Bloomberg

FINANCE AND RISK ENGINEERING

FRE-GY 7043 FINANCIAL ENGINEERING
CAPSTONE: PROJECT

FALL 2022

WRITTEN PAPER

Forecasting and Trading Cryptocurrencies with Deep Neural Network

Yiren Yao (yy3467)
Guojun Chen (gc2817)

Supervised By
Yang Wu

1 Introduction

1.1 Summary

In this paper we used Machine Learning and Deep Learning techniques to forecasting the future price movement up(1) and down(0) probability of cryptocurrency, and designed trading strategies based on the probability. We set Bagging and Boosting ML algorithms as our baseline model, including Random Forest(RF), Adaboost and XGBoost. And we also used typical DL model - Long Short Term Memory (LSTM) and TabNet - to forecast the probability of future movement. We found the forecast accuracy for the DL models are generally better than the ML models. We designed a Volatility Adapting trading strategies and backtested the model, found that volatility controlled strategy almost double the sharpe ratio and show greater risk controlling capacity compared with simply trading strategy.

1.2 Literature Review

The financial industry has always been interested in forecasting and trading cryptocurrencies. Numerous studies have been published that were based on Machine Learning (ML) models and Deep Learning(DL) techniques.

Omer Berat Sezer et al.[1] categorized the studies according to intended forecasting implementation areas and grouped them based on their DL model choices. Their paper was a coherent com comprehensive guidebook for DL model in time-series areas. For cryptocurrencies, Deep Neural Network (DNN), Long short-term memory (LSTM), Gated Recurrent Unit (GRU), as well as classical methods were used for cryptocurrency price forecasting.

There are generally two ways to forecast cryptocurrencies. One way is to forecast prices (returns) in the future. Laura Alessandretti et al.[2] used Gradient Boosting Decision Trees and LSTM with derived trading strategies, exploiting the prices inefficiency of the cryptocurrency market and generating abnormal profits compared with simple moving average strategy. Derbentsev et al. [3] applied two of the most powerful ensemble methods including Random Forests (RF) and Stochastic Gradient Boosting Machine (SGBM) and verified the applicability of the ML ensembles approach for the forecasting of cryptocurrency prices. Salim Lahmiri and Stelios Bekiros [4] found deep learning was found to be highly efficient in forecasting the inherent chaotic dynamics of cryptocurrency markets. Mohil Maheshkumar Patel et al. [5] proposed a LSTM and GRU-based hybrid cryptocurrency prediction scheme which accurately predicted the prices and revealed the scheme was applicable in various cryptocurrencies price predictions. Ioannis E. Livieris et al. [6] combined three of the most widely employed ensemble learning strategies: ensemble-averaging, bagging and stacking with advanced deep learning models for forecasting major cryptocurrency hourly prices, indicates that ensemble learning and deep learning can be efficiently beneficial to each other, for developing strong, stable, and reliable forecasting models. Sumit Biswas et al.[7] considered active LSTM networks for the benchmark data sets with high efficiency for forecasting of digital currency. Marco Ortu et al. [8] presented a comprehensive analysis of the predictability of price movements comparing four different deep learning algorithms (MLP, CNN, LSTM and ALSTM), finding that including both trading and social media indicators yields a significant improvement in the classification accuracy consistently across all algorithms. However, one of the biggest weaknesses for forecasting price is that price data has unit root and is non-stationary. Therefore it will be easily affected by regime change. Through our research, we also found there are certain points of price jumps if we set price as our target and forecasted price have much higher volatility than the actual data, so we turn to classification and forecast the probability of the up and down movement.

Another way to predict cryptocurrency with trading signal is to forecast states using classification model, labeling states as up(1) and down(0) using forecasted probability. Matthew Chen et al.[9] used Logistic Regression, Random Forest, Support Vector Machine(SVM) and Auto Regressive Integrated Moving Average(ARIMA) to forecast the movement of bitcoin price. A Erdinc Akyildirim et al.[10] analyzed most liquid twelve cryptocurrencies at the daily and minute level frequencies using the machine learning classification algorithms and reach about 55–65% predictive accuracy on average. David Mayo et al.[11] found the Artificial Neural Network(ANN) and RF classifiers consistently outperformed the other techniques and achieved a very high prediction accuracy, and there is no significant difference in predictability between the three prominent cryptocurrencies. Ahmed Ibrahim[12] found constructed an XGBoost-Composite ensemble model which achieved higher performance for predicting the sentiment than the state-of-the-art prediction models. Kwon, Do-Hyung et al. [13] found the LSTM model outperforms the gradient boosting model for the time series classification of the cryptocurrency price trend. With the LSTM model the performance improved about 7% compared to using the GB model. Nisarg P. Patel et al.[14] propose a deep-learning-based hybrid model (GRU and LSTM) to predict the price of Litecoin and Zcash with inter-dependency of the parent coin in real-time scenarios with high accuracy compared to existing models. Takuya Shintate et al.[15] provided a trend prediction classification framework and mitigated class imbalance problem, although strategy designed by the model didn't outperform the buy-and-hold strategy within the testing data period.

TabNet, initially written by Arik and Pfister for Google Cloud AI [16] recently shows great capacity in predicting tabular data. Yiyang Zheng[17] used TabNet to train the model on the Silver Futures Contract listed on Shanghai Futures Exchange and achieved an accuracy of 0.601 on predicting the directional change during the selected period. Although TabNet is a newly developed model and its application on market prediction is limited, it generated promising results in our reserach.

1.3 Framework

We designed our project framework as Figure 1. We introduce our Data in Section 2, Models and Results in Section 3 & 4, Backtesting in Section 5 and Conclusion in Section 6.

