Ethereum Price Prediction Using ML

Nirvan Sarju
Computer Science
Ryerson University
Toronto, Canada
nirvan.sarju@ryerson.ca

George Saade
Computer Science
Ryerson University
Toronto, Canada
gsaade@ryerson.ca

Saleh Al-Sailani
Computer Science
Ryerson University
Toronto, Canada
salsailani@ryerson.ca

***Abstract* - Final report of a project that aimed to use machine learning algorithms to predict the price of Ethereum. Features were obtained from a Kaggle dataset for 14 different cryptocurrencies, and the output labels were the price of Ethereum the next day. The models used were a Neural Network, Linear Regression, and Support Vector Machine. The best performing model was the Neural network, as hypothesized. From our results, it is clear that the accuracy of our models can be improved further, and more research should be done into regularization, feature selection, and alternative algorithms such as XGBoost gradient boosting and LSTM layers for Neural Networks. Code can be found HERE [1].**

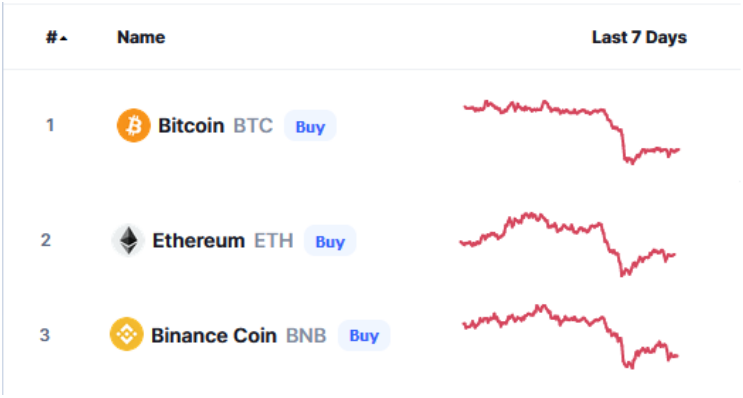

Fig. 1. A visual example of the connectedness of the crypto market [4]

I. Introduction

A. Motivation

In recent times, cryptocurrencies such as Bitcoin and Ethereum have grown from being worth pennies, to being worth tens of thousands of dollars, being thrust into the spotlight of both tech and economic culture. Although these currencies have seen an overall sharp increase in value, their values have also been proven to be quite volatile, so much so that there are even alternatives called ”stablecoins” being proposed [2]. The volatility of the current top cryptocurrencies has attracted interest from investors looking to make profits on trading the currencies. With this increase in interest, ways to accurately pin down the price trends of these currencies have become sought after by investors. It is quite clear from looking at price charts for multiple cryptocurrencies that there is some level of connectedness present in the overall cryptocurrency market. A 2019 study ’Dynamic connectedness and integration in cryptocurrency markets’ [3] shows some research has been conducted to analyze this relationship, finding that Litecoin and Bitcoin are market leaders that tend to dictate how the overall market moves. In this project our team made attempts to find a way to use this connectedness of the cryptocurrency market to predict the price of a single currency - Ethereum. This is where machine learning can be applied, as machine learning algorithms are adept at identifying relations in data, and classifying/predicting output. In this case, it seems that certain models may be able to accurately identify and exploit this proposed connectness between cryptocurrencies to build a stronger feature set for predicting prices than single currency data.

B. Overview

In this project, our team took on the task of relating cryptocurrency prices with each other by using key features of Bitcoin, DogeCoin, Litecoin and other cryptocurrencies to predict the price of Ethereum. The goal of this project was to gain some insight into whether the crypto market is driven primarily by the market leaders, with the smaller currencies following their trend, as well as evaluate the performance of a variety of machine learning algorithms on this problem of cryptocurrency price prediction.

