619	0.142869	0.142220	0.142544	0.143340	0.142077	0.142975	0.0	2022-01-31 23:59:56.026093
2262620	0.142869	0.142200	0.142534	0.143340	0.142077	0.142975	0.0	2022-01-31 23:59:57.210228
2262621	0.142869	0.142190	0.142529	0.143340	0.142077	0.142975	0.0	2022-01-31 23:59:58.394040
2262622	0.142869	0.142200	0.142534	0.143340	0.142077	0.142975	0.0	2022-01-31 23:59:59.582202

Figure 2: Original Data Format

One of the complexities with scraping data on a per-second basis was that the time between data points collected was inconsistent. As a result of inconsistent server responses and delays, data is not perfectly evenly distributed as would be preferred. A script was written to filter the data such that a new dataframe containing only the first data point after every N seconds would be included. Three base dataframes were created, saved, and loaded at the beginning of each model creation for consistency. These three dataframes, with timesteps of ten seconds, one minute, and one hour, served as a base for final filtering before the models were run. After trimming the dataframes, their sizes and data point counts were both reduced significantly. The one minute dataset was 7MB with 44,640 data points, the five minute model was 1.4MB with 8,928 data points, and the ten minute model was around 700 KB with 4,464 data points. This step both added consistency and made loading in the data significantly faster.

	ask_price	bid_price	mark_price	high_price	low_price	open_price	volume	Time
0	0.17143	0.17045	0.17094	0.17273	0.169663	0.17004	0	2022-01-01 00:00:00.653622
1	0.17138	0.170447	0.170913	0.172675	0.169663	0.17004	0	2022-01-01 00:01:00.069572
2	0.17136	0.17035	0.170855	0.17273	0.169663	0.17004	0	2022-01-01 00:02:00.205269
3	0.17145	0.17043	0.17094	0.17273	0.169663	0.17004	0	2022-01-01 00:03:00.426721
4	0.17154	0.17053	0.171035	0.17273	0.169663	0.17004	0	2022-01-01 00:04:00.712017
...	...	...	...	...	...	...	...	...
44635	0.142898	0.1422	0.142549	0.14334	0.142077	0.142975	0	2022-01-31 23:55:00.157258
44636	0.143	0.142362	0.142681	0.14334	0.142077	0.142975	0	2022-01-31 23:56:00.523735
44637	0.142987	0.14229	0.142638	0.14334	0.142077	0.142975	0	2022-01-31 23:57:00.862895
44638	0.14299	0.14228	0.142635	0.14334	0.142077	0.142975	0	2022-01-31 23:58:00.064264
44639	0.142984	0.1423	0.142642	0.14334	0.142077	0.142975	0	2022-01-31 23:59:00.407007

Figure 3: Dataframe with one minute timestep

As this thesis is ultimately an experiment to compare different timesteps, the project had to be set up in such a way that multiple timesteps could be run in parallel to allow for as direct of a comparison as possible. From this position, additional index manipulation could be used to get appropriate subsets of the dataframe. Ultimately, all three models used the one minute timestep as the base.

```
train_df = monthFilters[0].iloc[int(8640*(start_day-1)+360*start_hour):
                                int(8640*(start_day+2)+360*start_hour):
                                int(timeframe/10)].copy(deep=True).reset_index(drop=True)
```

Figure 4: Index manipulation for dataframe subsets

Through changing the start_day, start_hour, and timeframe, the dataframe could be shifted and filtered appropriately. There were thirty total models run in total for comparison: the first ten days of the month with timeframes of one minute, five minutes, and ten minutes. For each timestep, a best-fitting epoch learning rate pair was found empirically. A time of 60 seconds used 10000 epochs and a learning rate of 0.0005, 300 seconds used 100000 epochs and a learning rate of 0.005, and 600s used 100000 epochs and a learning rate of 0.1.

Using the three datasets of one, five, and ten minute intervals, each four day segment was run through the model using a 75:25 split, or a three-day training, one-day testing split. Each model was then used to make two types of predictions. The first was a form of validation, where the model was run directly on the test data to verify the accuracy of the model in a predictive situation. The second prediction was an autoregressive prediction of the test data.

```
# Autoregressive Predictions
newPreds = torch.Tensor([])
predSet = torch.tensor([x[0] for x in test_df_ask_price[0:50].tolist()])

newPreds = torch.cat([newPreds, model(predSet)])
predSet = torch.cat([predSet[1:], newPreds[-1].reshape(-1)])
index = 1

while index + 50 < len(test_df) and max(newPreds[-1].detach(),
    test_df['ask_price'].iloc[index + 50])/min(newPreds[-1].detach(),
    test_df['ask_price'].iloc[index + 50]) < 1.03:
    newPreds = torch.cat([newPreds, model(predSet)])
    predSet = torch.cat([predSet[1:], newPreds[-1].reshape(-1)])
    index += 1
```

Figure 5: Price Prediction Script

Given the first fifty test data points, the model would predict the following price and append it to those first fifty data points. It would then take the newest fifty data points (skipping the first data point in this instance), and predict the following price. This process was repeated until a prediction that was more than 3% away from the true price was guessed. Originally the cutoff was going to be either 1% or 5%, but 1% did not provide enough leeway with data fluctuations and would cut off the prediction early, and 5% provided too much leeway, to the point where almost every prediction made it the full time, even if it was a poor prediction. After testing, 3% was found to be a solid medium between the two.