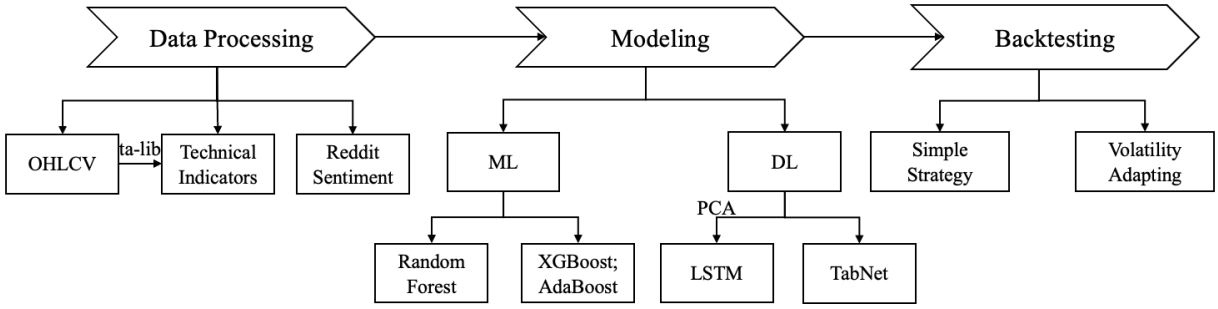


Figure 1: Framework

2 Data

Compared with traditional stock equities, cryptocurrencies do not have fundamental factors. Therefore we mainly focus on two aspects of data: Momentum and Sentiment. Momentum refers to the rate of acceleration of cryptocurrencies' price, i.e. the speed at which the price is changing. Momentum factors and technical indicators can be extracted from the price and volume data. On the other hand, investor's sentiment may also influence the price movement. We fetched discussion on social medias on Twitter and Reddit to extract and generate sentiment score based on the subjectivity and polarity of text data.

2.1 Price Data

We fetche the 1-minute level OHLCV (open, high, low, close, volume) data from Binance[1]. Binance is a cryptocurrency exchange which is the largest exchange in the world in terms of daily trading volume of cryptocurrencies. The price data are downloaded through Binance REST API[2]. We set the largest 50 coins by their market capitalization as our investable universe. Those stablecoins are excluded as they are not investment-worthy. The data range is from 2020/01/01 to 2022/11/30.

It should be noted that the price data we fetched are quotes of crypto against USDT, a stablecoin, rather than USD, a fiat coin. We made this choice due to data availability and resemblance to actual crypto trading. In actual exchange setting, cryptocurrencies are usually quoted against each other, including stablecoin. Those major exchanges, including Binance, only provide crypto pairs. To invest in crypto, the investor should first buy crypto by fiat currency, then trade it with other cryptos. Though there are some exchanges providing pairs between cryptos and fiat coins, like Coinbase and FTX, their coverage of our investment universe and data quality are unreliable.

2.2 Social Feeds

This project aims to use social feeds to extract and analyze sentiment levels in the crypto market. We choose Reddit as our major source of social feeds.

Reddit is a social news aggregation, content rating, and discussion website. Users submit content to the site such as links, text posts, images, and videos, which are then voted up or down by other members. There are several subreddits focusing on crypto-related topics, like r/CryptoCurrency and r/Bitcoin. Discussions on these subreddits are another source the extract crypto market sentiment.

We made a package [3] to fetch all posts and comments that we are interested in from Reddit. Besides text data, other stats, including the number of comments, up and down votes, and the number of likes, are also fetched. However, due to the limit of Reddit API[4], historical data cannot be fetched directly. To solve this issue, we switch to Pushshift[5], which was designed and created by the /r/datasets mod team to help provide enhanced functionality and search capabilities for searching Reddit comments and submissions. We obtain the archived post and comment from Pushshift and then update them through Reddit API. Figure 2 shows the way we download the Reddit text data.

[1]Binance Website: https://www.binance.com/en
[2]Binance API Document: https://binance-docs.github.io/
[3]Reddit Fetch: https://github.com/nyu-tandon-capstone/reddit-fetch
[4]Reddit API: https://www.reddit.com/dev/api/
[5]Pushshift API: https://github.com/pushshift/api

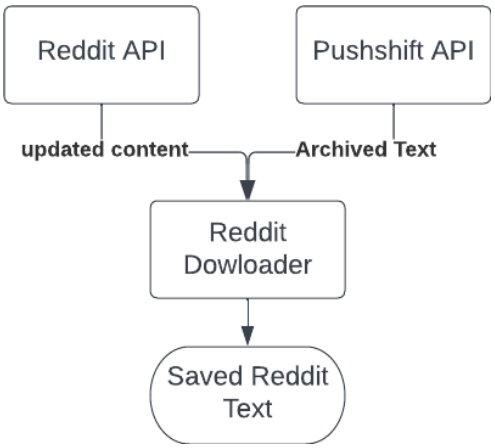


Figure 2: Flow of downloading Reddit data

At the current stage, we select r/CryptoCurrency, r/CryptoMarkets and r/Bitcoin and download all posts and comments from these subreddits. These three subreddits have a large number of users and lots of discussions every day. Figure 3 shows that these subreddit all have over 1 million users. Figure 4 shows the number of discussion every day on these subreddits. We can also notice that the number of discussion peaks when the crypto market is in good condition. Our final dataset contains 299,239 posts and 12,903,284 comments, posted between June 2017 and November 2022.

