Crypto Trading & Transfer Learning - Project

ML/DL - Final report - Results & Conclusions

Friday, April 14th 2023

Team members

Arian NAJAFY ABRANDABADY	najafy.arian@gmail.com
Lucas RODRIGUEZ	lucasrodriguez.08@outlook.com
Bastien TRIDON	tridon.bastien1@gmail.com

Master in Quantitative Finance (M2QF)

Table of contents

We have used Python as the main numerical tool for all the required computations and to perform model simulations. In order to find our practical approach, please refer to the attached Jupyter Notebook file.

⟶ Developed & Tested using **Python 3.10.6**

⟶ GitHub repository : `https://github.com/lcsrodriguez/cryptotrading`

1 Introduction

1.1 Context

Cryptocurrencies have emerged as a significant financial asset class, attracting increasing attention from investors, traders and researchers. The rapid growth in the cryptocurrency market has led to the development of various Machine Learning (ML) and Deep Learning (DL) strategies predicting price movements and facilitating automated trading. This project focuses on implementing an ML/DL algorithm to trade Bitcoin and subsequently using transfer learning techniques to trade other cryptocurrencies. The project's objective addresses the need of effective trading strategies in the highly volatile and rapidly evolving cryptocurrency space. To achieve this, the project incorporates aspects of data preprocessing, visualization and model implementation ensuring that the developed strategies are robust, adaptable and applicable to various cryptocurrency markets.

1.2 Problem definition

The primary problem addressed in this project is to develop an effective ML/DL trading strategy for Bitcoin and extend its applicability to other cryptocurrencies using transfer learning techniques. The chosen dataset for this project is obtained from *G-Research Crypto Forecasting Kaggle competition* [1], which contains almost 2 million observations for Bitcoin with missing values that need to be treated (see below section 2.2). The specific sub-problems that need to be addressed are :

(1) Understanding and choosing an ML/DL trading strategy for implementation on Bitcoin

(2) Preprocessing and visualizing the given dataset, treating missing values and the time series nature of the data

(3) Implementing the chosen strategy on Bitcoin and backtest the model on at least 200 trades

(4) Employing transfer learning techniques to adapt the Bitcoin trading model for other cryptocurrencies thereby leveraging the knowledge gained from training the Bitcoin model

(5) Investigating the effect of transfer learning on the model's performance when applied to other cryptocurrencies

(6) Evaluating the performance of the developed strategies in terms of P&L, error analysis and comparison with existing strategies.

Visualization will play a crucial role in this project, both before and after the training process. This includes visualizing trends and candlesticks before training and profit & loss statements, errors and other relevant metrics after training. Additionally, it is essential to consider the time series nature of the data and augment the dataset with any necessary transformations or external data sources.

By addressing these sub-problems, the project aims to learn from and to contribute to the growing field of ML and DL-based trading strategies in the cryptocurrency market, providing valuable insights into the effectiveness of transfer learning techniques in this domain. Furthermore, the project seeks to provide a comprehensive understanding

1. Source : https://www.kaggle.com/competitions/g-research-crypto-forecasting

of the challenges and limitations associated with the development and implementation of such strategies, offering a solid foundation for future research and practical applications.

2 Data overview

2.1 Introduction of the given dataset

Our dataset is composed of

In the following, we will denote the following timeseries :

▷ O_t : the open price of the corresponding 1-min bin at instant t

▷ H_t : the high price of the corresponding 1-min bin at instant t

▷ L_t : the low price of the corresponding 1-min bin at instant t

▷ C_t : the close price of the corresponding 1-min bin at instant t

▷ V_t : the volume of trade of the corresponding 1-min bin at instant t

▷ I_t : the trading intensity represented by the **Count** column of the corresponding 1-min bin

▷ W_t : the VWAP (Volume-Weighted Average Price)

Proposition 1 (Characteristic price)**.** The last transation price [2] is the simplest definition of the price of a financial asset.

2. It is also known as the close price

	Asset_ID	Count	Open	High	Low	Close	Volume	VWAP
count	26755084.00	26755084.00	26755084.00	26755084.00	26755084.00	26755084.00	26755084.00	26755084.00
mean	6.31	303.15	1703.22	1706.95	1700.01	1703.22	292991.92	1703.20
std	4.09	882.28	7137.28	7147.04	7127.61	7137.29	2367241.04	7137.22
min	0.00	1.00	0.00	0.00	0.00	0.00	-0.37	-799.75
25%	3.00	21.00	0.28	0.28	0.28	0.28	138.42	0.28
50%	6.00	71.00	15.39	15.41	15.37	15.39	1272.08	15.39
75%	10.00	243.00	247.58	248.21	246.98	247.57	30200.44	247.57
max	13.00	165016.00	68986.12	69024.20	68734.00	68973.56	759755403.14	68894.05

TABLE 1 – Statistical analysis of the whole pre-processed dataset

2.2 Pre-processing steps

After importing the three datasets, we have observed that the two train sets were complementary and could be added one after the other.

Regarding the test set (containing less than 60 rows), we have spotted that each test observation is already contained in the final train set ; we have finally decided to remove it. Now, we can rename our final train set, which is the concatenation of the two initial train sets, as our new dataset.

Remark 1. We will perform our main train/test/validation sets split in the coming sections.

We now have a clean data framework to work on.

2.3 Data visualization

One may for instance plot the evolution of some metrics (ex : Close price & Trading activity).