Results

While each of the models had their limitations in predicting cryptocurrency over a long duration, there were still some interesting results that could be extracted from the data collected. On average, the one minute model predicted 453.6 timesteps out of a maximum of 1390 timesteps. The five minute model averaged at 66.6 out of a maximum of 238, and the ten minute model predicted an average of 35.7 out of 94 possible steps. For each model, there were five important slices of data that needed to be graphed to fully explore the results. The first two, the training and test data, give a baseline for the model's performance to be compared against. The third was the result of running the model on the training data, and the fourth was the result of running the model on the test data. These two serve as the baseline for determining the accuracy of the model, both visually and numerically. The fifth was the result of the retrogressive generation of price prediction for the test data.

Figure 6: Models run on Day 1

As seen in the graphs, the models do a good job both in recreating the training data, and in predicting the test data. Even so, when procedurally predicting the prices, the model predictions tend to lack a lot of the nuance available in the true data, resulting in a near-linear prediction.

	One Minute Model			Five Minute Model			Ten Minute Model		
Day	Accuracy	Pred Length	Pred Time (hrs)	Accuracy	Pred Length	Pred Time (hrs)	Accuracy	Pred Length	Pred Time (hrs)
1	0.99730	663	11.05	0.99822	238	19.83	0.99797	94	15.67
2	0.99361	684	11.4	0.99489	103	8.58	0.99443	18	3
3	0.99693	365	6.08	0.99707	77	6.42	0.99684	74	12.33
4	0.99708	38	0.63	0.99708	69	5.75	0.99642	19	3.17
5	0.99712	583	9.72	0.99707	47	3.92	0.99623	15	2.5
6	0.99783	1390	23.17	0.99792	55	4.583	0.99732	30	5
7	0.99616	300	5	0.99599	28	2.33	0.99628	48	8
8	0.99524	217	3.62	0.99554	4	0.33	0.99525	6	1
9	0.99675	232	3.87	0.99629	18	1.5	0.99572	52	8.67
10	0.99478	64	1.07	0.99450	27	2.25	0.99058	1	0.17

The average duration of each of the three retrogressive models is 32.63%, 27.98%, and 37.98% respectively. One important thing to note is that, although the ten minute predicted the longest percentage-wise, it actually predicted the shortest amount of time into the future. This is an effect of the test data needing the first fifty test data points to begin testing. For the one minute model, this decrease in data doesn't cause that big of an impact on the time available to predict, but for the ten minute model, it cuts out almost a third of the time able to be predicted. In terms of time predicted, the one minute was best at 7.16 hours, with the ten minute following at 6.62 hours, and the five minute performed the worst at 5.55 hours. The issue with these numbers is that although they are averages, most prediction lengths are not actually close to the predictions. As such, the predictions, while of decent length on average, are not consistent for any of the three model types. Each timestep performs better on some datasets and worse on others with no apparent pattern. It appears that there is no correlation between the accuracy of a model and its ability to autoregressively predict future values. Ultimately, the models can predict the future, but only a few seconds out at a time.

Discussion

While not necessarily flaws, there were some shortcomings that revealed themselves throughout the course of the project. One was the limitations of the linear regression model. During research, there were also attempts to run different types of models beside the regression. One such model was the LSTM. While a lack of reproducible accuracy meant it wasn't fully pursued, some runs showed much stronger autoregressive future predictions that were non-linear, one of the key issues that revealed itself with the autoregressive linear regression predictions. Given the time to expand on the project, there are a few other ways I could see this that would be beneficial. One would be training the models on a larger dataset to see if it could detect more nuanced patterns. Due to the extended timeframe of the project, there is over a year's worth of data collected, but only around half a month's worth was used. Another would be to try and build a general purpose model for each timestep. The fact that a new model has to be run on each new dataset prevents any single model from being used for an extended period of time. This could be resolved through the creation of a general model that could predict a much larger time frame. This would also be beneficial as testing data could be fed in repeatedly, meaning that predictive methods like the autoregressive one implemented in this paper could be beneficial while still predicting less time into the future.

Even with these demerits though, there was a lot learned throughout the experiment that helped contribute to its success. One important tool was shuffling the data before training on it. Without shuffling, the model learns the data at the end of the dataset better, and while this is sometimes helpful, it ends up limiting the scope of significant learning to too small of a portion of the data, resulting in a decrease in accuracy. Another was the importance of getting a baseline dataset to begin with. Originally I was processing the data every time I reran the model, and I had to build a different processing function for each model. About halfway through the project, I

shifted to preprocessing and saving the 10 second dataframe, which saved time both during the model processing and when planning out experiments. With the saved dataset, I could use simple python indexing operations on the dataframe to get the timesteps I want rather than having to reprocess and get the data after every specific amount of time.

