VNKR TECHNOLOGIES S.L.

VANKOR ALGORITHM MEASURING MARKET RISK

Version 1.0

BTC Volatility

May 2023

Contents

1 Introduction

This document explains the process followed for the creation of the Vankor algorithm. This algorithm aims to predict market risk based on both traditional and onchain features as well as new ones created with financial knowledge and reasoning.

Technologically speaking, three different approaches are proposed to solve the problem: solution with classical algorithms for time series (GARCH, ARIMA), solution with regression algorithms based on tree assembly (bagging/boosting) and deep learning algorithms. The classical approach is applied only with the time series of the target variable, without complementary series.

The first two types of models have predicted interesting results, nevertheless we have found the lowest error in deep learning models such as the LSTM. Specifically, the best model found in this study is a **bidirectional LSTM**, which we will explain below.

2 Data Gathering

As mentioned in the introduction, we have collected information both offchain and onchain from different sources.

A summary with an explanation for each feature can be seen in the following tables.

2.1 Traditional Data

Variable	Real Name	Description	Source	On/Off Chain
High	High	Daily highest price	Trading View	OffChain
Low	Low	Daily lowest price	Trading View	OffChain
Open	Open	Daily open price	Trading View	OffChain
Close	Close	Daily close price	Trading View	OffChain
Upper Bollinger	Upper Bollinger	The upper band is calculated by taking the moving average and adding twice the daily standard deviation to that amount	Trading View	OffChain
Lower Bollinger	Lower Bollinger	The lower band is calculated by taking the moving average minus two times the daily standard deviation	Trading View	OffChain
Upper Donchian	Upper Donchian	The upper band marks the highest price of a security over N periods. Donchian Channels are three lines generated by moving average calculations that comprise an indicator formed by upper and lower bands around a midrange or median band.	Trading View	OffChain
Lower Donchain	Lower Donchain	The lower band marks the lowest price of a security over N periods	Trading View	OffChain
Red Arrow	Red Arrow	Buy signal on William Fractals Indicator	Trading View	OffChain
Green Arrow	Green Arrow	Sell signal on William Fractals Indicator	Trading View	OffChain
Value_Uspce	Personal Consumption Expenditures	A measure of the prices that people living in the United States, or those buying on their behalf, pay for goods and services	Trading View	OffChain
HL_sprd	High and Low Spread	High and Low spread	Trading View	OffChain
Value_SP500	SP 500 Index	Is *a market-capitalization-weighted index of 500 leading publicly traded companies in the U.S* .	Trading View	OffChain
Detrended Price Oscillator	Detrended Price Oscillator	Indicator in technical analysis that attempts to eliminate the long-term trends in prices by using a displaced moving average so it does not react to the most current price action.	Trading View	OffChain
Value_USunem	Unemployment Rate	Variation percentage of US unemployment	Ycharts	OffChain
US_bank_idx	US Bank Index	Is designed to measure the performance of the US banking sector.	Trading View	OffChain
Value_EFF	Efective Federal Funds Rate	Volume-weighted median of overnight federal funds transactions reported in the FR 2420 Report of Selected Money Market Rates	Ycharts	OffChain
Value_CPI	Consumer Price Index	Measures the overall change in consumer prices based on a representative basket of goods and services over time	Trading View	OffChain
Value_M3	Money Supply (M3)	Broad money (M3) includes currency, deposits with an agreed maturity of up to two years, deposits redeemable at notice of up to three months and repurchase agreements, money market fund shares/units and debt securities up to two years	Ycharts	OffChain
RSI	Relative Strength Index	A momentum oscillator that measures the speed and change of price movements	Trading View	OffChain
Variation_UStotal	Non-farm payroll	measure of the number of U.S. workers in the economy that excludes proprietors, private household employees, unpaid volunteers, farm employees, and the unincorporated self-employed	Ycharts	OffChain
CO_sprd	Open Close Spread	Open-Close Spread	Paper	OffChain
Put-call Ratio	Put-call Ratio	The Options Volume Put/Call Ratio shows the put volume divided by call volume traded in options contracts in the last 24 hours.	GlassNode	OffChain