C. Prior Works

There has been a great amount of research into the topic of predicting the cost of cryptocurrencies using machine learning, however it seems that there has been no models proposed that are accurate enough to fully estimate the fluctuations in value that these currencies experience. There are some studies such as the 2018 study ’Anticipating Cryptocurrency Prices Using Machine Learning’ [5] which show that machine learning algorithms can outperform the ’simple moving average’ method commonly used for portfolio’s, even turning a profit. This study specifically uses LSTM layers on a neural network, as well as the XGBoost gradient boosting algorithm. Other studies such as ’Prediction of cryptocurrency returns using machine learning’ [6] have applied models such as Support Vector Machines, Artificial Neural networks, and Logistic Regression with accuracy results of 55–65 percent. It is clear that there are a wealth of models to choose from, and a decent body of work to learn from for this project. It seems that

the profit motive for predicting cryptocurrency prices, as well as the general interest in a form of currency that has both technical and financial aspects has driven interest in the world of academia.

II. Problem Statement and Dataset

A. Problem

The problem that we tackled in this project was predicting the price of Ethereum using features of the other cryptocurrencies in the market. So, using features extracted from the day previous, a model is trained that can predict the closing price the next day. This is a fairly standard regression problem. We will be using features such as high, low, open, close, volume, and market cap from a variety of cryptocurrencies in order to predict the next-day value. A variety of algorithms such as linear regression, support vector machine, and neural networks will be used to find the most accurate estimator given these features. Understanding the volatility of the market, we hope to be able to roughly estimate the trends of Ethereum, and hopefully get the predicted prices close as well.

B. Challenges

There are some challenges in solving this problem from a theoretical standpoint. Ethereum is a volatile cryptocurrency, and it is possible that our models will face significant difficulty in accurately predicting the magnitude of its fluctuations consistently. The dataset is planned to use a rolling average of features from the last 30 days to reduce jitter in samples, but this may lead to predicting smaller fluctuations in the price compared to the actual result. Another issue may be the inclusion of too many low-value currencies. low value currencies can fluctuate significantly more than the higher-value currencies, and they can also experience spikes due to increased interest that don't necessarily have any effect on the larger currencies. This failure to track with the larger currencies such as Bitcoin and Ethereum may make the predictions worse. Something else that may be an obstacle is the change of scale of the cryptocurrency prices over time. Our dataset spans over 3+ years, and over those years the larger coins have grown significantly in price, which may affect the predictive ability of the model, since the models are trained on the oldest samples, and predict on the newest.

C. Data set

The dataset that we will use to train our prediction model is the 'Cryptocurrency Historical Prices' dataset from Kaggle [7]. This is a public dataset provided by Kaggle user 'sudalairajkumar'. This dataset compiles data from 23 cryptocurrencies with notable examples being Bitcoin, Ethereum, and DogeCoin, which should allow us to get a decent grasp on the overall trends of the market through data processing. The features that are listed on the dataset are 'date', 'high', 'low', 'open', 'close', 'volume', and 'market capitalization'. We are interested in every feature except for the date, as we did not believe that the date was going to provide valuable information for models to predict the price of Ethereum. The time period in which these features were captured was once per day, meaning that the time interval for this dataset is one day.

III. Methods and Models

A. Data Preprocessing

In order to prepare the dataset for model training, the data for each of the 23 cryptocurrencies included in the dataset were compared in order to establish the common start date for each of the currencies, as some of the currencies were established later than others. From this comparison, it was discovered that AAVE was the newest cryptocurrency, with its data starting from October 5, 2020. Considering that the data only includes dates up to July 6 2021, if we started the processed dataset from this date we would only be able to include 275 samples. After consideration, it was determined that a minimum of 1000 samples was necessary to ensure that the models could be trained properly; this would require a new start date for the data, leading to elimination of some of the currencies from the original dataset. The new start date that was decided was October 2 2017, the start date of the data for Cardano. This new start date allowed for over 1000 samples to be included in the final dataset, and also lead to the elimination of 8 of the original cryptocurrencies, leaving 16 for the final dataset.

Although the dataset contains data sampled on a daily basis, it was deemed necessary to take a 30-day average of each of the selected features in order to reduce the volatility or "jitter" associated with samples taken too often or days where the currency saw a massive change in value due to external factors such as good/bad publicity in the news or adoption by certain marketplaces. The formula that was used to take the moving average of the data was a standard mean the samples over the last 30 days calculated by using excel formulas:

$$avg_{30} = \frac{1}{30} \sum_{i}^{i+29} x_{30-i} \tag{1}$$

For the neural network, the dataset was normalized using Keras in order to ensure that the scales of the features were comparable.

B. Features Selection

After the newest cryptocurrencies were eliminated and the moving averages were calculated for the remaining data,the features that were selected for the dataset were the high, low, open, close, volume, and market capitalization of each of the cryptocurrencies from the Kaggle dataset. The only feature that was removed was the date, as it was determined that the date values did not provide valuable, quantifiable information to the models. The price of Ethereum on the last day of the 30 day average was used as the output labels for the dataset. A feature of the data that was not explicit was the sequential ordering of the samples; in the case of a model such as an RNN that requires time-series data, it was important to ensure

that the data was in the proper order so that those models can train correctly. This sequential ordering was also important for the data splits, as explained in the next section.

C. Dataset split

After all preprocessing, the dataset was split into training, validation, and test splits with percentages of 60% / 20% / 20% respectively. Originally, the data was split before it was shuffled, in order to ensure that the models were being trained on the oldest data, and testing on the newest. Unfortunately, due to the tendency of Cryptocurrencies to grow in value over time, this meant that training on the oldest data did not allow the models to accurately predict samples from later time periods (such as those in the test set) as the model had not seen inputs/predicted outputs of that scale before. In order to combat this inaccuracy, the data splits were modified to shuffle before being split, and this provided better performance, as can be seen in the results section.

D. Models

Seeing as this problem is a fairly standard regression problem, the models chosen were selected by starting from the lowest complexity (linear regression), and trying out increasingly complex/sophisticated models (SVM, Neural Network) in order to observe the difference in performance.

1) Linear Regression: As it was identified that this this a fairly standard regression problem, regression was used since it has been around and studied for so long. More specifically, Ordinary Least Square Regression was used to find a single output variable (price) from a linear combination of input variables. Inputs are independent as this is an OLSR model. Experimentation with shuffling of the dataset splits will be shown in the results. The only hyperparameter that can be tuned in linear regression is fitting an intercept and not a lot of difference is seen between the test/valid sets. By removing the intercept in the calculations, the model performed slightly worse. according to the documentations we can infer that the data is expected to be centered and this is because the performance better by adding the intercept.

2) Support Vector Machine: Support Vector Machines were researched and used as theoretically it should provide better results than linear regression. Different SVM kernels functions like the Linear, Polynomial, Gaussian Radial Basis Function, and the Sigmoid Kernel were explored. SVR model includes a variety of hyperparameters to tune, including a regularization parameter C. For this purpose, a GridSearchCV method was used to find the best values for our model. Knowing that polynomial kernel is the best performer; the hyperparameters selected to tweak were the degree (again), epsilon, and the regularization parameter C. The GridSearchCV method decided on these values as the best: 'C': 10, 'degree': 3, 'epsilon': 0.01.

3) Gradient Boosting: Gradient Boosting will be explored as the third algorithm since it is known for its accuracy with large data. Experimentation with hyperparameters were explored. the hyperparameters were also picked by utilizing the gridsearchcv method. 'nestimators': [25,50,100,150,200] , 'learningrate':[0.1, 0.25,0.5,1], 'maxdepth':[1,3,6,8,10] are the hyperparameters combination and the hyperparameters chosen were: 'learningrate': 0.25, 'maxdepth': 6, 'nestimators': 100.

4) Neural Network: A standard Deep Neural Network (DNN) was used, implemented using TensorFlow and Keras. The hyperparameters that were tweaked during training were the learning rate, epochs, and the depth/width of the network. The network employs Keras "Dense" layers, meaning that each neuron in the network takes in input from all of the neurons in the previous layer. In general, the network was expanded width-wise (larger layers) first, then reduced in width until the optimal solution was found. The learning rate and epochs were adjusted relative to each other, with more epochs corresponding to a decreased learning rate. The chosen model had a learning rate of 0.001, trained for 200 epochs, and consisted of a 6-layer network, with its widest layer being 1024 units.

E. Performance Metrics

The performance metrics that were considered to assess the accuracy of our models were Mean Absolute Error (MAE):

$$MAE = \frac{\sum_{i=1}^{n} |Predicted - True|}{n} \tag{2}$$

and Mean Squared Eror (MSE).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(True - Predicted)^2 \tag{3}$$

After testing our models, we found that Mean Squared Error was less useful of a metric than Mean Absolute Error, because it does not accurately represent the margin of error of the model due to the fact that it assigns much more weight to errors than Mean Absolute Error. Mean Squared Error works very well for models that have very high accuracy, however the models trained in this project tended to have lower accuracy, leading to exaggerated values.

IV. Results and Discussions

A. Linear Regression

The linear regression model was first assumed to not perform very well when taking into consideration how volatile the crypto currency prices and trends are. Therefore, the intuition was that a linear relation would not describe the dataset very well. However, two results emerged. Shuffling the data before doing the train/validation/test split gave back unexpectedly relatively good results, with mean absolute error of only 60 dollars, the line of fit looked good graphically as well. It should be known that these good results translate poorly into application because the predicted price made big jumps and changed a lot of its traits throughout the years. Shuffling the data gave the model more advantage by learning more generalized view of it and we may be facing a case of overfitting. The trained model with shuffled data may be of a large bias but very low variance in nature. This was further confirmed when the split approach was more realistic and

```
Mean squared error: 11835.98
mean_absolute_error: 59.87
```

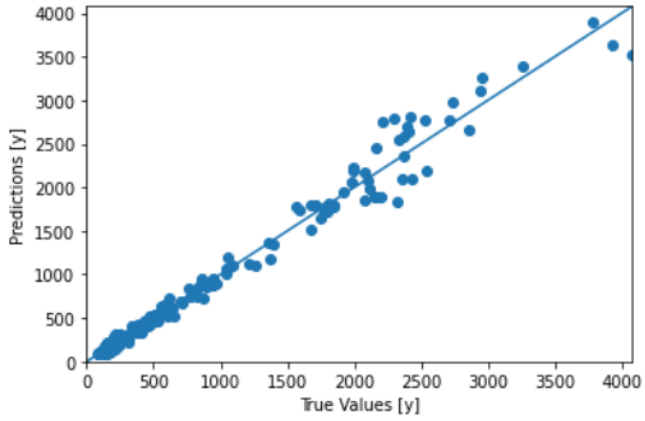

Fig. 2. Linear Regression with a shuffled dataset.

```
Mean squared error: 366665.21
mean_absolute_error: 423.67
```

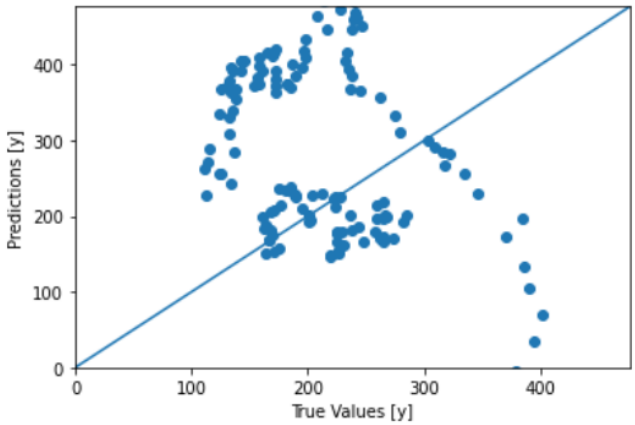

Fig. 3. Linear Regression with a none-shuffled dataset.

was done on a time sensitive basis, the model was trained on old data and tested/validated on future data. The model, as expected, performed relatively worse on both test/valid sets. However, this was a trade off that is absolutely needed for a better bias and variance.

Linear Regression Results	
Model	Mean Absolute Error
Linear Regression	423.67

B. Support Vector Machine

While the support vector regression algorithm is very similar to the linear regression in terms of the theory behind both, the support vector regression algorithm gives us way more flexibility by offering a lot of hyperparameters to tune and refine the model, so it theoretically performs better. SVR offer many kernels and a "for loop" that goes through all the kernels was used initially for some analysis. As shown below, the polynomial kernel clearly did a better job over the other two, RBF, and sigmoid.

```
kernel: poly
Mean squared error: 2138.52
mean_absolute_error: 38.10
kernel: rbf
Mean squared error: 6332.27
mean_absolute_error: 56.87
kernel: sigmoid
Mean squared error: 8976.33
mean_absolute_error: 70.04
```

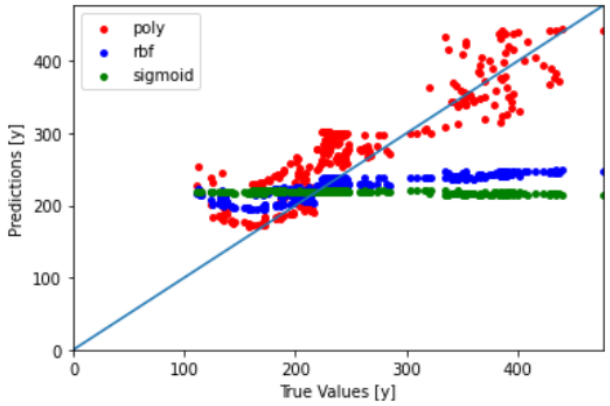

Fig. 4. SVR KERNELS.

A polynomial kernel can use different degrees to fit the model, so to test it further, another for loop was used to test the model further with degrees of 2, 3 ,4, and 9. Surprisingly, the algorithm performed the best with a lower degree of just 2. As shown below, it seems that with higher degrees the model appears to lose fit around the data and especially as the predicted values grows. The model appears to under predict the data by large amounts as the degree of the polynomial grows as well.

The validation set gave Mean squared error: 32999.43. Mean absolute error: 163.00 and the final test on the test set gave Mean squared error: 18659307.16 Mean absolute error: 2878.61.

SVR Results	
Set	Mean Absolute Error
Support Vector Machine	2878.61

The best result on the validation dataset was achieved by manually tuning the hyperparameters. The mean absolute error was only 38.10 and the mean square error 2138.52. Unfortunately, the real test on the test dataset still performed substantially badly with a mean squared error: 337595483.66 and mean absolute error: 12878.98. This is a clear case of overfitting the data and it was discarded. The gridsearch with time series split should give the best variance/bias trade off for any future data.

C. Gradient Boosting

Gradient boost showed great results on the validation set after tuning the hyperparameters.

```
degree: 2
Mean squared error: 2804.11
Mean_absolute_error: 45.37
degree: 3
Mean squared error: 2138.52
Mean_absolute_error: 38.10
degree: 4
Mean squared error: 7754.26
Mean_absolute_error: 68.74
degree: 9
Mean squared error: 27758.86
Mean_absolute_error: 116.42
```

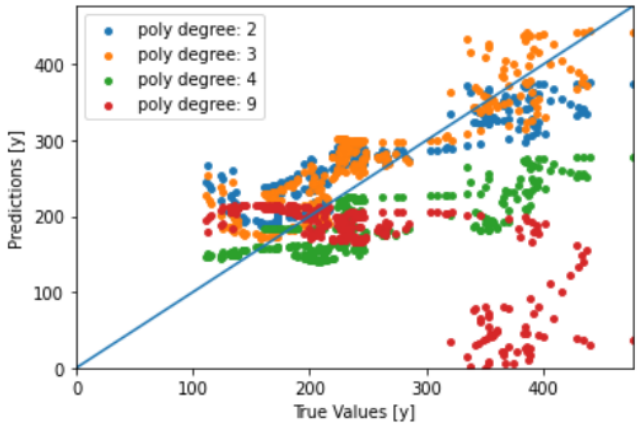

Fig. 5. Poly kernel with different degrees.

Gradient Boosting Results	
Model	Mean Absolute Error
Gradient Boosting	470.06

```
Mean squared error: 356329.31
mean_absolute_error: 470.06
```

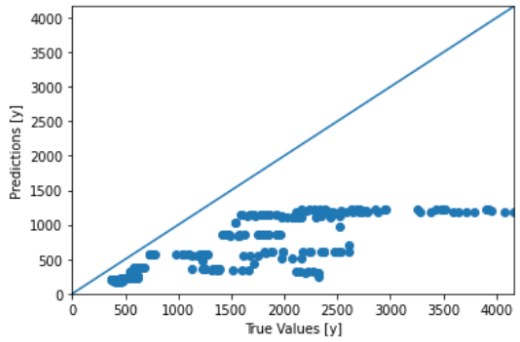

Fig. 6. gradientboost on test set

D. Neural Network

The neural network seemed to provide some of the best predictions out of all of the models. In order to reduce the bias and increase the accuracy of the model, a variety of hyperparameter combinations and model structures were tried. All layers except for the input and output layers had ReLU activation, with the output layer having a linear activation function. The input layer was a normalization layer. The training features were normalized before being used to train the model, as without normalization the results were very poor.

Neural Network Results	
Dataset type	Mean Absolute Error
Split, then shuffled	2404.545898
Shuffled, then split	35.291431

It is clear that while training on the oldest data may help in some cases, the model trained on the ”split, then shuffled” data has a variance issue, and while it performed well on the training and validation data, it was unable to generalize and make quality predictions on the test data. While choosing the better dataset along with model and hyperparameter tuning helped to reduce the bias, there seems to still be a significant amount of bias as shown in the Mean Absolute Error. Noise that comes from overfitting did not seem to be an issue with the model, as the Mean Absolute error shows there is still clearly quite a bit of training to do before the model begins to overfit to the noise in the dataset.

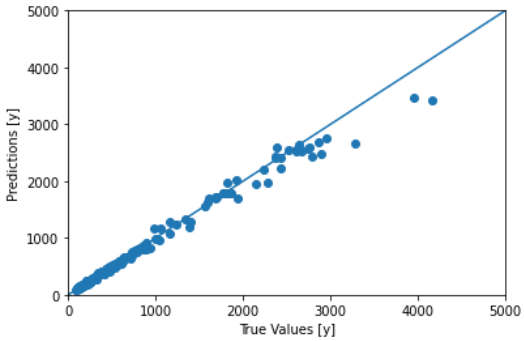

Fig. 7. The accuracy of the DNN compared against the true values

From Fig. 3 we can see that the model generally has good accuracy, but the accuracy has a visible decline as the true label increases in value. This can be attributed to not having enough samples with high-value outputs, as Ethereum has seen a large rise in value and volatility in the time period that makes up the test set.

E. Comparison

Model Comparison	
Model	Mean Absolute Error
Linear Regression	423.67
Support Vector Machine	2878.61
Gradient Boosting	470.06
Neural Network	35.291431

From these results, it is clear that the Deep Neural Network provided the best performance on our test data, and is the most accurate model that we trained. This is in line with our predictions. The Support Vector Machine’s lack of accuracy for this problem was not in-line with our expectations, it seems that the model had an exceedingly hard time relating the features to ouptut. The Linear Regression model performing better than gradient boosting was also an unexpected result,

and more research should be conducted to understand if these results are due to hyperparameter tuning issues, or are more fundamental to the models. Overall, it seems that the mean absolute error of even the neural network is quite high, even considering the volatility of the Ethereum.

F. The Bias-Variance-Noise Trade-off

As stated earlier, the dataset that was split before shuffled (time-series), had significantly worse performance on test sets due to the fact that the models were overfitting to the qualities of the older data, causing a variance issue. In order to combat this, it was decided that the non-time series (shuffled before split) dataset needed to be used, as the models performed better on the test data. This is because the training samples included data from all time periods, allowing the models to avoid the overtraining issue and be able to better predict prices on data from all time periods.