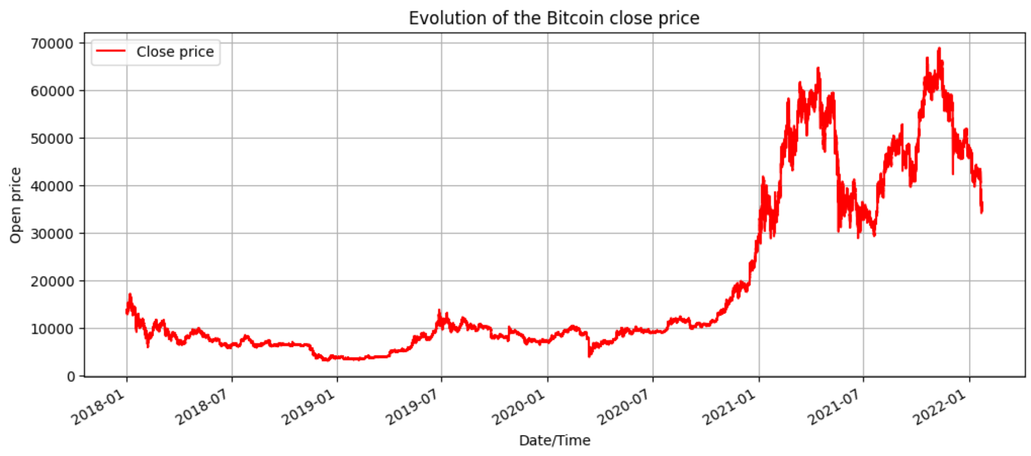


FIGURE 1 – Plot of the BTC close price

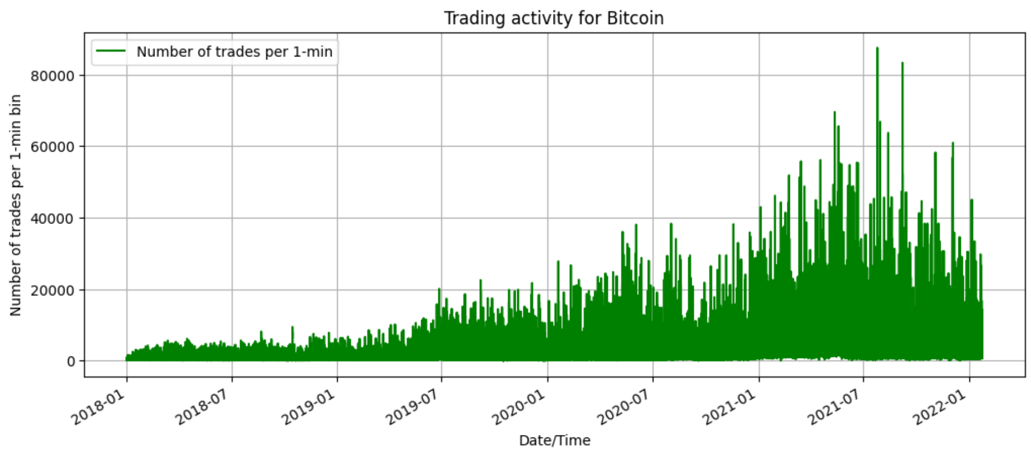


FIGURE 2 – Plot of the BTC trading intensity (number of trades per 1-min bin

We can also conduct data visualization on correlation.

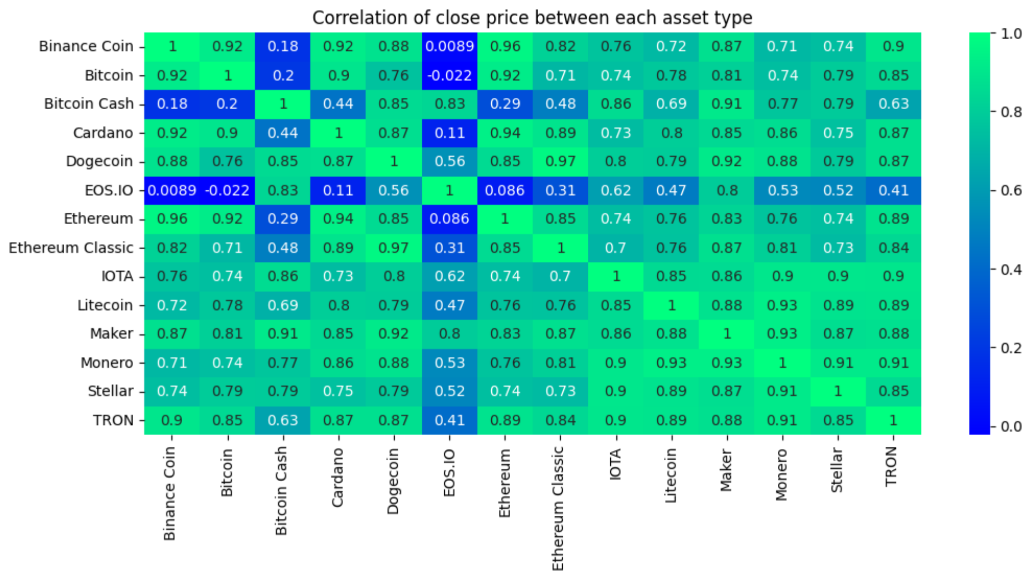


FIGURE 3 – Correlation matrix of the close price for each asset

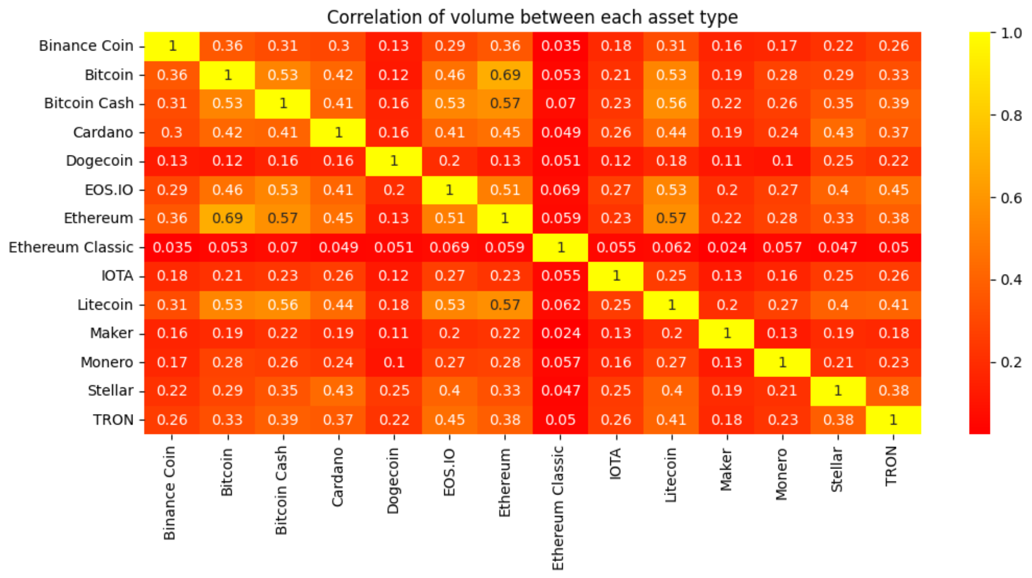


FIGURE 4 – Correlation matrix of the volume for each asset

Finally, we can build a *candlestick representation* of our dataset, which is very common in financial engineering.