2.2 Onchain Data

Variable	Real Name	Description	Source	On/Off Chain
hash_rate	Hash Rate	The average estimated number of hashes per second produced by the miners in the network	GlassNode	OnChain
active_addre	Active Addresses	The number of unique addresses that were active in the network either as a sender or receiver. Only addresses that were active in successful transactions are counted.	GlassNode	OnChain
rev_all_miners	Miner Revenue	The total miner revenue, i.e. fees plus newly minted coins.	GlassNode	OnChain
tran_num	Transactions Number	The total amount of transactions. Only successful transactions are counted.	GlassNode	OnChain
size_tran	Transactions Size	The total size of all transactions within the time period (in bytes).	GlassNode	OnChain
created-utxos-btc	Created UTXOS	The number of created unspent transaction outputs.	GlassNode	OnChain
spent-utxos-btc	Spent UTXOS	The number of spent transaction outputs.	GlassNode	OnChain
price-drawdown	Price Drawdown from ATH	The percent drawdown of the asset's price from the previous all-time high.	GlassNode	OnChain
Aggregate security spend (Thermocap)	Thermocap	It is the aggregated amount of coins paid to miners and serves as a proxy to mining resources spent	GlassNode	OnChain
Balance Price	Balance Price	It is the difference between Realized Price and Transfer Price. Transfer Price is the cumulative sum of Coin Days Destroyed in USD, adjusted by circulating supply and total time since Bitcoin's inception. Balanced Price attempts to detect major cycle bottoms	GlassNode	OnChain
stock-to-flow	Stock to flow ratio	The Stock to Flow (S/F) Ratio is a popular model that assumes that scarcity drives value. Stock to Flow is defined as the ratio of the current stock of a commodity (i.e. circulating Bitcoin supply) and the flow of new production (i.e. newly mined bitcoins). Bitcoin's price has historically followed the S/F Ratio and therefore it is a model that can be used to predict future Bitcoin valuations.	GlassNode	OnChain
realized-price-usd	Realized Price	Realized Price is the Realized Cap divided by the current supply.	GlassNode	OnChain
Balance on Exchanges	Balance on Exchanges	The total amount of coins held on exchange addresses	GlassNode	OnChain
Block Height	Block Height	The total number of blocks ever created and included in the main blockchain	GlassNode	OnChain
Circulating Supply	Circulating Supply	The total amount of all coins ever created/issued	GlassNode	OnChain
Delta Cap	Delta Cap	It is the difference between Realized Cap and Average Cap, where Average Cap is assumed to be the life-to-date moving average of Market Cap	GlassNode	OnChain
Investor Capitalization	Investor Capitalization	It is the difference of Realized Cap and Thermocap. It discounts the capital paid to miners from the market's general cost basis, serving as a bottom indicator in bear cycles	GlassNode	OnChain
Median Value of created UTXOS	Median Value of created UTXOS	The median amount of coins in newly created UTXOs	GlassNode	OnChain
Minning Difficulty	Minning Difficulty	The current estimated number of hashes required to mine a block	GlassNode	OnChain
MVRV-Z Score	MVRV Z Score	It is defined as the ratio between the difference of market cap and realized cap, and the standard deviation of all historical market cap data	GlassNode	OnChain
Realized Cap	Realized Cap	Realized Cap values different part of the supplies at different prices (instead of using the current daily close)	GlassNode	OnChain
balance-100	Addresses with Balance	The number of unique addresses holding at least 100 coins.	GlassNode	OnChain
balance-1	Addresses with Balance	The number of unique addresses holding at least 1 coin.	GlassNode	OnChain
balance-10	Addresses with Balance	The number of unique addresses holding at least 10 coins.	GlassNode	OnChain
balance-10k	Addresses with Balance	The number of unique addresses holding at least 10k coins.	GlassNode	OnChain
percent-of-supply	Percent of supply	The percent of circulating supply that has not moved in at least 3 years	GlassNode	OnChain
trans_fees	Transactions Fees	Price to pay for every transaction	GlassNode	OnChain
Volume	Volume	Volume traded	GlassNode	OnChain
unrealized-profit	Unrealized profit	Total profit (in USD) of a coin in existence whose price at realization time was lower than the current price, normalized by the market cap	GlassNode	OnChain
ratio_mvrv	Market Value to Realized Value	The ratio between market cap and realised cap. It gives an indication of when the traded price is below a "fair value"	GlassNode	OnChain
block_reward	Miner Revenue	Miner's money amount recieved for every minted block	GlassNode	OnChain
block_size	Block Size	Size of the block	GlassNode	OnChain
ratio_nvt	Network Value to Transactions	The Network Value to Transactions (NVT) Ratio is computed by dividing the market cap by the transferred on-chain volume measured in USD	GlassNode	Onchain

3 Exploratory Data Analysis (EDA)

The final dataset contains 2026 rows x 66 columns with a timeframe that goes from 2016-02-29 to 2023-05-08. We can observe that there are only two categorical variables and the rest are numerical. Fortunately, these two categorical variables have only two labels (0,1), so there is not much work to be done with the encoders.

A first analysis is performed to study duplicate, outlier and null values. It is obtained that the dataset does not contain any duplicate values, but it does present null values.

When dealing with missing values in the dataset, certain columns were removed due to specific reasons. This decision was made to maintain a balanced representation of the data and avoid any bias introduced by the skewed distribution of values in those columns.

Overall, these steps were taken to clean the dataset, removing columns and rows with missing or irrelevant data. This process helps to ensure the quality and integrity of the data used for further analysis and modeling.

4 Feature Engineering

To define the initial data model, we created new variables from those in the original dataset.

We decided that the **target variable will be the price volatility** (close). Volatility of a price refers to how much it tends to change over time. High volatility means the price can experience big swings, while low volatility means the price changes more gradually. Volatility is important because it affects investment risks and trading opportunities. Higher volatility brings more risk and unpredictability, while lower volatility indicates more stability.

5 Multivariate Analysis

Once all variables are collected and the new ones calculated, we proceed to study the correlation between them and specifically with the target variable.

Performing a multivariate analysis is crucial in machine learning as it helps us understand the relationships and dependencies among multiple features. It allows us to capture interactions and uncover valuable insights. When handling correlation between features, strategies such as feature selection, dimensionality reduction, and regularization are employed to improve model performance and interpretability. It is important to assess the magnitude and direction of correlation and make informed decisions based on domain knowledge. By considering the interdependencies between features, we can build more accurate and robust models.

In this particular case, we did not apply any dimensionality reduction techniques to address the correlation between features. The dataset was small enough, and the computational resources available were sufficient to handle the original feature space without significant concerns.