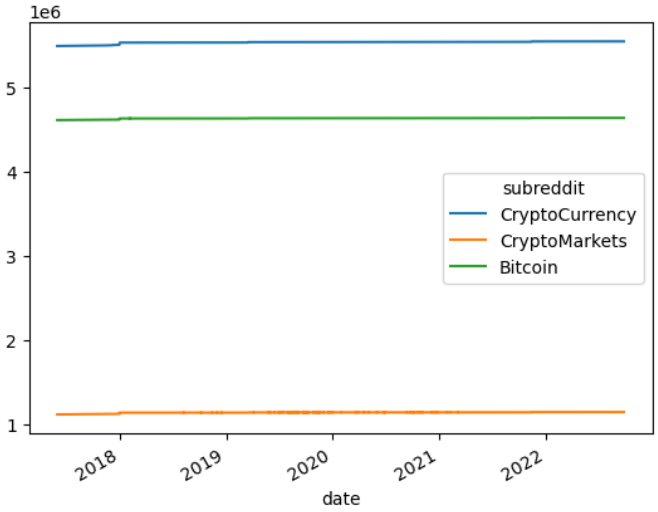


Figure 3: Number of subscriber

2.3 Data Preprocessing

After fetching all price and text data, we apply the pre-processing procedure. For the price data, we first check the continuity of timestamps and insert the missing ones. Then we deal with the NA values. The close price is propagated with the last available close price. The open, high, and low prices are filled with the close price. The volume is set to 0. We also transfer the 1-min level price data to lower frequencies. 5-min, 10-min, 30-min and 60-min price data are prepared. Technical indicators for different frequencies are then extracted by the ta−lib[6] package. Total 146 technical indicators, which can be seen in Appendix A, are applied into model.

[6]Technical Indicators (ta-lib) API Document: https://mrjbq7.github.io/ta-lib/

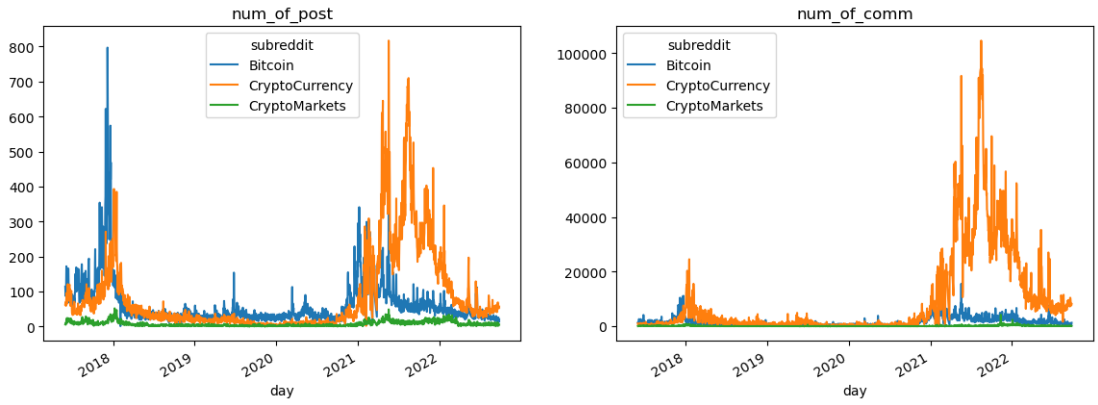


Figure 4: Daily Number of Post and Comment

For the text data, we follow the common textual data pre-processing procedures, First, all texts are lower cased. Sequences of repeated letters are reduced to a maximum length of three. Tickers, hashtags, mentions, and hyperlinks are extracted as tags and removed. Other special characters are removed. We also removed stopwords with less potential function in the sentence. All punctuation is removed. In addition, there are a lot of comments are sent by robots, which are marked by the phrase "I am a bot, and ...". Comments in this format are deleted.

Raw Text	Processed Text
June of next year. BTC will be 9k.	june of next year. cryptotag will be 9k.
... is elaborated in the paper at: https://drive.google.com/...	... is elaborated in the paper at linktag.
#InvestAtYourOwnRisk #Bitcoin	hashtag hashtag
...I am a bot, and this action ...	[delete]

Table 1: Text Preprocessing

After preprocessing these raw texts, we combine post and comment text data and perform a transform by a pre-trained language model (nltk.SentimentIntensityAnalyzer) to get polarity scores for each text entry. We then aggregate them into different frequencies, aligning with the price data. Take 1-min level as an example, all posts and comments submitted in one specific minute are aggregated and the mean, sum, and std of their sentiment scores are calculated.

3 Model

In this section, we will introduce basis of the models we used and selected. For Machine Learning techniques we mainly focused on ensemble learning method, and for Deep Learning techniques we mainly focused on

3.1 Machine Learning

There are two main types of ensemble learning, namely bagging and boosting. Bagging combines Bootstrapping and Aggregation to form one ensemble model. Specifically, independent base models are performed on different subsets of the training data in a parallel way. Boosting, however, trains several weak models sequentially by focusing on the mistakes of prior iterations. Specifically, two ensemble models are chosen to train target variable.

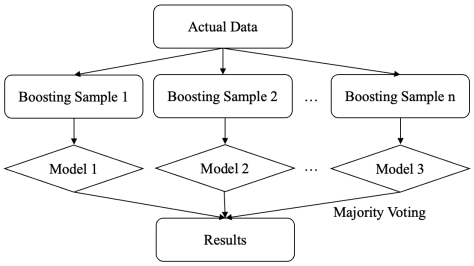


Figure 5: Bagging: Build Parallel

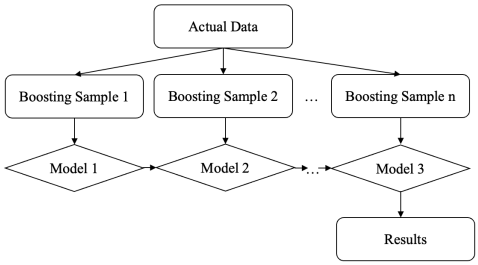


Figure 6: Boosting: Build Sequentially

3.1.1 Bagging: Random Forest

Random forests can be considered as a bagging of decision trees, which typically used for classification and regression problems. For classification tasks, the output of the random forest is the class selected by most trees. For regression tasks, the mean or average prediction of the individual trees is returned. Random decision forests correct for overfitting habits of decision trees on training set and generally outperform decision trees.

The only difference is that random forest models make split decisions based on different randomly selected features. By doing so, we achieve a decrease in correlation which leads to a lower variance. In other words, this model generates relatively robust results. The main problem is that random forest is a black box type of model and does not allow much participation in estimation aside from randomization.