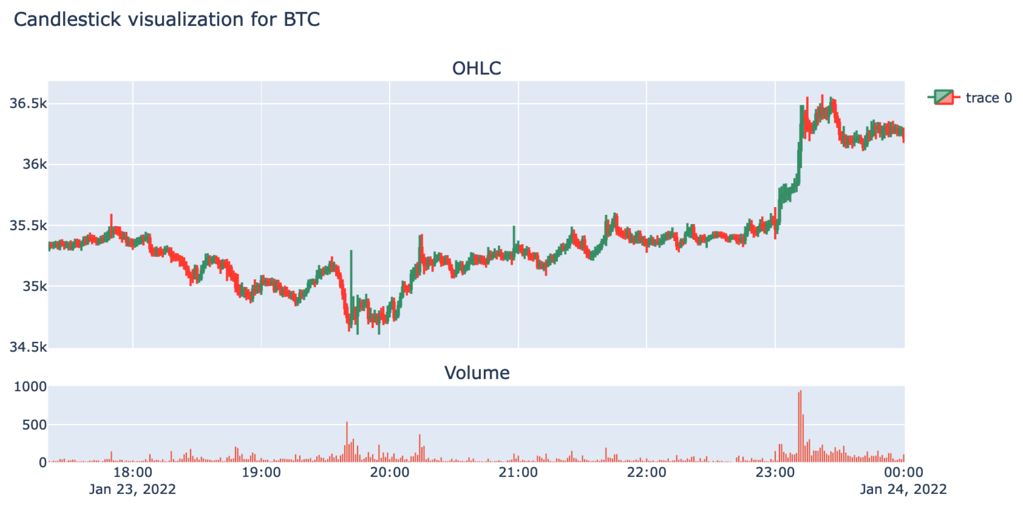


FIGURE 5 – Candlestick representation for BTC

Remark 2 (Heavy computations)**.** We are forced to truncate our dataset for the candlestick representation, since the **Plotly** library tends to have difficulties to handle it.

However, we might be using the **Plotly-Resampler**[3] to visualy aggregate our dataset to render it in a better way on the UI.

2.4 Aggregation & Composite indicators

2.4.1 Additional datasets (Coinbase, Binance, FTX, ...)

One could also add additional datasets and merge them with the previously sampled dataset. However, this could create new NaN as external datasets and/or historical data from online APIs can be available on 24/5 scale for instance (weekends, nights, trading opening/closing, ...).

2.4.2 Additional financial indicators (technical analysis)

We have decided to aggregate our resampled dataset by adding new technical analysis indicator (see pre-processing step on the Jupyter Notebook).

- ▷ RSI
- ▷ Bolling bands (Upper, Medium & Lower)
- ▷ Open - Open
- ▷ Close - Close
- ▷ Standard Deviations
- ▷ Several moving averages

3. Documentation : `https://github.com/predict-idlab/plotly-resampler`

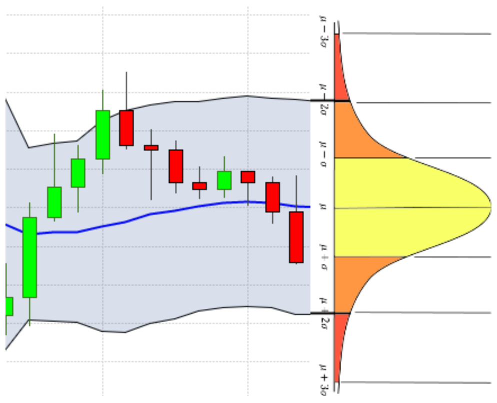


FIGURE 6 – Graphical representation (sketch) of the Bollinger bands
Source : Wikipédia

3 Price evolution prediction using ML/DL models

3.1 Binary classification of future price movement

Let us discuss the binary classification of future price movement in the context of machine learning trading strategy. We will first explain the concept of target and close price movement and then discuss why the close price is chosen in this particular strategy.

Main target In our machine learning code, the target variable represents the future price movement of the financial asset, specifically, whether the asset's price will increase or decrease. To create this binary classification, the target variable is set to 1 if the close price of the asset at time $t+1$ is greater than the close price of the asset at time t and 0 otherwise. Thus the target variable can be defined as :

$$\textbf{target}_t = \begin{cases} 1 & \text{if } C_{t+1} > C_t \\ 0 & \text{otherwise} \end{cases}$$

The close price is chosen as the primary input for our trading strategy because it represents the last transaction price of a financial asset within a specific time interval and thus reflects the final market consensus on the value of the asset at the end of the trading period. This choice is directly linked to the fact that we have only been provided by OHLCV data instead of tick data.

3.2 Ternary classification

In the context of Bitcoin price movement, a ternary classification such as *UP/-CONSTANT/DOWN* is not really relevant. A binary classification already captures the essential information needed for the trading strategy : whether the price of Bitcoin will increase or decrease. The inclusion of a *CONSTANT* category would only serve to introduce additional complexity into the model without providing any significant benefit in terms of predictive accuracy since in the case of Bitcoin the price is highly volatile and can experience significant fluctuations within a short period.

3.3 Regression on log-returns prediction

Another idea would have been to directly predict the future price fron the given underlying asset (here : Bitcoin). However, this raises two challenges : first, we need to identify and benchmark a relevant regression model to perform this predictive task. Another issue might be related to the extensive seek of precision. While in the first two cases, we are looking for a discrete prediction over a 2-elements or 3-elements set, here, we are trying to predict the price on $\mathbb{R}$ (with a 10^{-2} precision for instance).

The more we will choose to introduce precision in our machine learning model, the more likely prediction errors will occur. This definitely disqualifies this research track.

Conclusion Therefore, a binary classification, that only focuses on capturing the directional movement of Bitcoin's price is sufficient for this particular strategy.

We will test and benchmark several robust machine and deep learning models to implement our binary classification.

4 Trading strategies for BTC-USD pair

4.1 A first trading algorithm

4.1.1 Principle

Our first trading algorithm (Algorithm 1) uses predicted signals to make buy and sell decisions. The algorithm is implemented in Python and it leverages the Plotly library for visualizing results as charts.

The function `trading_algorithm` takes several arguments :

- ▷ `Y_pred` : An array containing predicted signals. A value of 1 indicates a buy signal, whereas a value of 0 indicates a sell signal
- ▷ `X_test` : A DataFrame containing the market data, including columns for Open, High, Low and Close prices
- ▷ `transaction_cost` (optional) : A float representing the transaction cost as a percentage of the trade value [4]
- ▷ `candlestick_chart` (optional) : A boolean value indicating whether to plot a candlestick chart of the market data [5]
- ▷ `candlestick_chart_daily` (optional) : A boolean value indicating whether to resample the market data to daily frequency before plotting the candlestick chart [6].

The function initializes the algorithm with a starting cash balance of 100,000 and no shares. It then iterates through each prediction in `Y_pred` and takes the following actions based on the predicted signal and the current portfolio state :

- ▷ If the prediction is a buy signal (1) and there are no shares in the portfolio, the algorithm buys one share at the closing price, adjusted for transaction costs

4. Default value is 0.005, or 0.5%.
5. Default value is `False`.
6. Default value is `False`.

- ▷ If the prediction is a sell signal (0) and there are shares in the portfolio, the algorithm sells one share at the closing price, adjusted for transaction costs
- ▷ If the prediction is a buy signal (1) and there are shares in the portfolio, the algorithm continues to hold the shares.

After each action, the algorithm calculates the current profit and loss (PnL) and appends it to a list. The PnL is calculated as the sum of the cash balance and the current value of the shares. Once all predictions have been processed, the function plots the PnL over time using Matplotlib.

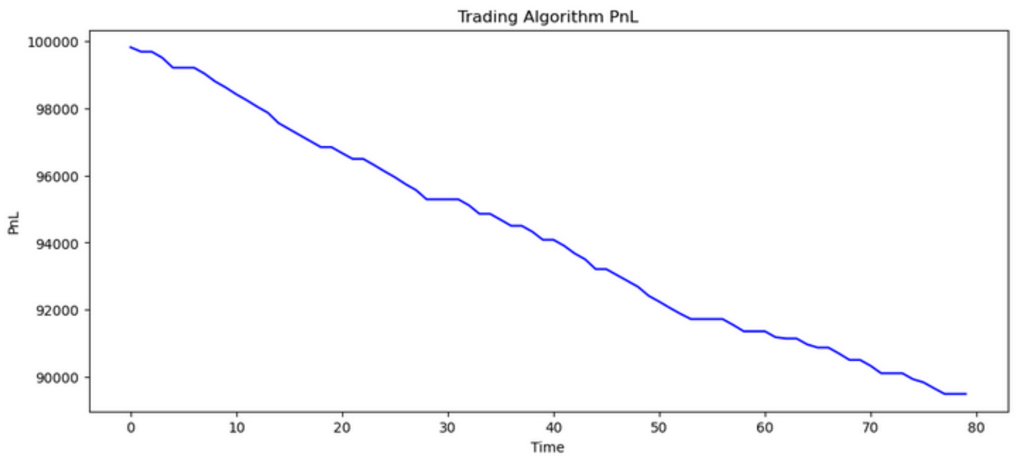
The algorithm is capable of generating candlestick charts with buy and sell markers. This functionality is available in two modes : daily and standard. The daily mode is particularly useful when working with large inputs, as it helps to avoid potential crashes in the Plotly library (cf. Remark 2). On the other hand, the standard mode is more appropriate for smaller inputs. Both modes provide a valuable tool to visualize market trends and decisions. If the `candlestick_chart` argument is set to `True`, the function will plot a candlestick chart of the market data using the Plotly library. If the `candlestick_chart_daily` argument is set to `True`, the market data will be re-sampled to daily frequency before plotting.

Algorithm 1 Trading Algorithm

```
Require: $Y\_pred$, $X\_test$, $transaction\_cost = 0.005$, $candlestick\_chart = False$,
    $candlestick\_chart\_daily = False$
 1: $shares \leftarrow 0$
 2: $cash \leftarrow 100000$
 3: $pnl \leftarrow []$
 4: $buy\_signals \leftarrow []$
 5: $sell\_signals \leftarrow []$
 6: for $i \in \{0, \ldots, \text{length}(Y\_pred) - 1\}$ do
 7:     if $Y\_pred[i] = 1$ and $shares = 0$ then
 8:         $shares \leftarrow shares + 1$
 9:         $cost \leftarrow X\_test["Close"][i] \times (1 + transaction\_cost)$
10:         $cash \leftarrow cash - cost$
11:         $buy\_signals.append(i)$
12:     else if $Y\_pred[i] = 0$ and $shares > 0$ then
13:         $shares \leftarrow shares - 1$
14:         $revenue \leftarrow X\_test["Close"][i] \times (1 - transaction\_cost)$
15:         $cash \leftarrow cash + revenue$
16:         $sell\_signals.append(i)$
17:     end if
18:     $pnl.append(shares \times X\_test["Close"][i] + cash)$
19: end for
20: Plot $pnl$ with labels
21: if $candlestick\_chart$ then
22:     if $candlestick\_chart\_daily$ then
23:         Resample $X\_test$ to daily frequency
24:         Plot daily candlestick chart with buy and sell markers
25:     else
26:         Display performance note for large datasets
27:         Plot standard candlestick chart with buy and sell markers
28:     end if
29: end if
```

4.1.2 Results with Logistic regression

Let us analyze the output (Figure 7) of our first trading algorithm in the case of Logistic regression predictions.



```
Please note that plotly may experience performance issues or crashes when handling large input datasets. If you encounter any issue
s, you may need to reduce the size of your input data or use data downsampling techniques to improve performance.
```

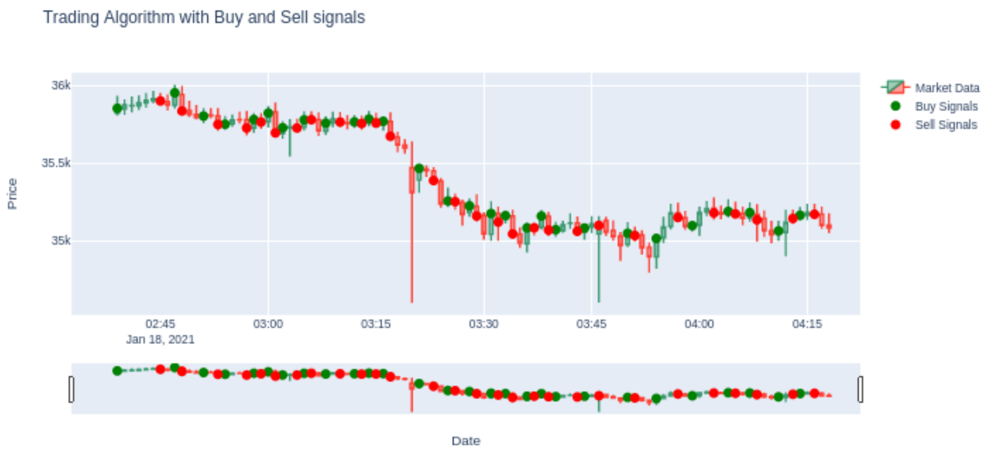


FIGURE 7 – Trading algorithm output for Logistic Regression

As expected, the logistic regression model is not performing well in predicting the price movement of Bitcoin. Its accuracy score of $\approx 50\%$ implies that this model is not better than flipping a coin and is essentially randomly guessing the price movement half of the time. Trading based on such predictions is likely to result in losses over time.

Additionally, our first trading algorithm has some limitations that could impact its performance. For example, it only buys or sells one share at a time which may not be optimal for capturing larger price movements. Furthermore, the algorithm assumes a fixed transaction cost which may not be accurate in practice.

To improve the performance of the algorithm, we could consider more sophisticated models such as neural network and incorporate more relevant features into the model such as additional financial and technical indicators.

4.2 Refined model using Incremental learning and XGBoost

For an initial cash pool of $100,000.00$$, we have observed a clear decreasing linear trend on the P&L profile of the first trading strategy. To avoid that, we will introduce a new model using the XGBoost machine learning.

Dataset First, we needed to adapt our current time-series dataset into a supervised machine learning dataset. To do so, we have to apply the rolling window[7] procedure involving the shifts of several features of interests in order to explain the main target at time t as a combination of knowledge from the values of the previous instants $t-1$, $t-2$, $\ldots$, $t-H$ where $H > 0$ denotes the maximal past-shifting index. In our implementation, we have decided to take $H = 20$; indeed, after several experiments, it was the value with the best performances.

In addition, first of all, we have resampled the dataset into a 30-min bins dataframe. This can be explained in several papers[8] by the fact that ultra-high frequency bar data aren't good indicator of price significance as microstructure noise and market-making behaviors could affect the overall signals.

Model Next issue was the lack of retro-action or feedback loop in our previous model. In fact, we will predict at each time t the next 10 target values, compare them with the current target values and make performance analysis.

Then, the next iteration, for time $t+1$, will take into account the value of the target that was captured on the financial market at time t.

In mathematical and probabilistic terms, this can be described as a *growing filtration* $\mathcal{F}_t$.

At each iteration (each time $t > 0$), we are storing on the disk, the current version of the model. Then, at the iteration $t+1$, we are retrieving the knowledge from the model at time t and we apply a partial fitting over the new datasets ; this is the exact of incremental learning using an anchored walk-forward (AWF) prediction.

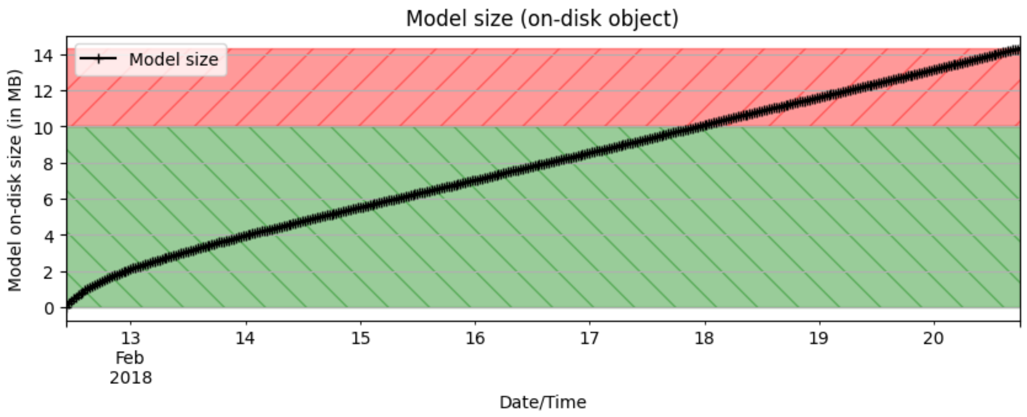


FIGURE 8 – Evolution of the size of the model (in MB)

Remark 3. Thanks to these intermediary on-disk storages, we are able to retrieve them for a future transfer learning application (see below).

7. Reference : `https://machinelearningmastery.com/convert-time-series-supervised-learning-probl`

8. Reference : `https://jfin-swufe.springeropen.com/counter/pdf/10.1186/s40854-020-00217-x.pdf` (Page 6)

However we have two different kinds of walk-forward procedure :

(1) the anchored (WFA) using for each iteration, the all past knowledge from 0 to $t-1$

(2) the unanchored (WFU) using for each iteration, the past knowledge from $t-\alpha$ to $t-1$

The second option allows us to drastically reduce the memory cost for each model file but also the computational time [9]

Finally, time-series K-Fold and GridSearch were experimented but due to the incremental learning procedure, currently implemented, we have decided to remove them because of the huge computational time, implied by those two new procedures.

Strategy The strategy is simple : at each time t, we are counting the number of prediction of up and down movements in the near future (next ten iterations).

According to empirical threshold, we then decide to take a long or short position.

- ▷ The backtest results are shown in the following section
- ▷ The prediction results analysis has been performed in the Jupyter Notebook (see GitHub repo).

4.3 Backtesting & Result analysis

Backtesting is essential to machine learning trading strategies.

Definition 1 (Backtesting)**.** Backtesting is a technique used to evaluate the performance of a trading strategy by simulating its performance using historical data. It can be used to evaluate the performance of a machine learning trading strategy by simulating its performance using historical data.

This process involves feeding historical market data into the algorithm and then using the algorithm to make trades based on that data. The results of these simulated trades are then compared to the actual historical performance of the market to determine the accuracy of the trading strategy.

Backtesting is an important step in the development of a trading strategy, as it allows for the assessment of the strategy's performance and the identification of any issues before real money is invested. However, past performance does not guarantee future results, so backtesting should be used together with other evaluation methods, such as forward-testing and cross-validation.

To perform our backtest, we have used an external library **Backtesting.py** [10] [11] which offers an extensive object-oriented architecture separating the building blocks of a trading engine : **Strategy**, **Backtest**, **Order**, **Trade** & **Position**.

Remark 4 (Comparative study against other strategies)**.** Thanks to the modularity of the **Backtesting.py**, we are able to define and run various trading strategies on the same execution framework. By default, during a backtest, we are comparing the P&L and the returns of our strategy

9. More than 3 hours and 25 GB of model data (binary files).
10. Documentation : `https://kernc.github.io/backtesting.py/doc/backtesting/`
11. Official website : `https://github.com/kernc/backtesting.py`

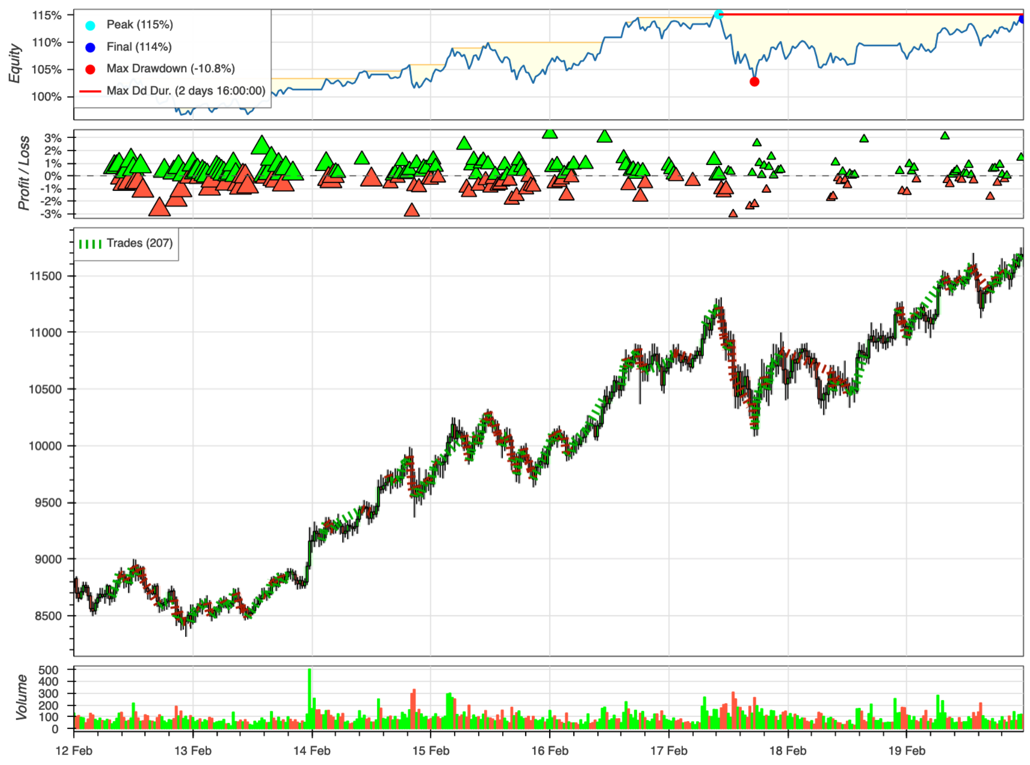


Figure 9 – Backtesting results

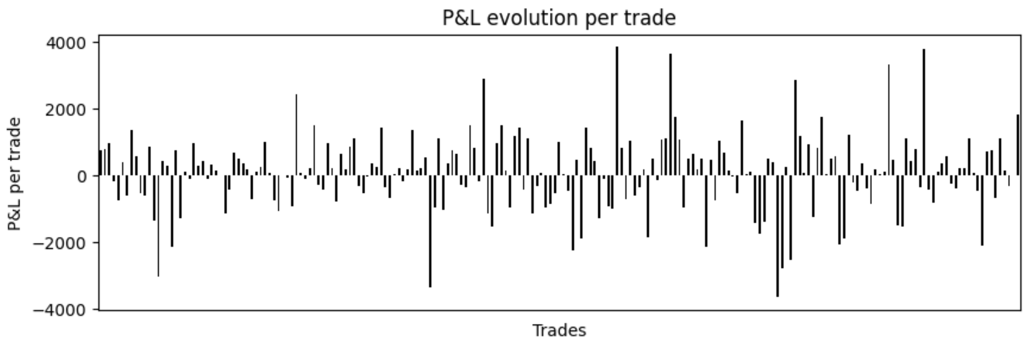


Figure 10 – Individual P&L for each trade

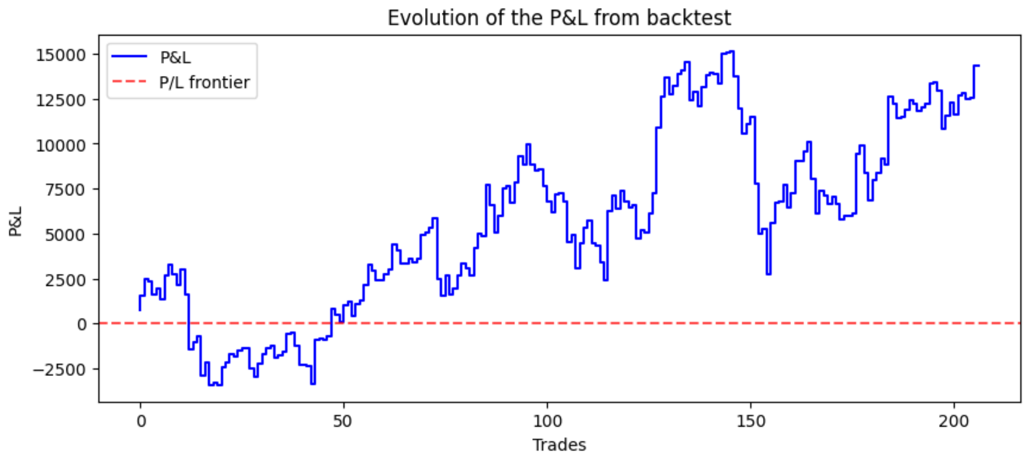


FIGURE 11 – Final P&L evolution (as cumulative sum)

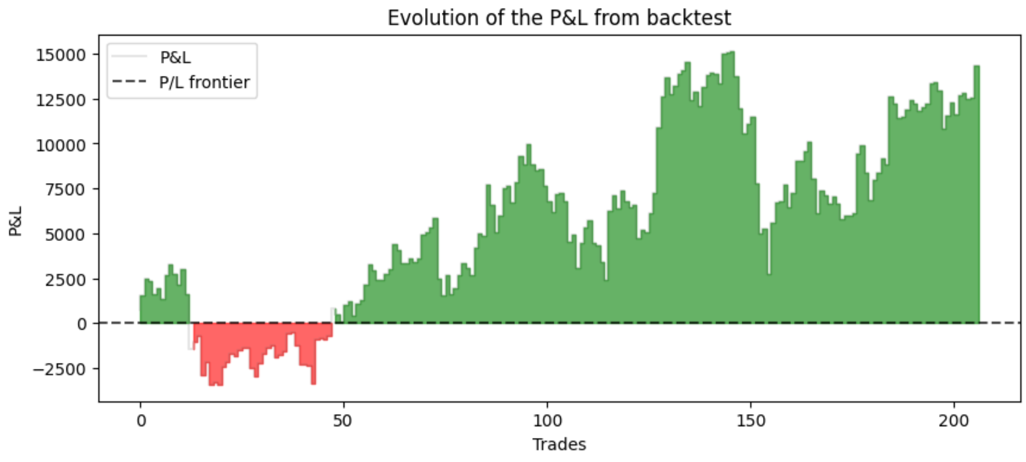


FIGURE 12 – Final P&L evolution (as cumulative sum)

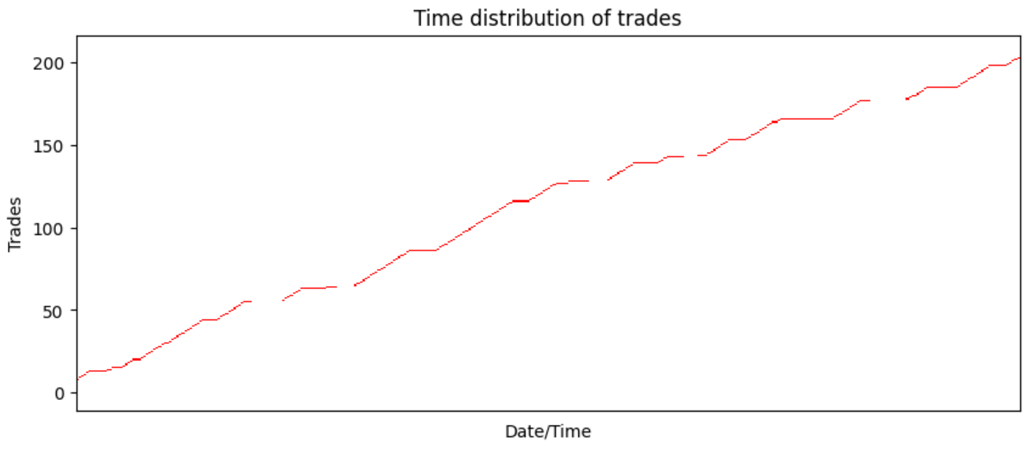


FIGURE 13 – Organization of each trade over time

Indicators	Value
Start	2018-02-12 10 :30 :00
End	2018-02-20 18 :00 :00
Duration	8 days 07 :30 :00
Exposure Time [%]	85.75
Equity Final [$]	114325.535918
Equity Peak [$]	115151.981125
Return [%]	14.325536
Buy & Hold Return [%]	32.239443
Return (Ann.) [%]	7527.131325
Volatility (Ann.) [%]	9948.457655
Sharpe Ratio	0.756613
Sortino Ratio	122.700239
Calmar Ratio	697.296602
Max. Drawdown [%]	-10.794734
Avg. Drawdown [%]	-3.890339
Max. Drawdown Duration	2 days 16 :00 :00
Avg. Drawdown Duration	0 days 17 :54 :00
# Trades	207
Win Rate [%]	58.937198
Best Trade [%]	3.31398
Worst Trade [%]	-3.038891
Avg. Trade [%]	0.05785
Max. Trade Duration	0 days 10 :30 :00
Avg. Trade Duration	0 days 00 :49 :00
Profit Factor	1.194854
Expectancy [%]	0.062497
SQN	0.867856

TABLE 2 – Resulting indicators from the given simulation

Conclusion From (2), the P&L (Profit & Loss) is greater than 14% on the studied time period, which is higher than the S&P 500 annual performance.

In addition, one may study the Sharpe ratio's value which is $\sim 0.76 < 1$; this means that this strategy takes more risk for the level of realized profit.