It can be represented in a heat map that looks as follows:

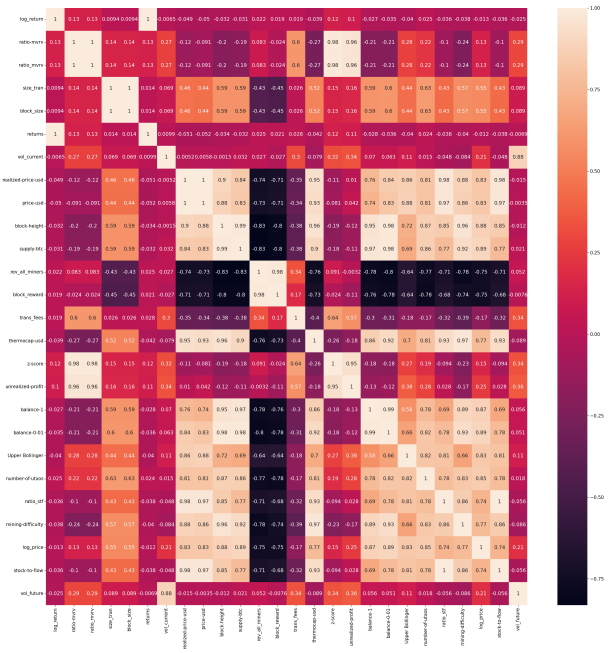

Figure 1: Correlation with all the original features

As can be seen, there is a high correlation between features. In particular, there are columns that correlate with almost all the variables (the blackest and whitest columns), therefore, the next step is to get rid of highly correlated columns in order to decrease the size of the dataset.

Finally the heat map correlation looks as follows:

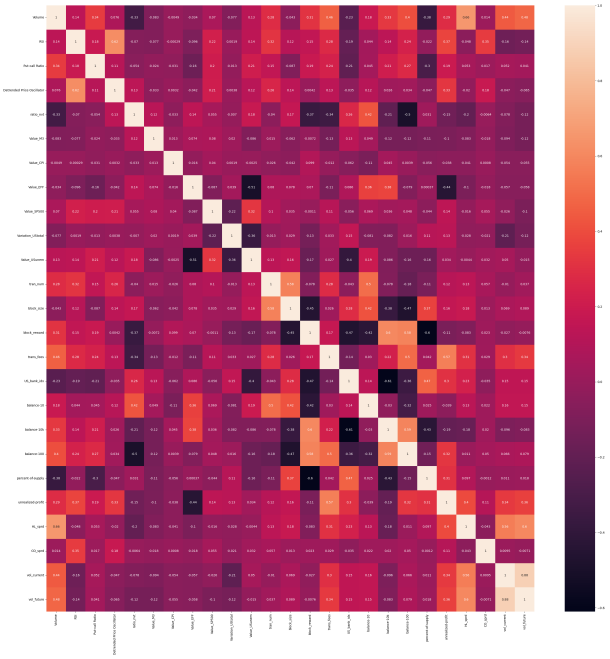

Figure 2: Correlation with rest of features

It shows now that there is no more high correlations between features.

6 Train-Validation-Test Splits

In machine learning, data is typically divided into three sets: training, validation, and test. The training set is used to teach the model and adjust its parameters. The validation set is used to fine-tune the model's settings and assess its performance on unseen data. Finally, the test set serves as a final evaluation, providing an unbiased measure of the model's capabilities on completely new examples. This separation ensures accurate assessment, prevents overfitting, and enables the development of reliable models for real-world applications.

In this case, since cryptocurrencies are not traded on a regulated exchange, the Bitcoin market is open 24/7, 1 year covers a whole 365 trading days instead of 252 days a year like with other stocks and commodities, so we decided to reserve 365 days to test the model and **30 days to test with new data**. In conclusion, final splits are 2126 days to training, 365 days to validation and 30 days to test.

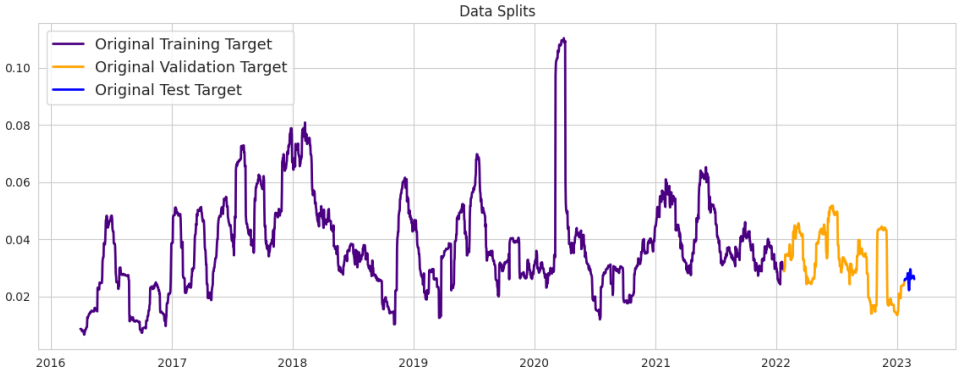

Figure 3: Spliting Data

7 Features Importance

The final dataset is defined with all the variables mentioned above and a selection is made. The first filtering of variables takes into account the high correlation with the rest of the data and the redundancy in the meaning of these variables.

Prior to modeling, the importance of the variables is studied with a linear regression and a simple algorithm based on decision trees, applying a "recursive feature elimination" (stepwise) technique.

Feature selection is a critical step that involves choosing the most important features from a dataset. It has several benefits, including improving predictive accuracy by focusing on relevant information, reducing overfitting by eliminating noise, enhancing interpretability by considering only meaningful features, and improving computational efficiency by reducing the dimensionality of the problem. In summary, feature selection plays a vital role in ensuring accurate and efficient machine learning models.

Recursive Feature Elimination (RFE) is a technique used to select the most important features from a dataset. It works by training a model on all features and iteratively eliminating the least important ones. This process helps improve model performance, reduce overfitting, and prioritize the most influential features for accurate predictions. RFE simplifies the model, enhances interpretability, and improves predictive accuracy by focusing on the most informative aspects of the data.

Surely there will be some gradient boosting model testing, so another recursive feature elimination (RFE) has been made with a Decision tree regressor.

These techniques are not used to filter variables directly, but to see which variables are going to be more relevant in future models. From these results, different final sets of variables are defined to launch the predictions.

8 Cross Validation Models

The solution with regression algorithms is treated as a typical problem with continuous numerical target variable, but avoiding the draw of observations to maintain temporal order in preprocessing and training, using the cross validation function customized for this problem. A first grid with different regression models is launched to initially see which ones fit the data best and we select to tune individually.

The different models selected are:

1. Linear Regression
2. Decision Tree Regressor
3. KNeighbors Regressor
4. Random Forest Regressor
5. Gradien Boosting Regressor
6. XGBoost
7. CatBoost Regressor

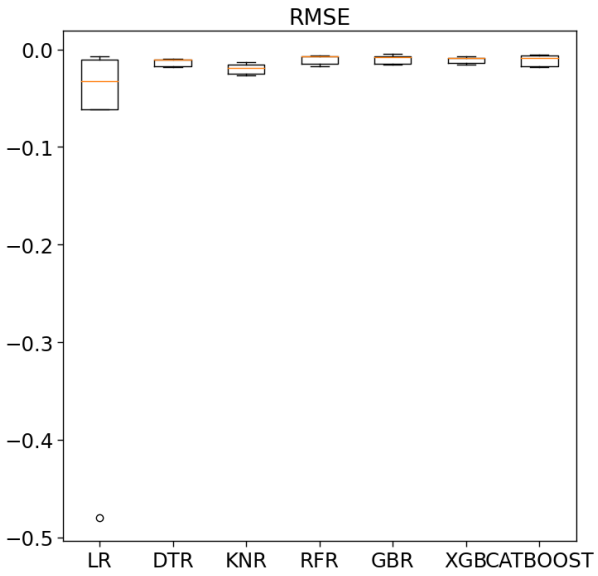

Figure 4: Cross Validation between Models

Observing Figure 6 we find that almost all the models worked well with the selected features. We finally determined that the **XGBoost** and **CATBoost** ensemble models are the candidates to obtain the best predictions. These are widely tunable and a better study of the optimal parameters will improve the predictions.

9 Modeling

Usually with financial time series, if we just shift through the historic data trying different methods, parameters and timescales, it's almost certain to find to some strategy with in-sample profitability at some point. However the whole purpose of "forecasting" is to predict the future based on currently available information, and a model that performs best on training data might not be the best when it comes to out-of-sample generalization (or overfitting). Avoiding/Minimizing overfitting is even more important in the constantly evolving financial markets where the stake is high.

The two main metrics we will be utilizing to evaluate the performance of our algorithm are the RMSPE (Root Mean Square Percentage Error) and RMSE (Root Mean Square Error). These metrics provide insights into the accuracy of our model's predictions compared to the actual values present in the dataset.

The RMSPE metric is particularly valuable when we want to assess the accuracy of our predictions while considering the magnitude of the actual values. It measures the percentage difference

between the predicted and actual values. This is especially useful when dealing with datasets that contain values with varying scales.

The formula for RMSPE can be expressed as follows:

$$\text{RMSPE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{(y_{\text{pred}_i} - y_{\text{true}_i})}{y_{\text{true}_i}}\right)^2}$$

Here, 'n' represents the total number of data points in the dataset, 'y_pred' corresponds to the predicted value, and 'y_true' denotes the actual (true) value. The formula calculates the square of the percentage difference between each predicted and true value, sums them up, takes the average, and finally computes the square root of the result. Multiplying by 100 converts the result into a percentage.

On the other hand, the RMSE metric provides a measure of the average magnitude of the differences between predicted and actual values in the dataset. It is commonly used in regression problems to evaluate the accuracy of a model's predictions.

The formula for RMSE can be expressed as follows:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_{\text{pred}_i} - y_{\text{true}_i})^2}$$

Similarly, 'n' represents the total number of data points, 'y_pred' represents the predicted value, and 'y_true' represents the actual (true) value. The formula calculates the squared difference between each predicted and true value, sums them up, takes the average, and finally computes the square root of the result.

Both the RMSPE and RMSE metrics provide valuable insights into the error between the predicted and actual values. While RMSPE is a relative error metric expressed as a percentage, RMSE is an absolute error metric expressed in the same units as the target variable. Including these metrics in our algorithm's documentation will help users understand and interpret the accuracy of our model's predictions in relation to the actual values present in the dataset.

9.1 XGBoost

XGBoost is a powerful machine learning algorithm that has demonstrated superior performance compared to other algorithms in various tasks, such as classification, regression, and ranking. The model operates by training multiple decision trees, where each tree is trained on a different subset of the data. The predictions from these individual trees are then combined to produce the final prediction. XGBoost is an advancement over the traditional GBM algorithm, primarily due to its use of a more regularized model, which helps prevent overfitting.

Furthermore, XGBoost offers several parameters that can be adjusted to enhance the algorithm's performance. However, in the context of this work, we did not fine-tune these parameters as our focus was on a different model that yielded better results.