3.1.2 Boosting: Adaboost

Adaboost (Adaptive Boosting) is the first successful boosting algorithm. More weight is put on previously larger residuals, while less on those already handled well. Intuitively, boosting can result in better performance than bagging. Whereas they tend to be harder to tune and be more likely to overfit.

Random Forest and Adaboost based on the creation of a Forest of trees, which can be used for both classification and regression tasks. Apart from the difference mentioned above, several other key difference between the two models are:

- Random Forest set certain number of full sized trees on different subsets of the training dataset while Adaboost uses decision stumps (decision tree with only one split), basically a forest of stumps, i.e. weak learners, which have high bias and low variance.

- Each decision tree in the Random Forest is made up using all the features in the dataset while decision stumps in Adaboost use one feature at a time.
- Each decision tree in Random Forest is made independent of each other, therefore the order in which trees are made is not important. While in Adaboost, order of stumps do matter. Each stump takes the previous mistakes into account, which essentially, focuses on modelling errors from previous stumps.
- Each tree in the Random Forest has equal weights in the final decision while in Adaboost different stumps have different weights in the final decision. The stump which makes less error in the prediction, has high amount of weights as compared to the stump which makes more errors.
- Random Forest aims to decrease variance not bias while Adaboost aims to decrease bias not variance.
- There are rare chances of Random Forest to overfit while there are good chances of Adaboost to overfit.

Generally, in modeling procedure, Adaboost provides more accurate predictions than Random Forest while it is also more sensitive to overfitting than Random Forest.

3.1.3 Boosting: XGBoost

XGBoost (eXtreme Gradient Boosting) is a scalable, distributed gradient-boosted decision tree (GBDT) model. It provides parallel tree boosting for regression, classification, and ranking problems.

Both of XGBoost and Adaboost methods are built based on the idea of converting weak learners to a strong learner by updating based on the residuals (XGBoost) or misclassifications (AdaBoost). XGBoost was developed to increase speed and performance, with introducing regularization parameters to reduce overfitting that successfully reduces variance. However, XGBoost is more difficult to understand, to visualize and to tune parameters compared to AdaBoost, where there is a multitude of hyperparameters that can be tuned to increase performance.

AdaBoost is relatively robust to overfitting in low noise datasets and has only a few hyperparameters that need to be tuned to improve model performance. However, for noisy data (min-level data), its performance is debated with some arguing and leads to poor performance due to the algorithm spending too much time on learning extreme cases and skewing results.

3.2 Deep Learning

DL is a type of Artificial Neural Network(ANN) that consists of multiple processing layers and enables high-level abstraction to model data. The key advantage of DL models is extracting the good features of input data automatically using a general-purpose learning procedure.

Deep Multi Layer Perceptron (DMLP) is one of the first developed ANNs. The difference from shallow nets is that DMLP contains more layers. Even though particular model architectures might have variations depending on different problem requirements, DMLP models consist of mainly three layers: input, hidden and output. The number of neurons in each layer and the number of layers are the hyperparameters of the network. In general, each neuron in the hidden layers has input vector(x), weight matrices (W) and bias vector (b). In addition, each neuron has a nonlinear activation function which produces a cumulative output of the preceding neurons. The output of a single neuron in the Neural Network (NN):

$$y = f(\sum_i Wx + b)$$

There are different types of nonlinear activation functions (f). Most commonly used nonlinear activation functions are: sigmoid (σ), hyperbolic tangent (tanh), Rectified Linear Unit (ReLU), Leaky-ReLU, sigmoid linear Unit (SiLU/swish) , and softmax.

Activation Function	Formula
σ	$\frac{1}{1+e^{-z}}$
tanh	$\frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$
ReLU	$\max(0, z)$
Leaky-ReLU	$1_{x<0}(\alpha x) + 1_{x>=0}(x)$
SiLU	$x\sigma(\beta x)$
softmax	$\frac{e^(z_i)}{\sum_j e^{z_j}}$

Table 2: Activation function

DMLP models have been appearing in various application areas. Through DMLP models, problems such as regression and classification can be solved by modeling the input data. However, if the number of the input features is increased, the parameter size in the network will increase accordingly due to the fully connected nature of the model and it will jeopardize the computation performance and create storage problems. To overcome this issue, different types of Deep Neural Network methods are proposed. With DMLP, much more efficient classification and regression processes are performed. A DMLP model, layers, neurons in layers, weights between neurons are shown in Figure 7.

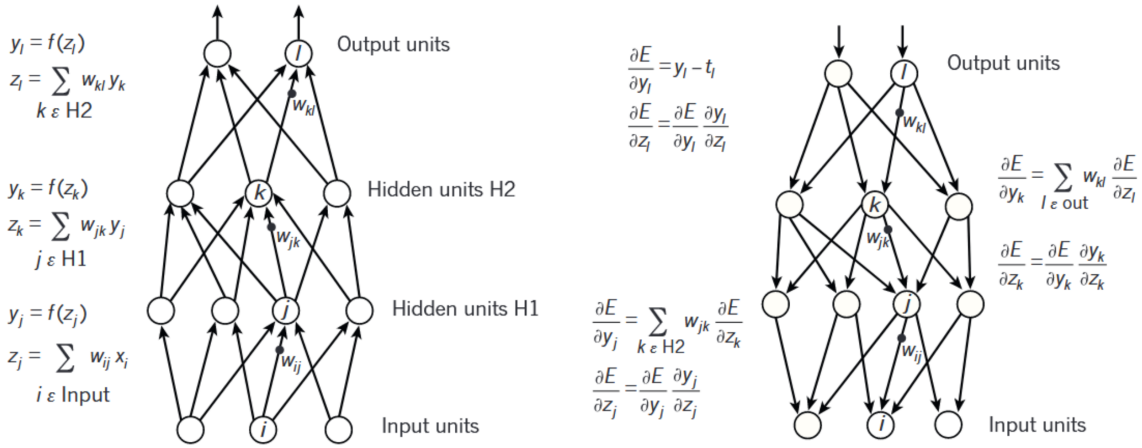


Figure 7: DMLP Forward Pass (Left) and Backpropagation (Right)

The important issue in the DMLP are the hyperparameters of the networks and method of tuning these hyperparameters. Hyperparameters are the variables of the network that affect the network architecture, and the performance of the networks. The number of hidden layers, the number of units in each layer, regularization techniques (dropout, L1, L2), activation functions (Sigmoid, ReLU, hyperbolic tangent, etc.), learning rate, decay rate, momentum values, number of epochs, batch size (minibatch size), and optimization algorithms (SGD, AdaGrad, RMSProp, etc.) are the hyperparameters of DMLP. Choosing better hyperparameter values/variables for the network result in better performance.

3.2.1 Long Short Term Memory (LSTM)

Recurrent Neural Network (RNN) is a type of DL network that is used for time series or sequential data. Deep RNNs are preferred due to their ability to include longer time periods. Unlike Fully Connected Neural Networks, RNNs use internal memory to process incoming inputs. RNN model architecture consists of different number of layers and different type of units in each layer. Each RNN unit takes the current and previous input data at the same time, and the output depends on the previous data in RNN model. The RNNs process input sequences one by one at any given time, during their operation. In the units on the hidden layer, they hold information about the history of the input in the state vector. When the output of the units in the hidden layer is divided into different discrete time steps, the RNNs are converted into a DMLP. In Figure 8, the information flow in the RNN's hidden layer is divided into discrete times. The status of the node s at different times of t is shown as s_t, the input value x at different times is x_t, and the output value o at different times is shown as o_t. Parameter values (U, W, V) are always used in the same step.

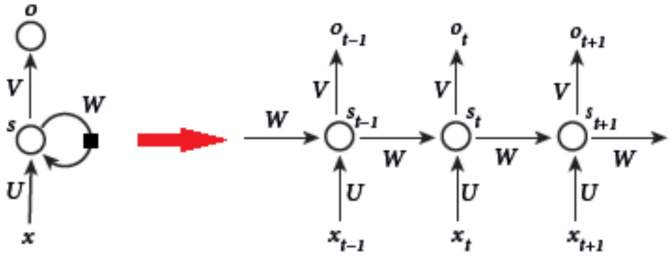


Figure 8: RNN cell through time

RNNs can be trained using the Back Propagation Through Time algorithm. With this, the error change at any t time is reflected in the input and weights of the previous t times. The difficulty of training RNN is due to the fact that the RNN structure has a backward dependence over time. Therefore, RNNs become very complex in terms of the learning period. Most of hyperparameters of RNN are the same as DMLP. The most important difference is we need to decide the time sequence length.

Gated Recurrent Unit (GRU) are a gating mechanism in recurrent neural networks. The GRU is like a long short-term memory (LSTM) with a forget gate, but has fewer parameters than LSTM, as it lacks an output gate. The performance of GRU on certain tasks was found to be similar to that of LSTM. GRUs have been shown to exhibit better performance on certain smaller and less frequent datasets. To solve the vanishing gradient problem of a standard RNN, GRU uses update gate(z_t) and reset gate(r_t). Basically, these are two vectors which decide what information should be passed to the output. The special thing about them is that they can be trained to keep information from long ago, without washing it through time or remove information which is irrelevant to the prediction. h_t and $\hat{h}_t$: output vector and candidate activation vector, σ_g: sigmoid function, σ_h: hyperbolic tangent function, $\otimes$: element-wise (Hadamard) product, W , U: weight matrices that need to be learned, b: bias vector parameters that need to be learned).

$$
\begin{aligned}
z_t &= \sigma_g(W_z x_t + U_z h_{t-1} + b_z) \\
r_t &= \sigma_g(W_r x_t + U_r h_{t-1} + b_r) \\
\hat{h}_t &= \sigma_h(W_h x_t + U_h(r_t \otimes h_{t-1}) + b_h) \\
h_t &= z_t \otimes h_{t-1} + (1 - z_t) \otimes \hat{h}_t
\end{aligned}
$$

Long Short Term Memory (LSTM) is a type of RNN where the network can remember both short term and long term values. LSTM networks are the preferred choice than RNN when tackling complex problems. LSTM models are mostly used with time-series data. It is used in different applications such as Natural Language Processing (NLP), sentiment analysis, financial time series analysis, etc.

LSTM networks consist of LSTM units. Each LSTM unit merges to form an LSTM layer. An LSTM unit is composed of cells having input gate, output gate and forget gate. Three gates regulate the information flow. With these features, each cell remembers the desired values over arbitrary time intervals.

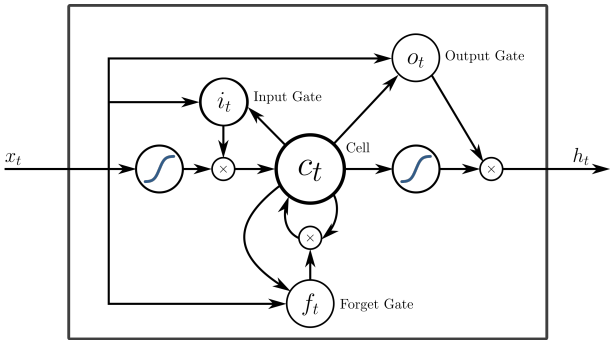


Figure 9: Single LSTM Unit

Equations below show the form of the forward pass of the LSTM unit (x_t: input vector to the LSTM unit, i_t, f_t, o_t: input, forget, output gate's activation vector,: h_t: output vector of the LSTM unit, c_t: cell state vector, σ_g: sigmoid function, σ_h: hyperbolic tangent function, $\otimes$: element-wise (Hadamard) product, W , U: weight matrices that need to be learned, b: bias vector parameters that need to be learned).

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
c_t &= f_t \otimes c_{t-1} + i_t \otimes \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
h_t &= o_t \otimes \sigma_h(c_t)
\end{aligned}
$$

LSTM is a specialized version of RNN. Therefore, the weight updates and preferred optimization methods are the same. In addition, the hyperparameters of LSTM are just like RNN. In order to find the best hyperparameters of LSTM, the hyperparameter optimization methods that are used for RNN are also applicable to LSTM.

LSTM and GRU are popular variants of RNN with long-term memory. Shudong Yang et al. [18] compares the performance differences of these models and found: in terms of model training speed, GRU is nearly 30% faster than LSTM for processing the same dataset; and in terms of performance, GRU performance will surpass LSTM in the scenario of small dataset, and inferior to LSTM in other scenarios. Considering the two dimensions of both performance and computing power cost, the performance-cost ratio of LSTM is higher than that of GRU. Therefore, for our studies, we will place greater importance on LSTM for training the model.

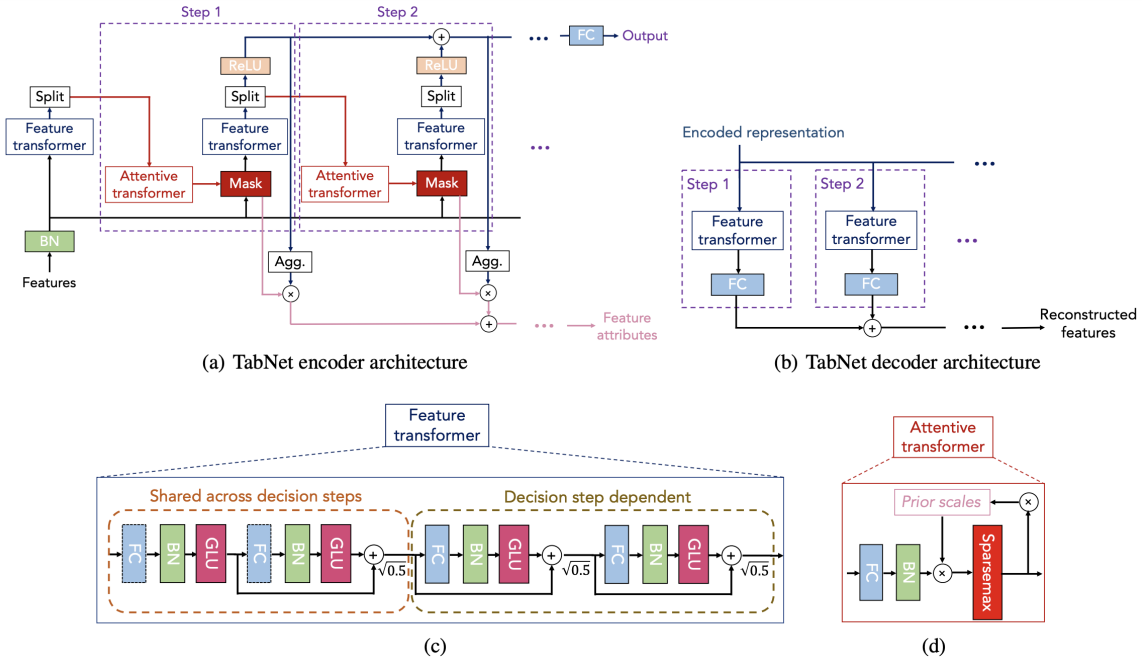
One thing we want to mention here is that LSTM transfer the time-series tabular data to 3D data and stack all the data together, therefore it needs more computational power than other ML and DL models. To facilitate modeling work, we first used Principal Component Analysis (PCA) to reduce the dimensions and normalize the data, and retain 80% explained variance in final model.

3.2.2 TabNet

TabNet, initially written by Arik and Pfister for Google Cloud AI [16] recently shows some promising results. TabNet is designed to learn a 'decision-tree-like' mapping in order to inherit the valuable benefits of tree-based methods (explainability) while providing the key benefits of deep learning-based methods (high performance and new capabilities).

There are three important parts of TabNet structure: Encoder Architecture, Feature Transformer and Attentive Transformer. The architecture basically consists of multi-steps which are sequential, passing the inputs from one step to another. If we take a single step, three processes happen:

1. Feature Transformer which is a four consecutive Gated Linear Unit(GLU) decision blocks
2. An Attentive Transformer that uses sparse-matrix to give sparse feature selection which enables interpretability and better learning as the capacity is used for the most salient features.
3. Mask which is used with the transformer to give out the decisions parameter: **n(d)** and **n(a)** which is then fed to the next step.



(a) TabNet encoder architecture (b) TabNet decoder architecture

(c) (d)

Figure 10: TabNet Logic

Tabnet Encoder Architecture. Initially the whole dataset with all the features is taken without any feature engineering. It is Batch Normalized (BN) and passed to the feature transformer where it passes through the 4-decision GLU steps to give two parameters: n(d) which is the output decision from that particular step giving its prediction of continuous numbers/classes in case of regression/classification; n(a) which is going as input to the next attentive transformer where the next cycle begins. After the attention transformer, where the different features along with their importance are figured out, the feature importance for that step is aggregated along with the other steps. This individual importance from the step($f(i)$) is then multiplied with the step's ($s(i)$) importance and added with other steps ($S(n)$) and its values ($F(n)$) to give out the final feature importance of the model. The decision output from the Feature transformers (n(d) or d[i]) is also aggregated and embedded in the form: $d_{out} = \sum_{i=1}^{N_{steps}} \text{ReLU}(d[i])$, and applied a linear mapping to get the final decision as output.

Feature Transformer. Feature Transformer has 4 consecutive blocks, a Fully Connected (FC) Layer followed by a Batch Normalization (BN) Layer followed by GLU. GLU stands for Gated linear unit which is just sigmoid of x multiplied by x ($\sigma(x) \otimes x$). So they consist of two shared decision steps followed by two independent decision steps. For robust learning, the layers are shared across two decision steps as the same input features are used in different steps. Normalization with $\sqrt{0.5}$ helps to stabilize learning by ensuring that the variance throughout the network does not change dramatically, and it gives two outputs n(d) and n(a).