Conclusion

After completing this thesis, I can say that, while this was an interesting project, it took far longer than it should have. Before starting this thesis, I had never touched machine learning before, and saw this as a good introduction that involved a modern topic: cryptocurrency. Unfortunately, I had vastly underestimated what I needed to know, which caused me difficulty throughout the project. Issues came up with everything from model types to data formatting. Even so, it was a great learning experience. I was forced to learn more of the nuances of machine learning, a constantly growing field at the forefront of computer science, and deepen my knowledge of Python through the use of generators and dataframes, two aspects I had previously avoided. Ultimately, I was able to write my own data collection, design my own data processing, and build my own models, all topics I had never previously had the chance to explore. Ultimately, while I was unable to make any unique discoveries during this process, I was still able to explore a new field in an interesting way.

References

Browning, J. (2018). Utilizing Machine Learning Methods to Model Cryptocurrency. *Barrett, The Honors College Thesis/Creative Project Collection*. Retrieved from https://hdl.handle.net/2286/R.I.48225

Hayes, A. (2022, September 28). *Who is Satoshi Nakamoto?* Investopedia. Retrieved November 19, 2022, from https://www.investopedia.com/terms/s/satoshi-nakamoto.asp

Hertig, A. (2022, September 16). *What is a stablecoin?* CoinDesk Latest Headlines RSS. Retrieved November 19, 2022, from https://www.coindesk.com/learn/what-is-a-stablecoin/

IBM. (2022, July 14). *AI and Machine Learning platform integration*. IBM Cloud Paks. Retrieved November 14, 2022, from https://www.ibm.com/docs/en/cloud-paks/1.0?topic=cloudpaks_start%2Fibm-process-mining%2Fuser-manuals%2Fai_ml_platformintegration%2Fintroduction.htm

McKinney, W., & others. (2010). Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference* (Vol. 445, pp. 51–56).

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., … Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32* (pp. 8024–8035). Curran Associates, Inc. Retrieved from http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Shah, S. (2019). Predicting Sneaker Resale Prices using Machine Learning. *Barrett, The Honors College Thesis/Creative Project Collection*. Retrieved from https://hdl.handle.net/2286/R.I.52272

Top 50 cryptocurrency prices, Coin Market Cap, price charts and historical data. crypto.com. (2022). Retrieved October 22, 2022, from https://crypto.com/price.

"Triple-Entry Accounting and Blockchain." *University of North Dakota Online*, 3 Feb. 2020, https://onlinedegrees.und.edu/blog/triple-entry-accounting-blockchain/.

"What Is 'Proof of Work' or 'Proof of Stake'?" *Coinbase*, Coinbase, 2022, https://www.coinbase.com/learn/crypto-basics/what-is-proof-of-work-or-proof-of-stake.

Appendix:

Data Collection

```
with open(csvName, 'w', newline='') as output:
    csvWriter = csv.writer(output)
    startTime = datetime.now()
    while datetime.now() < (startTime + timedelta(hours=24)):
        try:
            time.sleep(1)
            data = robin.crypto.get_crypto_quote(tag, info=None)
            temp = (data['ask_price'], data['bid_price'], data['mark_price'],
                    data['high_price'], data['low_price'], data['open_price'],
                    data['volume'], datetime.now())
            csvWriter.writerow(temp)
        except:
            print("Gateway error")
```

Data Upload

```
gauth = GoogleAuth()
gauth.LoadCredentialsFile("mycreds.txt")
if gauth.credentials is None:
    # Authenticate if they're not there
    gauth.LocalWebserverAuth()
elif gauth.access_token_expired:
    # Refresh them if expired
    gauth.Refresh()
else:
    # Initialize the saved creds
    gauth.Authorize()
# Save the current credentials to a file
gauth.SaveCredentialsFile("mycreds.txt") # Creates local webserver and auto handles authentication.
drive = GoogleDrive(gauth)

gfile = drive.CreateFile({'parents': [{'id': 'folderLocation'}]})
# Read file and set it as the content of this instance.
gfile.SetContentFile(sys.argv[1])
gfile.Upload() # Upload the file.
subprocess.Popen(["rm", sys.argv[1]])
print("Uploaded " + sys.argv[1])
```

Data Download

```
drive = GoogleDrive(gauth)
file_list = drive.ListFile({'q': "'1gwh3hSm3R--5en_2KrqrT9ksOofRTw5N' in parents and trashed=false"}).GetList()
print([file1['title'] for file1 in file_list])
for file1 in file_list:
    file1.GetContentString() # Gets content as string
```

Data Filter

```
def dataFilter(dataDF: pandas.DataFrame,
               timesteps): #List[timedelta]
    newDFs = [pandas.DataFrame(columns=dataDF.columns) for _ in timesteps]
    nextTimes = [parseTime(dataDF['Time'][0]).replace(second=0, microsecond=0) for _ in timesteps]

    for dfIndex in range(0, len(dataDF)):
        for index, t in enumerate(nextTimes):
            if t < parseTime(dataDF['Time'][dfIndex]):
                newDFs[index] = pandas.concat([newDFs[index], pandas.DataFrame(dataDF.iloc[dfIndex]).T])
                nextTimes[index] += timesteps[index]
    return newDFs
```

Day 6 Models: