Bitcoin Price Predictor

Foundations of Machine Learning
Project 1 - Final Report
DETI

Pedro Gonçalves 88859
Student
pedrog8@ua.pt

Rui Oliveira 89216
Student
ruimigueloliveira@ua.pt

Pétia Georgieva
Course Instructor
petia@ua.pt

***Abstract* - This document is the final report of the first project of the Curricular Unit of Foundations of Machine Learning.**
It contains the motivation for the topic in question and the respective state of the art.
Thus, a description of the data and its pre-processing is presented, such as the data in the form of graphs, the description of the machine learning models used, the training models, and finally the results and respective conclusions.

Keywords - Python; Machine Learning; Linear Regression; Cost Function; Gradient descent; Model selection and validation; Data visualization.

I. Introduction

A. Motivation

The world of Cryptocurrency and Blockchain is increasingly getting more exciting and rightfully so, as the prices of **Bitcoin** and other cryptocurrencies have reached the highest values ever.

Traditional trading bots used in the stock market today come with embedded machine learning-powered algorithms. Therefore, it's no surprise that **ML** techniques also lend a hand in building systems that trade the **cryptocurrency** markets.

B. Objective

Our goal is to use Machine Learning techniques and models to predict the future values of Bitcoin prices using a memory system that stores the daily value of Bitcoin.

II. State of the art

Cryptocurrency markets are particularly **hard to predict** because they are based on speculation, rather than fundamentals.
It's difficult for the market to price anything correctly.

What Determines Bitcoin's Price [1]:

- The supply of Bitcoin and the market's demand for it
- The cost of producing a bitcoin through the mining process
- The rewards issued to Bitcoin miners for verifying transactions to the blockchain
- The number of competing cryptocurrencies
- Regulations governing its sale and use
- The state of its internal governance
- News developments
- Influence of key investors

There are lots of bots that predict the price of Bitcoin with some precision.

The aim of our algorithms is "simply" to **predict future price values** and not automate purchases and sales, we also emphasize that our work is not related to the mining process algorithms either.

Here are some examples of the **main crypto bots** [2]:

- Best for Preset Strategies: **Coinrule**
- Best for High-Volume Traders: **Pionex**
- Best All Around: **Cryptohopper**

- Best for Automating Strategies: **Trality**

All these bots have custom strategies to predict the price of various cryptocurrencies including Bitcoin.

It should be noted that each **strategy** depends on what the investor intends to do. One of the main factors is to decide whether the investment is short term, medium term or long term.

In our case, the dataset is daily based, that is, there is a unique value for each day, which contributes to a more accurate prediction of an investment based on days.

III. Data description and Pre-processing

A. *File organization*

Our project is composed of three main directories:

- **Dataset**
  Here we can find all data both processed and completely raw.
  The data we chose for our algorithm uses the **price** of the beginning of each day (**00.01h UTC**) of the last **407 days** in euro. This data was taken from CoinMarketCap[2] and then pre-processed according to our needs.

- **Jupyter Notebooks**
  In this directory we find the main program where the machine learning algorithms are carried out and, consequently, the predictions are made.
  The entire reasoning and process for the final results are described here.

- **Parsers**
  In the Parsers directory we find all the necessary scripts for the pre-processing of data included in the Dataset.

B. *Normalization*

After some training with the price values we were using, we realized that the numbers were too big and the calculation times were starting to take more than they should.

To avoid those problems, we had to normalize the data. For that, we created a simple function. We take the biggest price value, in our case, 58283.8 and we check what the next multiple of 10000 is, in this case, 60000. Finally we divide all the data by 60000, meaning that all the data is now less than 1.

This was a big improvement since it's now easier to calculate the errors, and the calculation times are way faster.

IV. Data Visualization

count	407
mean	35482.75
std	12585.74
min	10070.59
25%	27755.58
50%	37287.16
75%	46656.33
max	58283.80

Table 1 - Some information about the dataset

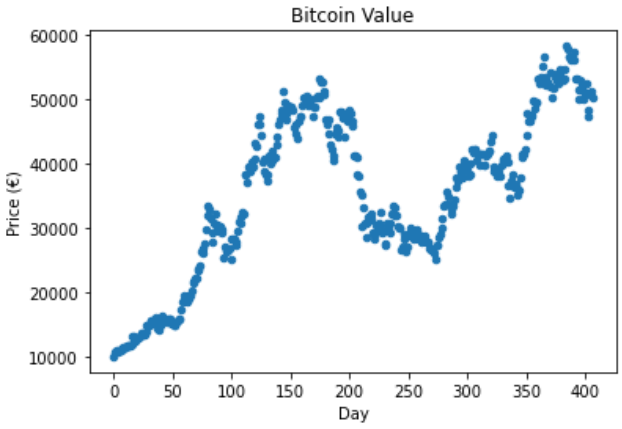

Figure 1 - Bitcoin price variation

V. Machine Learning Models

Our **initial approach** was a simple growing **linear regression**, since on the first day of our dataset the Bitcoin price was 10070.59€ and 407 days later it was at 50229.48€. This prediction could be useful for a long term prediction, as the price of Bitcoin will most likely keep increasing. However, **this model didn't satisfy our purpose.**

A. *Multivariate Regression*

It didn't take us long to realize that we needed a more detailed algorithm that predicted the value, not based on the day, but on the **price of the last few days**. Which means it would be a memory-based model.

In other words, we would have to apply a multivariate regression formula:

$h_\theta(x) = \theta_0 + \theta_1 x1 + \theta_2 x2 \ldots$

In our case the number of **parameters (thetas)** would be the number of **columns** we would use to predict the next value.

So we proceeded to divide the data in columns, in which the first column was the price of the N day ago, the second, the N-1 day ago, up until the current day.

At this moment the number of columns is not defined yet. Later, the model will be trained to decide the number of columns that allows predicting a future price with less error. (More about this in the "Model Training" chapter).

DAY N-2	DAY N-1	DAY N	DAY Y
10070.59 €	10803.79 €	10985.89 €	10901.23 €
10803.79 €	10985.89 €	10901.23 €	11050.01 €

Table 1 - Example of Dataset/btc_data.txt file organization

This way, **the last value of each line is the price we try to predict (Y), which will be the price of the last day, in the next line.**

B. Difference between prices

For our algorithm to be as precise as possible we implemented **two features**, the first one, as we've seen, was merely based on the price of the last few days. The second one was **based on** the **difference** of the prices of the last days. Therefore, for the example above (Table 1), the result would be:

Difference 1	Difference 2	Y
733.2 €	182.11 €	-84.66 €
182.11 €	-84.66 €	148.78 €

Table 2 - Example of Dataset/btc_diff.txt file organization

As we can see, the first value of the first line is the difference of the first two days (10803.79 - 10070.59) and so on.

On this feature, we decided to implement a different algorithm of normalization, for that we used a python library called **sklearn**, more specifically, the preprocessing method: "preprocessing.normalize([x_array], return_norm=True)", in which x_array is a numpy array of the difference values. The return_norm parameter means that we want to return the scalar value, so that we can multiply the result by it to get the final prices.

C. Final Model (Mean)

Finally, when both the features have made their predictions, we compute the mean of the two values and that's our final prediction.

VI. Model Training

A. Data Splitting

After parsing and normalizing our data, we proceeded to split the data in three files, one for the **training data**, approximately **60%,** one for the **validation data**, **20%** and the last for the **testing data,** the other **20%.**

When all our data was correctly divided, we started training the model. Our first objective was to find out the **best memory for our model (best number of columns)**. For this, we used 1 to 6 days of data to calculate the next.

B. Cost function (Regularized)

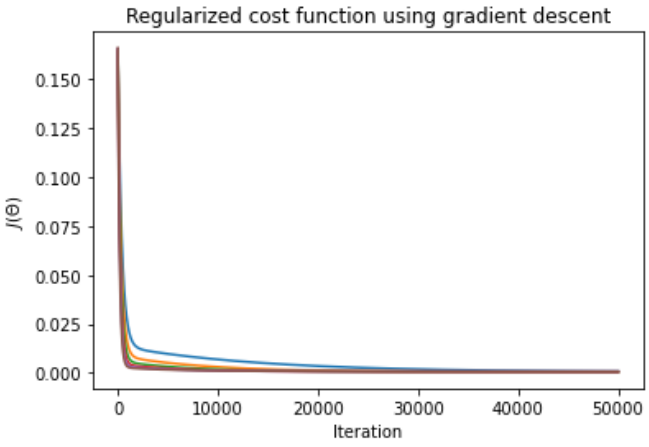

Figure 2 - Cost function trajectory over iterations graph

We used a function (gradientDescent) to calculate the thetas, which are the parameters of the linear function. We decided to make 50000 iterations since the thetas had clearly converged.

Afterwards we compared the results of the training with the validation ones and found the best number of columns to use.

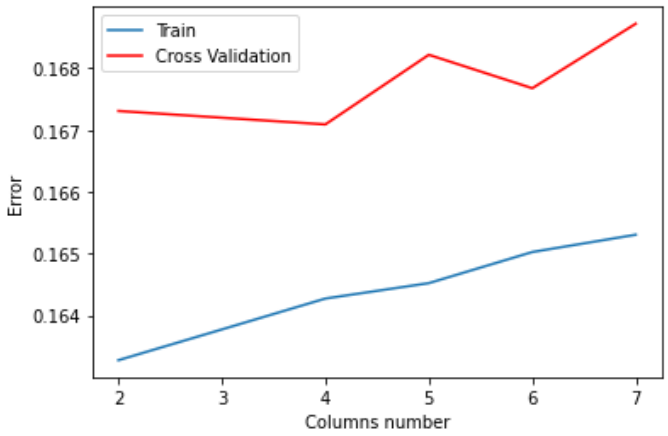

Figure 3 - Training and cross validation error variation depending on the number of columns/parameters in memory (p)

From the graph above we concluded the **best number of columns** to use were **4.** This means that in order to predict a value with the least error, **we need to use the values of the 3 previous days.**

Thus, the multivariate regression with **four** features is:

$$h_\theta(x) = \theta_0 + \theta_1 x1 + \theta_2 x2 + \theta_3 x3$$

VII. Results

A. *Predict Function*

After making our regression, we created the function to predict the value for the next day. This first function is rather simple, it receives two parameters, **x**, which is the Bitcoin price of the last 3 days, and the **theta** from our **regression function** and simply calculates the price for the **next day**.

The next function is a bit more complex, it lets us calculate the **price in N days**. For this, it receives the same two parameters plus the number of **days from now**, when we want to predict the price.

To do that we implemented recursivity, we call the function a number of times equal to the **days** variable, reducing it's value by 1 each time, so the first time we call it, we predict the price of the next day, the second time, we take that value and use it as the value of the last day, calculating the next day and so on, until we find the price of the day we are looking for.

```
# Predict from the previous prices
def predict(x, theta):
    h = np.dot(x, theta)
    return h[0]

def predictInNDays(x, theta, days):
    if days < 1:
        return x[0]
    elif days == 1:
        return predict(x, theta)
    else:
        final_x = predict(x, theta)
        # Each day becomes the day before
        for i in range(1, len(x) - 1):
            x[i] = x[i + 1]
        x[len(x) - 1] = final_x
        return predictInNDays(x, theta, days - 1)
```

Figure 4 -predict and predictInNDays functions

B. *Computing the regression function*

We proceeded to compute the values of theta for the regression we found. For our first feature, with normal price training, the results were the following:

h(x) = 0.0112 + 0.2545x + 0.318x2 + 0.4104x3

After having this regression, we calculated the **testing error.**

```
theta_ini = np.zeros((X.shape[1], 1))
theta_poly = gradientDescent(X, y,theta_ini,learn_rate,num_iter,0)[0]

pred_test = np.dot(Xtest, theta_poly)
Etest = 1/(2*mtest) * np.sum((pred_test - ytest)**2)
```

Figure 5 - Testing error example

For the error (**Etest**), we computed the **mean squared error** of the testing data, comparing the real bitcoin values (**ytest**) with our predictions (**pred_test**).

Our final **testing error** was **0.00052725.** We have to take in consideration that the values are normalized so this error is not something visible to our data, it's just a matter of comparison with the other features.

We also made a graphic to show the differences between our predictions and the correct Bitcoin price.

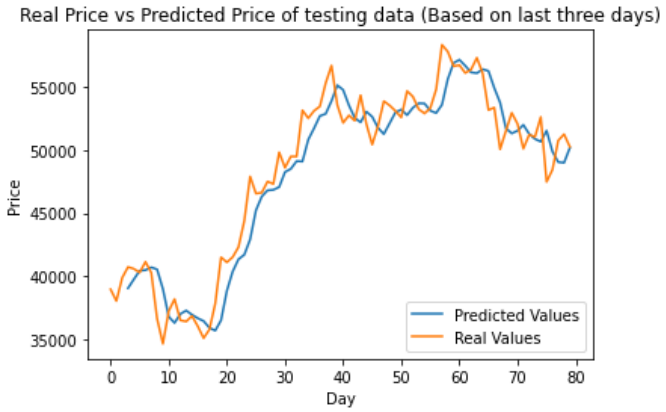

Figure 6 - Graphic that compares the real Bitcoin price with the predicted one from the testing dataset

In this graphic we can see that our predictions are not too far from the real price, even though in some days there's a big difference, which is normal considering the high volatility of the coin.

For our second feature, with price difference training, the results were:

$$h(x) = 0.0021 + 0.055x + -0.0448x2$$

We calculated the error as well, this time **Etest_diff**, resulting in **0.00038399**. This meant that, at least for the testing values, this feature performed better than the previous.

This feature provided us values much closer to the value of the previous day, which explains why the graphic below has such similar lines.

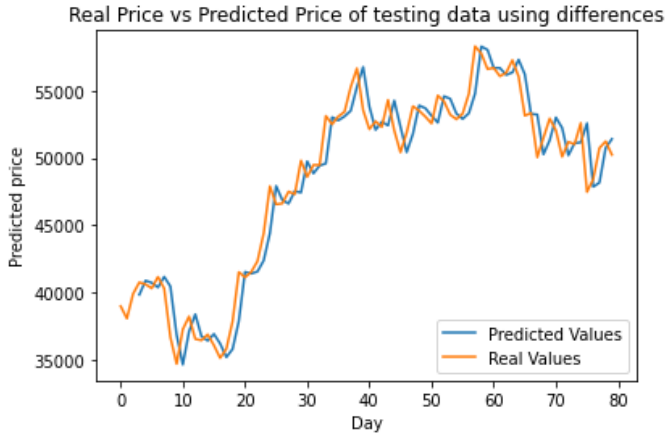

Figure 7 - Graphic that compares the real Bitcoin price with the predicted one from the testing dataset, using differences

Lastly, we decided to calculate the mean value of both the features to try to get the closest value possible. With that in mind, we simply computed the average of the two predictions and once again, found the corresponding error. With the same testing data, that error was **0.00040644**. To our surprise, the error was a bit higher than with the previous feature, although that can be explained by the fact that we have a relatively small testing data, and we expect this error to be smaller with a bigger dataset.

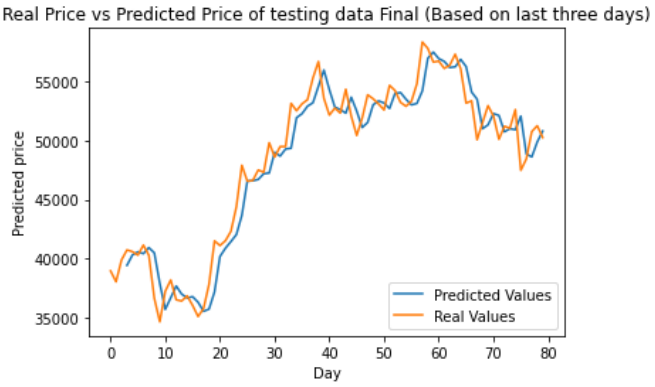

Figure 8 - Graphic that compares the real Bitcoin price with the predicted one from the testing dataset, using the mean of the features

VIII. Future Work

Considering the error on our predictions, it would be interesting to involve more features in our calculations, ideally our algorithm would have to be capable of interpreting news related to Bitcoin, and decide whether those news were positive or negative to the value of Bitcoin.

Furthermore, we would also want to access the Twitter API to take some of the tweets of famous people into consideration, as they can change the price abruptly, "Musk's latest bitcoin tweet (...) shot the price of the cryptocurrency up by nearly 10 percent"[4].

IX. Work Load

We believe that both elements of the group participated equally in the development of the project.

X. References

[1] How Much Is 1 Bitcoin? Factors of Bitcoin Pricing https://www.investopedia.com/tech/what-determines-value-1-bitcoin/

[2] 10 Best Crypto Trading Bots in 2021 • Benzinga Crypto https://www.benzinga.com/money/crypto-trading-bots/

[3] Bitcoin price today, BTC to USD live, market cap and chart CoinMarketCap https://coinmarketcap.com/currencies/bitcoin/historical-data/

[4] Chart: How bitcoin prices move with Elon Musk's tweets - Vox https://www.vox.com/recode/2021/5/18/22441831/elon-musk-bitcoin-dogecoin-crypto-prices-tesla