Attentive Transformer. Attentive Transformer consists of an FC layer, BN layer, Prior scales layer, and Sparsemax layer. The n(a) input is passed into an FC layer followed by BN. It is then multiplied by the Prior scale which is a function that tells you how much you know about the features already from the previous steps and how many features have been used before in the steps. If it is set to 1 all features have equal importance. But the main advantage of TabNet is that it employs soft feature selection with controllable sparsity in end-to-end learning - a single model jointly performs feature selection and output mapping. So we have the parameter $P_0 = 1$ means all features are equal. $P_i = \prod_{j=1}^{i}(\gamma - M_j)$, the lower γ, the more independent steps are. If γ is close to 1 then we can select different features at every step or if it is larger than 1 then we can reuse the same features across multiple steps. The Sparsemax: $\sum_{i=1}^{n}$ sparsemax $(x)_i = 1, \forall x \in \mathbb{R}^n$ is like softmax, but instead of all features adding up to 1 some will be 0 and only the rest will add up to 1. This helps in making it an instance-wise feature selection where different features are taken at different steps and then fed to the mask layer which helps to identify the selected features.

Compared with other neural network models, TabNet is equipped with some superior advantages, including:

- Encode multiple data types (tabular data, time-series data, etc.) and use nonlinearity to solve.
- No need for Feature Engineering: we can throw all the columns and the model will pick the best features.
- TabNet neural network enables an inherent form of explainability that makes it possible to apply it to problems that require the model explanations.
- TabNet achieves high performance on a wide range of different tabular datasets.

Combined with all the strengths above, TabNet also shows great capacity in our model and is able to produce satisfactory trading results compared with other ML and DL algorithms.

4 Results

In this part, we show our results and performance from ML and DL models. We select the BTCUSDT pair as our target and define the target variable as the price movement direction of the next period (i.e. if the price move up we set the target variable as 1 otherwise 0). We set training period: 01/01/2022 - 01/13/2022; validation period: 01/24/2022 - 07/30/2022 and test period: 08/01/2022 - 11/30/2022. Backtests are also implemented in the test period. The 10-min frequency price data, technical indicators, and text sentiment data are applied to train our models.

4.1 Machine Learning Models (Baseline)

We applied all price, technical indicators and sentiment scores to fit a random forest model, an AdaBoost model and an XGBoost model. To prevent overfitting, we limit *max_depth* to 5 and set *n_estimators* to 20. Table 3, 4, 5 show their performances.

	Train			Validation			Test		
	precision	recall	f1	precision	recall	f1	precision	recall	f1
Negative	0.54	0.53	0.54	0.52	0.51	0.52	0.54	0.42	0.47
Positive	0.54	0.56	0.55	0.52	0.54	0.53	0.53	0.64	0.58
Accuracy			0.54			0.52			0.53
Macro avg	0.54	0.54	0.54	0.52	0.52	0.52	0.53	0.53	0.53

Table 3: Random Forest Performance

	Train			Validation			Test		
	precision	recall	f1	precision	recall	f1	precision	recall	f1
Negative	0.54	0.48	0.51	0.52	0.47	0.50	0.54	0.35	0.42
Positive	0.54	0.60	0.56	0.52	0.57	0.54	0.52	0.70	0.60
Accuracy			0.54			0.52			0.53
Macro Avg	0.54	0.54	0.54	0.52	0.52	0.52	0.53	0.53	0.51

Table 4: AdaBoost Performance

The accuracy and precision scores for RF, AdaBoost and XGBoost are almost the same, while recall scores show slight differences. XGBoost achieves the highest recall on positive, which means XGBoost has the capacity to capture the price up movement. However, this might not be a very great property for trading since it might incur huge losses for missing the down movement. All models tend to have higher recall and f1 score for positive than negative on the testing set.

	Train			Validation			Test		
	precision	recall	f1	precision	recall	f1	precision	recall	f1
Negative	0.54	0.44	0.49	0.53	0.41	0.46	0.54	0.35	0.42
Positive	0.53	0.63	0.58	0.52	0.63	0.57	0.52	0.71	0.60
Accuracy			0.54			0.52			0.53
Macro avg	0.54	0.54	0.53	0.52	0.52	0.52	0.53	0.53	0.51

Table 5: XGBoost Performance

4.2 Deep Learning Models

4.2.1 LSTM

Similarly, we applied all price, technical indicators and sentiment scores to fit a random forest model. However, rather than points, the input of LSTM are data series. Therefore we need to stack our dataset to a 3-d array before throwing it into the model. As we have a bunch of technical indicators and the frequency is relatively high, it is easily to exhaust the memory when stacking our dataset. To avoid compromising too much in the series length, we drop the CDL (Appendix A) indicators and apply the PCA method to the remaining technical indicators to reduce dimension. The number of technical features is reduced to 4 after the PCA transform. A total of 15 features are inserted into our LSTM model.

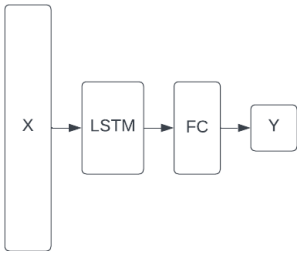


Figure 11: LSTM structure

The structure of our model is shown in figure 11. Our input is first fed to the LSTM module, then the output is fed to a full connection layer which gives a logit number, and finally the probability of up-movement is given. We set 60 (10 hours of 10-min data) as the length of the data series. The LSTM module has 2 layers and each layer has 4 cells. We apply cross entropy as our loss function and train our model. Table 6 shows the performance of the optimized LSTM model.

4.2.2 TabNet

Again, we applied all features and data we have to train a TabNet model without PCA, due to the feature engineering work already done by TabNet. Before training the classification model, we first perform a semi-supervised pre-training with the features. After that, we train a classification based on the pre-trained model. The performance of the optimized model is shown in Table 7.