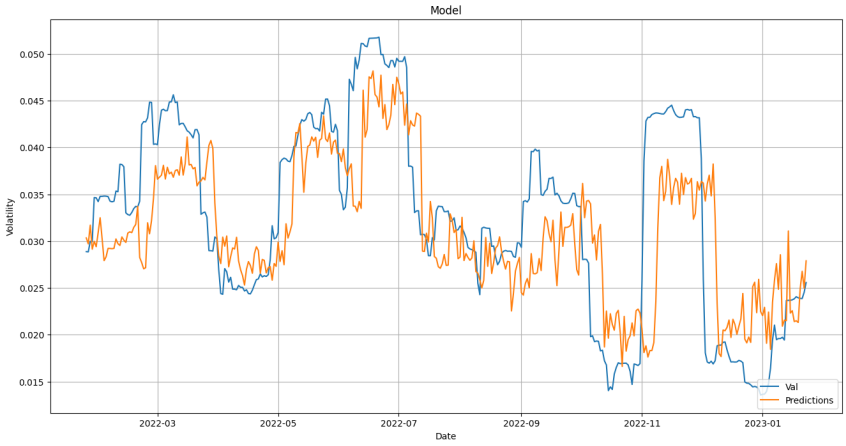

Figure 5: XGBoost Predictions

9.2 CATBoost

CatBoost, which stands for Categorical Boosting, is a gradient boosting algorithm specifically designed to handle categorical features effectively. It automatically handles categorical variables without requiring explicit preprocessing steps, such as one-hot encoding. CatBoost employs a unique method called ordered boosting, which helps handle categorical variables by sorting the categories based on the target variable's response rate. This technique allows CatBoost to capture valuable information from categorical features and improve model performance.

Another key distinction is in the handling of missing values. XGBoost requires explicit handling of missing values, where the user needs to provide a strategy for dealing with missing data. In contrast, CatBoost has built-in mechanisms to handle missing values, allowing it to handle missing data without the need for additional preprocessing steps.

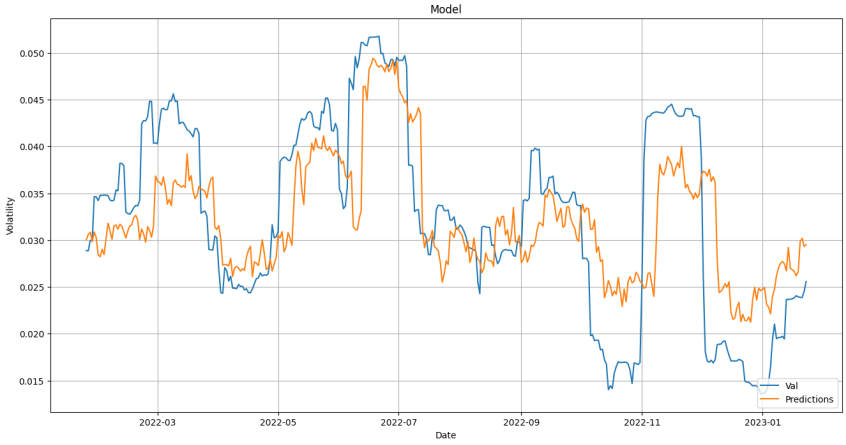

Figure 6: CATBoost Predictions

Model	Validation RMSPE	Validation RMSE
Xgboost	0.238343	0.006288
Catboost	0.262853	0.007015

Table 1: XG/CATBoost Results

9.3 LSTM

An LSTM model is a type of recurrent neural network (RNN) architecture that is specifically designed to work well with sequences of data, such as time series or text. It is particularly effective at capturing long-term dependencies and patterns in sequential data.

Imagine you have a sentence: "The weather is sunny, and the birds are chirping." When reading this sentence, our human brain understands the context and remembers the relevant information as we encounter new words. LSTM models aim to mimic this ability to retain and recall information over long periods.

Here's a simplified explanation of how an LSTM works:

- Memory Cells: LSTMs have memory cells that can store information. These memory cells act as the "memory" of the model and can remember important features from past inputs.
- Information Flow: When processing a sequence of inputs (e.g., words in a sentence), the LSTM model decides which information to keep and which to forget or update in its memory cells. It does this through a series of mathematical operations called gates.
- Input Gate: The input gate determines which parts of the new input should be stored in the memory cells. It selectively updates the memory cells with relevant information from the current input.
- Forget Gate: The forget gate determines which parts of the existing memory cells should be forgotten or discarded. It selectively removes information that is no longer relevant or useful.
- Output Gate: The output gate controls what information from the memory cells should be passed to the next step in the sequence or used as the model's output. It selectively retrieves relevant information from the memory cells.

By using these gates and memory cells, LSTMs can capture long-term dependencies in the data. This ability is particularly useful in tasks such as natural language processing, where understanding the context and relationships between words over long distances is crucial.

In summary, an LSTM model is a type of neural network architecture that excels at processing and understanding sequential data. It can remember important information over long periods, making it useful for tasks such as language translation, sentiment analysis, speech recognition, and more. Its ability to capture long-term dependencies sets it apart from traditional feedforward neural networks and makes it an excellent choice for various applications involving sequential data.

9.3.1 BILSTM 2 Layers

A BILSTM (Bidirectional LSTM) model with 2 layers is designed to help us understand the meaning of a sentence by considering not only the words that come before a specific word but also the words that come after it. Let's consider the previous sentence, the BILSTM model processes the words in the sentence one at a time, taking into account the sequential order in which the words appear. It consists of two main layers that work together to capture the relationships and dependencies between words.

In the first layer, the model processes the words in a forward direction, just as we typically read a sentence. This layer focuses on understanding the connections between the current word and the words that come before it. For instance, it recognizes that 'sunny' is related to 'weather' and 'is', and it understands the context created by these preceding words.

In the second layer, the model processes the words in a backward direction, reading them from right to left. This layer considers the words that come after the current word and captures the dependencies and relationships in the reverse order. In our example sentence, the backward layer recognizes that 'birds' and 'chirping' are connected to 'are' and 'and', respectively.

By processing the words in both forward and backward directions, the BILSTM model captures a more comprehensive understanding of each word's context within the sentence. It combines the information from both layers to create a richer representation of the sentence's meaning.

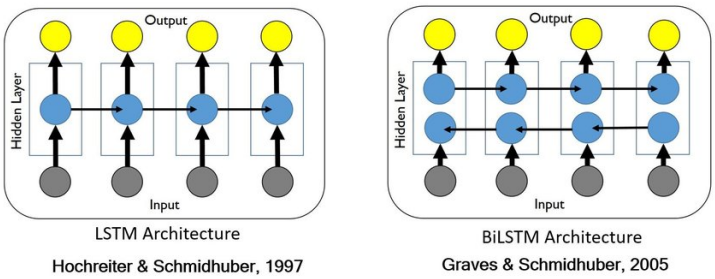

Figure 7: Diagram of LSTM - BILSTM [1]

In summary, a BILSTM model with 2 layers reads a sequence of words in both the forward and backward directions. It considers the relationships between words that come before and after each word, enhancing its ability to understand the meaning of a sentence and its context.

9.3.2 BILSTM 3 Layers

Adding another layer to a BILSTM model enhances its ability to capture intricate patterns and dependencies in a sentence. With 3 layers, the model can extract higher-level features and context, resulting in a more nuanced understanding of the sentence. The extra layer incorporates broader connections and global information, enabling the model to uncover deeper contextual nuances. However, adding more layers increases computational complexity and the risk of overfitting, so the number of layers should be chosen carefully based on the task and available resources.

Best model found:

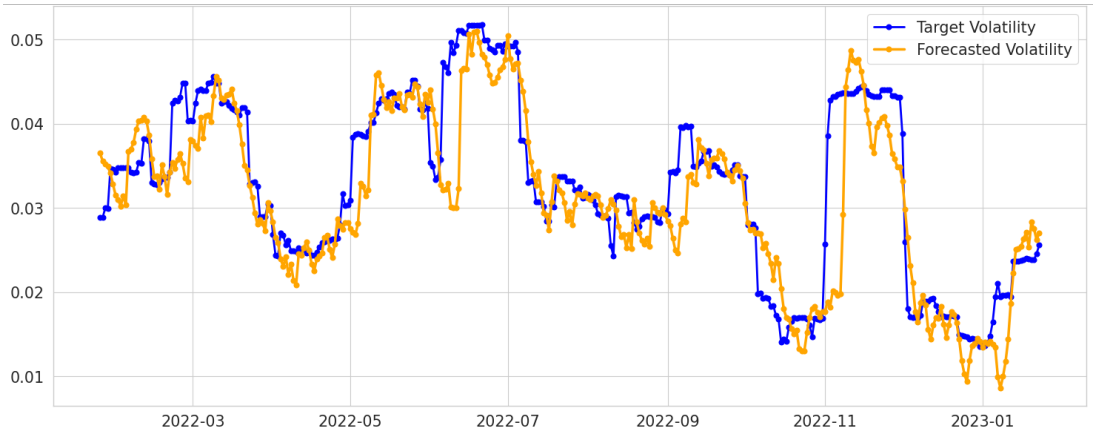

Figure 8: Best BILSTM found for selected features

9.3.3 BILSTM 4 Layers

When attempting to add an additional layer to the 3-layer BILSTM model, the team observed that the performance worsened compared to both the 2 and 3-layer configurations. This outcome may be attributed to several factors. Firstly, the increased complexity introduced by an extra layer can make it more challenging to train the model effectively. Deeper architectures with more layers typically require larger datasets, longer training times, and advanced optimization techniques to converge to optimal solutions. Insufficient data or inadequate training processes can hinder the model's ability to generalize well, resulting in poorer performance compared to shallower alternatives. Additionally, the risk of overfitting increases as the number of layers grows. The model may start to memorize the training data rather than learning meaningful patterns, leading to decreased generalization on unseen data. To mitigate overfitting, proper application and tuning of regularization techniques, such as dropout or L2 regularization, are crucial. The curse of dimensionality should also be considered. As the model's parameter space expands with each additional layer, the risk of encountering spurious correlations and noise grows. This can cause the model to become overly sensitive to small variations in the training data, ultimately leading to reduced performance.

It is essential to carefully evaluate different model configurations, including monitoring training dynamics, assessing validation performance, and considering the impact on computational resources. Ultimately, finding the optimal architecture requires a thorough analysis of the dataset, problem complexity, and available resources.

Performance of BILSTM for different layers and selected features is shown in table 3.

Model	Validation RMSPE	Validation RMSE
lstm 10	0.150788	0.005144
lstm 5	0.153847	0.005011
lstm 3	0.163785	0.005943
lstm 1	0.170119	0.005076
lstm 16	0.178508	0.005151
lstm 9	0.180441	0.005454
lstm 4	0.180544	0.005459
lstm 14	0.180991	0.005510
lstm 15	0.181374	0.006113
lstm 11	0.191571	0.005256
lstm 8	0.207020	0.005755
lstm 2	0.212684	0.005568
lstm 7	0.215779	0.005722
lstm 12	0.244531	0.006044
lstm 6	0.254794	0.006083
lstm 13	0.292771	0.006668

Table 2: BILSTM Results

10 TEST Results

In the final phase of our analysis, we tested our model using the test data, which was held separate from the training and validation data. We evaluated the performance of our model using different lag values, specifically n_lag=1, n_lag=7, n_lag=15, and n_lag=30, to assess their impact on the results.

Interestingly, we observed that our model performed exceptionally well with both n_lag=1 and n_lag=7, yielding accurate predictions. However, the performance deteriorated when we increased the lag value to n_lag=15 and n_lag=30.

Actual Date	Horizon	Predictions	Date	RMSPE
2023-06-06	Days 1	0.0252	2023-06-07	0.089
2023-06-06	Days 7	[0.0201, 0.0239]	2023-06-13	0.159
2023-06-06	Days 15	[0.0238, 0.0289]	2023-06-21	0.266
2023-06-06	Days 30	[0.0259, 0.038]	2023-07-06	0.402

Table 3: Final Results

The reason behind this disparity in results lies in the nature of the data and the characteristics of the time series being analyzed. When using a lag of 1 or 7, the model can capture the immediate or short-term dependencies and patterns in the data. This is particularly effective when the underlying time series exhibits relatively fast or moderate changes.

However, as the lag value increases to 15 or 30, the model's ability to capture short-term dependencies diminishes, and it starts to focus more on long-term patterns. In certain cases, the underlying time series may not exhibit strong long-term dependencies or may be subject to significant fluctuations, making it challenging for the model to accurately capture and predict the future values.

Moreover, a higher lag value increases the complexity of the model, which can lead to overfitting or capturing noise in the data. Overfitting occurs when the model becomes too specialized to the training data and fails to generalize well to new, unseen data. This can result in poorer performance when using larger lag values.

Therefore, the superior performance observed with n_lag=1 and n_lag=7 suggests that the underlying time series data may primarily rely on short-term dependencies and exhibit relatively fast or moderate changes. On the other hand, the poorer performance with n_lag=15 and n_lag=30 indicates that long-term dependencies might be less pronounced or subject to higher fluctuations, making it more challenging for the model to accurately predict future values.

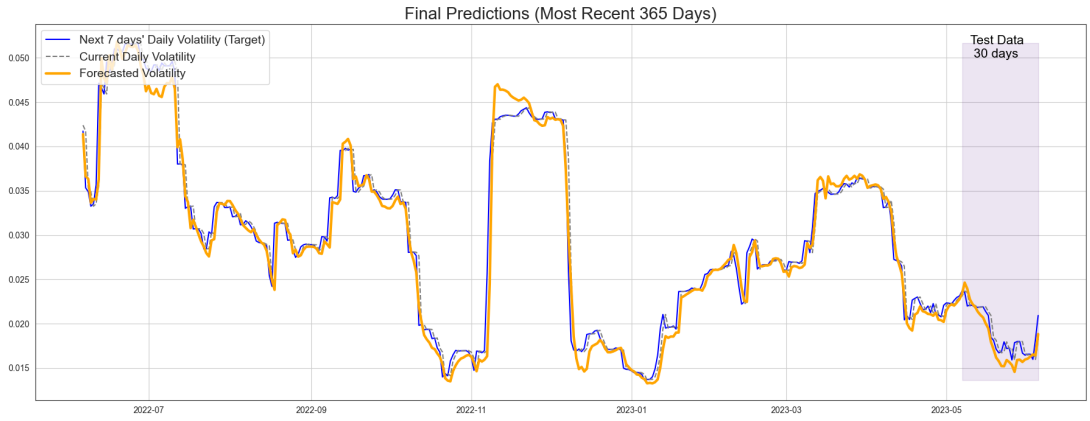

Figure 9: Predicted volatility with one-day horizon

11 Conclusion

After evaluating various models for time series forecasting, including traditional ARIMA-GARCH models, boosting algorithms like XGBoost and CatBoost, as well as deep learning models like LSTM, we have determined that a BiLSTM model outperforms the others.

There are a few reasons why the BiLSTM model with these specific features is yielding the best results. The BiLSTM architecture allows the model to capture complex temporal dependencies and patterns in the time series data. By considering both past and future information, the model can effectively learn from the historical sequence of data points and make accurate predictions.

Furthermore, the combination of the architecture and the selected features allows the model to capture both short-term and long-term dependencies in the data. The model can learn from the intricate relationships between the features and the target variable, enabling it to make accurate volatility predictions.

It's important to note that the performance of the BiLSTM model is not solely due to its architecture and feature selection but also influenced by the characteristics of the dataset. The complexity and non-linear nature of BTC volatility require a more advanced model like BiLSTM to capture the underlying patterns effectively.

In conclusion, the BiLSTM model with the selected features demonstrates superior performance in forecasting BTC volatility compared to other models tested. Its ability to capture temporal dependencies, leverage informative features, and handle the unique characteristics of the dataset contribute to its success in providing accurate predictions.

12 Use Case: Dynamic Collateral Ratio

In the realm of Decentralized Finance (DeFi), the lending and borrowing landscape is primarily underpinned by collateral management practices. To secure loans, borrowers are required to over-collateralize their positions. However, most DeFi lending protocols use static collateral ratios, which don't account for changing market conditions. This can lead to over-collateralization in stable periods or increased risk during volatile periods, both of which are sub-optimal outcomes.

We are addressing this issue by introducing a groundbreaking solution: the Dynamic Collateral Ratio (DCR). Harnessing the power of data and advanced algorithms, the DCR aims to improve efficiency and reduce risk in DeFi lending protocols.

Algorithmic Approach

The heart of the DCR is a robust algorithm designed to measure the volatility of Bitcoin and other tokens. It uses a range of inputs, including recent price data, trading volume, market sentiment, and other relevant factors. The algorithm is dynamic, continually updating to provide measurements of volatility, thus enabling a responsive collateral management system.

DCR: The Core Concept

The DCR takes the volatility data and translates it into a recommended collateral ratio. During periods of increased volatility, the DCR will be higher to account for the greater risk, and during periods of low volatility, the DCR will be lower, which prevents over-collateralization and enables more efficient capital allocation.

Real-Time Implementation

With our DCR tool, DeFi lending protocols can implement a dynamic collateral management system. Using data feeds, the DCR adjusts in response to changes in market volatility. For instance, during a period of high volatility, the DCR would increase, thus indicating a need for a higher collateral ratio. Conversely, in times of low volatility, the DCR would decrease, highlighting the possibility for a lower collateral requirement. This allows the DeFi protocol to adjust their collateral requirements proactively, thus managing their risk more effectively.

Risk Management Enhancement

In times of increasing volatility, protocols can take proactive measures such as issuing margin calls, reducing the risk of mass liquidations. Furthermore, DeFi protocols should develop risk management policies around the use of the DCR, including procedures for communicating changes in the collateral ratio to borrowers and managing situations where borrowers can't meet the new collateral requirements.

Integration and Support

Our DCR tool is designed with seamless integration in mind. We provide an Application Programming Interface (API) that allows DeFi protocols to easily pull the data on the recommended collateral ratio. Additionally, we offer comprehensive support during the integration process, including technical assistance, training, and troubleshooting, ensuring a smooth and efficient integration with existing systems.

In Conclusion

We are driving innovation in the DeFi space with our Dynamic Collateral Ratio tool. By providing a real-time, responsive solution to collateral management, we aim to improve efficiency, reduce risk, and ultimately contribute to the stability and growth of the DeFi ecosystem. We believe that with our DCR, DeFi lending protocols can evolve to become more resilient and adaptive, fostering trust and confidence among their users.

13 Next Steps

13.1 Time Series GANs:

- Synthetic Data Generation: GANs will be used to generate synthetic price data for cryptocurrencies, including Ethereum and other tokens, under various market conditions. This data will be used to further train and test our volatility algorithm, potentially improving its accuracy and robustness. This will be particularly useful for rare events (Black Swans), such as extreme price movements, for which real data may be scarce.
- Scenario Simulation: in this case GANs will be used to simulate different market scenarios based on historical data. By generating synthetic data that reflects possible future states of the market, GANs will help anticipate how our algorithm might perform under different conditions. This will be useful for stress testing our model and identifying potential weaknesses or areas for improvement.
- Transfer Learning: with this technique GANs will potentially be used to transfer the key features, to predict the behavior of cryptocurrencies, like Ethereum and other tokens, which have similar yet distinct characteristics.
- Anomaly Detection: lastly, GANs will be used to improve the anomaly detection capabilities of the algorithm. By training a GAN on normal market conditions, it will potentially generate what it considers to be "normal" data. Then, by comparing real-time data to this "normal" data, the algorithm will be able to better identify anomalous market conditions that could indicate increased volatility.

13.2 Credit Risk

The main purpose of the algorithm is to evaluate the credit risk associated with each entity or transaction within a DeFi system, providing a risk score or rating that can be utilized to make informed lending decisions.

The algorithm will take into account a wide range of parameters and data sources to generate a comprehensive credit risk score. These parameters include:

- Collateral quality: evaluates the type and quality of the collateral, considering factors such as volatility, liquidity, and market depth.
- Entity transaction history: Considers the entity's past transactions and activities within the DeFi ecosystem.
- External credit data: the algorithm will use external credit scoring data to enrich the information.
- Economic variables: takes into account broader market conditions and economic indicators, which can impact the risk of default.

Our Credit Risk Algorithm will be a sophisticated tool designed to navigate the unique challenges of risk assessment in the DeFi space. By considering a wide range of factors from collateral quality to economic indicators, we will provide a comprehensive and dynamic risk score that allows lenders to make informed decisions, promoting safer and more efficient lending practices in the DeFi ecosystem.

13.3 Liquidity Risk

Our DeFi Liquidity Risk Algorithm will be designed to address the unique challenges of evaluating liquidity risk within the decentralized finance (DeFi) ecosystem. The algorithm's primary objective is to provide a comprehensive assessment of the liquidity risk associated with individual assets or liquidity pools, generating a risk score that can be used to make informed decisions.

It will consider a variety of parameters and data sources to calculate a comprehensive liquidity risk score. These parameters include:

- Trading volume: Measures the amount of the asset that is being bought and sold. Higher trading volumes typically indicate lower liquidity risk.
- Order book depth: Assesses the market's ability to absorb large orders without significantly impacting the price.
- Volatility: Considers the price volatility of the asset. Higher volatility can increase liquidity risk as it may deter traders.
- Slippage: Evaluates the potential price impact of executing a trade. Liquidity pool depth: For assets in liquidity pools, the algorithm considers the total value locked in the pool.
- Market sentiment: Uses AI and Natural Language Processing (NLP) to analyze market sentiment from various online sources. Negative sentiment can sometimes lead to sudden withdrawals, increasing liquidity risk.

Based on this, we will create new derivatives such as **liquidity option.**

Bibliography

[1] LSTM and BiLSTM Architectures. Extracted from: : https://www.researchgate.net/figure/LSTM-and-BiLSTM-Architectures_fig2_324769532