Finally, we highlight that the number of winning trades are around 60% which seems to be a good ratio, witness of a well-designed strategy.

4.3.1 Re-writing of the library

As we have decided to first make and store the predictions for the near future, then to iterate over our results to apply our trading strategy, we need to give access the strategy object to the predictions.

As the library **Backtesting.py** does not provide external getters and setters, we have decided to overwrite a new Backtest class, inheriting from the previous one and adding the prediction collections.

Remark 5. In addition, several on-disk saving are performed to store the trades table, the relevant informations on the results and the plots.

4.4 Further developments & Extensions

- ▷ One could implement another benchmark strategy, widely-used by private investors nowadays : the **Dollar Cost Average** (DCA) strategy, consisting in investing passively small portion of the cash pool to be invested, at frequent times (each month, each week, ...) to attempt to capture the positive market movement instead of investing all the money at one unique time t.
- ▷ It can also be relevant to add limit on the number of short and longs operations to perform over a given time horizon in order to limit the loss related to the exchange fees.
- ▷ Following the previous point, it might also be a good extension to add the possibility to define proportional or time-varying fees to make our model more accurate and best suited to the real-world.

5 Application to Ethereum using Transfer Learning

We have decided to apply our trading strategy on ETH as we observe a strong correlation on the trading activity and close price between Ethereum and Bitcoin (see (3)).

Results & Remarks We can highlight the greater predictive power on ETH dataset from this transfer learning by comparing the two confusion matrices.

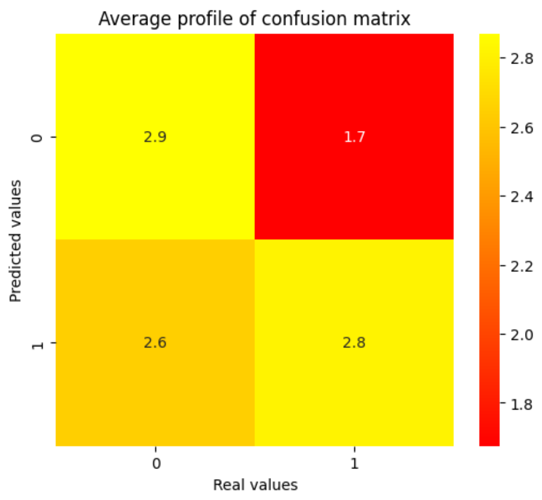


FIGURE 14 – Average confusion matrix for ETH binary classification

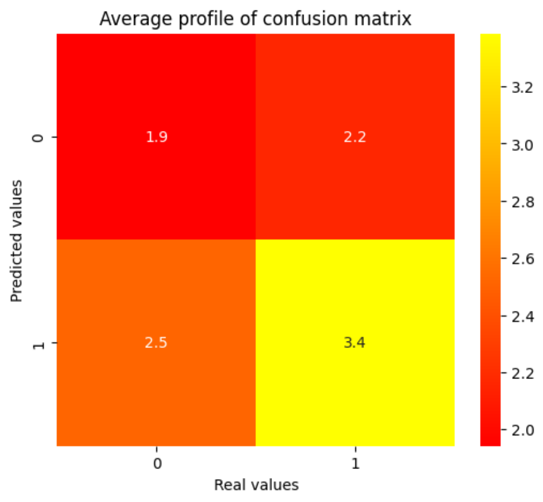


FIGURE 15 – Average confusion matrix for BTC binary classification

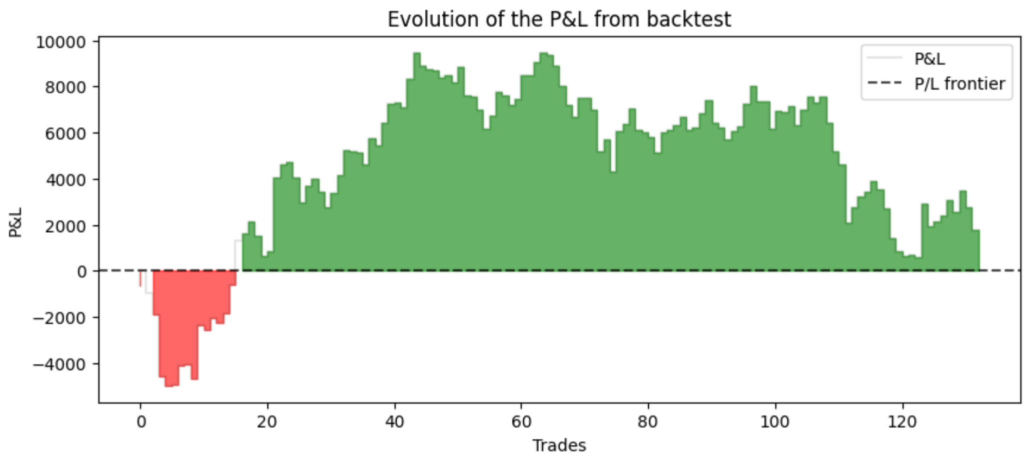


FIGURE 16 – Final P&L evolution (as cumulative sum) for Ethereum

Conclusion The P&L is 1.749% and the Sharpe ratio is $0.36 < 1$, lower than the one for the Bitcoin case, indicating that the strategy takes a greater risk over ETH than with BTC.

6 Conclusion

In conclusion, we have investigated some challenges of Machine Learning and Deep Learning in the fields of finance and algorithmic trading.

The use of bar data (with aggregation functions) can reduce precision and embedded information due to its lower granularity.

Our study has demonstrated the importance of pre-processing to better handle data and provided a state-of-the-art review of current simple trading strategies. The implementation and backtesting of a strategy on Bitcoin as well as transfer learning to Ethereum have been explored.

For future extensions, the use of L1/L2 data (trades and quotes) for greater granularity is suggested, as bar data often overestimates the profits generated by a strategy. Furthermore, introducing parallelism or multithreading to speed up fitting jobs and employing cloud computing instances with larger CPU cores (e.g. AWS Lambda, SageMaker) can enhance computation capabilities.

Ultimately, the extension to real-time world applications through platforms such as Binance and FTX using real-time WebSocket APIs can be studied as a final thought.