After dealing with the variance issue through dataset selection, it was found that the models still had a bias issue due to the relatively high Mean Absolute Error (lowest MAE was 35). Tuning hyperparameters was attempted to deal with this issue, however not much success was seen from this. It is possible that the bias issues come from our dataset; things such as smaller currencies features not tracking the market well, or a lack of relevant features that could have been added external to the Kaggle dataset.

Noise did not become a notable issue for us, due to the fact that the Mean Absolute Error of each model was so high that it was clear that the models were not overfitting to the training dataset, much less to the noise in the dataset itself. To identify noise issues, we should have seen the loss function reduce to a minimum and then begin to increase, but this behaviour was not observed on our models to a significant degree.

G. Goals for Further Study

For further consideration there must be further analysis on how the data is split and the models are trained. for a problem of this nature time is a very sensitive factor considering the rise of cryptocurrency in the recent years whether in value or in public exposure. a more time sensitive cross validation may be utilized such as the TimeSeriesSplit function from sklearn could yield over all better more realistic predictions for future data, rather than our manual method of splitting the data into old/current for training/validation.

In terms of feature selection, it would help to chart the prices for each of the currencies that we used in our dataset, and eliminate those that seem to be overly volatile compared to Ethereum, or are simply not following the market trends. Further features could be added to the dataset such as the Google Search Trends for cryptocurrency related terms, as they can be an indicator of interest in the market. Another possible added feature could be the cost of cryptocurrency transactions at each of the major exchanges, as people's likelihood to buy, sell, and use cryptocurrencies can depend on how much money they lose per transaction.

Along with re-evaluating hyperparameters in the existing models, Research into new models may help as well. Implementing a RNN using Long Short Term Memory (LSTM) layers may provide better accuracy than a standard DNN, as RNN's can take advantage of the time-series nature of our stock price data. Algorithms such as XGBoost should be tried as well. Ideally, once we get the best-performing model down to an acceptable Mean Absolute error, we can then begin to work on regularization in order to combat the increasing variance and over-fitting to noise in the dataset,

V. Implementation and Code

A. Technologies

Microsoft Excel and Google Sheets were used for dataset preparation and feature selection, as we chose to work with .CSV files for our data.

Google Colab was used to collaborate in real-time on the code for this project, as the colab machines make environment setup for SciKit-Learn and Tensorflow/Keras easy. Working in colab also helped with general compute time, as the machines offered were more powerful than our own.

SciKit-Learn was used for the linear regression and SVM models, while TensorFlow/Keras was used for the neural network.

Numpy and Pandas were used for dataset reading, with SKLearn and Tensorflow/Keras being used to create the splits and shuffle the data. Matplotlib was used for graphing.

References

[1] N. Sarju, G. Saade, and S. Al-Sailani, "Machine learning for ethereum price prediction," 2021. [Online]. Available: https://github.com/nirvansarju/Crypto-Prediction-ML

[2] D. Bullmann, J. Klemm, and A. Pinna, "In search for stability in crypto-assets: Are stablecoins the solution?" ECB Occasional Paper, 2019. [Online]. Available: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3444847

[3] Q. Ji, E. Bouri, C. K. M. Lau, and D. Roubaud, "Dynamic connectedness and integration in cryptocurrency markets," 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1057521918305416

[4] "Cryptocurrency ranking." [Online]. Available: https://coinmarketcap.com/

[5] M. Zanin, L. Alessandretti, A. ElBahrawy, L. M. Aiello, and A. Baronchelli, "Anticipating cryptocurrency prices using machine learning," 2018. [Online]. Available: https://www.hindawi.com/journals/complexity/2018/8983590/

[6] E. Akyildirim, A. Goncu, and A. Sensoy, "Prediction of cryptocurrency returns using machine learning," 2021. [Online]. Available: https://link.springer.com/article/10.1007%2Fs10479-020-03575-y

[7] "Kaggle cryptocurrency historical prices dataset." [Online]. Available: https://www.kaggle.com/sudalairajkumar/cryptocurrencypricehistory?select=coin_Ethereum.csv