We can see the results from Table 6 and 7. One interesting thing we want to mention is that the training recall scores for Negative and Positive in TabNet model are different, while other

	Train			Validation			Test		
	precision	recall	f1	precision	recall	f1	precision	recall	f1
Negative	0.53	0.54	0.53	0.52	0.52	0.52	0.55	0.38	0.44
Positive	0.54	0.54	0.54	0.52	0.53	0.52	0.53	0.69	0.60
Accuracy			0.54			0.52			0.53
Macro avg	0.54	0.54	0.54	0.52	0.52	0.52	0.54	0.53	0.52

Table 6: LSTM Performance

	Train			Validation			Test		
	precision	recall	f1	precision	recall	f1	precision	recall	f1
Negative	0.52	0.64	0.58	0.52	0.53	0.53	0.53	0.51	0.52
Positive	0.54	0.42	0.47	0.53	0.52	0.52	0.53	0.55	0.54
Accuracy			0.53			0.53			0.53
Macro avg	0.52	0.53	0.52	0.53	0.53	0.53	0.53	0.53	0.53

Table 7: TabNet Performance

models are close to each other. This generates more balanced recall scores in the testing period, which means the model generalizes better in the future period.

4.3 Comparison

In summary, DL models we built perform better than those ML tree-based baseline models in terms of accuracy score. Table 8 summarizes and compares the performance of all models we trained.

	Train acc.(%)	Validation acc.(%)	Test acc.(%)
Random Forest	54.22%	52.35%	53.18%
AdaBoost	53.85%	51.92%	52.60%
XGBoost	53.71%	52.37%	52.85%
LSTM	53.52%	52.11%	**53.36%**
TabNet	52.80%	52.59%	**53.34%**

Table 8: Model Performance

Although ML models perform well in the training set if we do not control overfitting, they generally perform poorly in the validation and testing sets. Tree-based models are more easily affected by noisy data, especially the high-frequency ones in our research. Therefore we turned to DL models for better performance and use their results to design trading strategies for backtesting.

5 Backtest

In this section, we design and apply two trading strategies, namely simple and volatility-adapting, to the predictions from the models and backtest these strategies.

5.1 Trading Strategy

5.1.1 Simple Strategy

Simply strategy is relatively naive. Since we can get the probability of positive return from predictions, we define a label L as the action we long(1), short(-1) or no position(0), and y is the predicted probability from models.

$$L_t = \begin{cases} 1, & y_t > 0.5, \\ -1, & y_t < 0.5, \\ 0, & other \end{cases}$$

For the simple strategy, there is no control on volatility risk and therefore the number of trades will be greater compared with Volatility Adapting (VA) strategy.

5.1.2 Volatility Adapting Strategy

Volatility Adapting Strategy is a strategy that uses historical Exponential Weighted Moving Average (EWMA) volatility and high-low spread to control the market risk. We define:

- σ_1: EWMA of squared 10-min returns, half-life set as 24 hours
- σ_2: EWMA of 10-min High-Low spread, half-life set as 24 hours

We set half-life as 24 hours, which means: if we stand at time t, the weight for the t minus 24 hours is 50%. σ_1 measures the relative volatility (returns), while σ_2 measures the absolute volatility (spread).

We define volatility percentile σ_1^{th}, σ_2^{th} and weighted average volatility percentile σ^{th}:

- σ_1^{th}: percentile of σ_1 in past 48 hours
- σ_2^{th}: percentile of σ_2 in past 48 hours
- $\sigma^{th} = \frac{1}{2}\sigma_1^{th} + \frac{1}{2}\sigma_2^{th}$: equal weighted average of σ_1^{th} and σ_2^{th}:

In this step, we have the same units for the two volatility. If σ_1 or σ_2 increase, their corresponding percentile also increases, suggesting the market is volatile.

For the volatility Adapting Strategy, we also define a Penalty Factor α. The Penalty Factor is a hyperparameter in our model and we need to adjust the number over time. For our strategy, we set α = 0.12. Note α can be changed as market condition changes. It also adjusts the threshold for trading.

$$L_t = \begin{cases} 1, & y_t > 0.5 + \alpha\sigma^{th}, \\ -1, & y_t < 0.5 - \alpha\sigma^{th}, \\ 0, & other \end{cases}$$

For VA strategy, we do control the volatility risk. If the market becomes volatile, we expected the percentile of the return volatility and high-low spread will increase. Therefore the threshold for trading will increase correspondingly, making our decision more conservative.

5.2 Backtest

We backtest all the ML and DL models and compare their performance with the buy-and-hold strategy. Every trading strategy outperforms the market if we do not consider commission. If we set the commission at 0.0001, the return will drop by 20%, due to the high-frequency trading. Metrics and Trend for different models and strategies are shown in Table 9 and Appendix B.

	Peak	Final	Sharpe	Drawdown	# Trades	Win Rate	Profit Factor
Random Forest	25.71%	-8.75%		-31.29%	3915	67.48%	1.05
Random Forest_VA	5.66%	-0.24%	0.21	-10.72%	2103	67.76%	1.05
AdaBoost	54.25%	29.04%	0.8	-27.86%	4249	67.15%	1.12
AdaBoost_VA	2.26%	0.22%	0.16	-9.34%	1165	66.52%	1.05
XGBoost	15.65%	6.63%	0.33	-24.16%	4176	68.20%	1.09
XGBoost_VA	2.75%	-0.96%		-20.02%	2425	66.47%	1.03
LSTM	55.74%	35.34%	0.83	-31.37%	4338	69.66%	1.12
LSTM_VA	22.48%	19.71%	1.51	-8.90%	2100	69.95%	1.15
TabNet	61.15%	44.32%	1.1	-26.16%	3998	67.11%	1.26
TabNet_VA	27.48%	25.70%	1.6	-8.72%	2040	68.43%	1.24

Table 9: Backtest Performance Summary

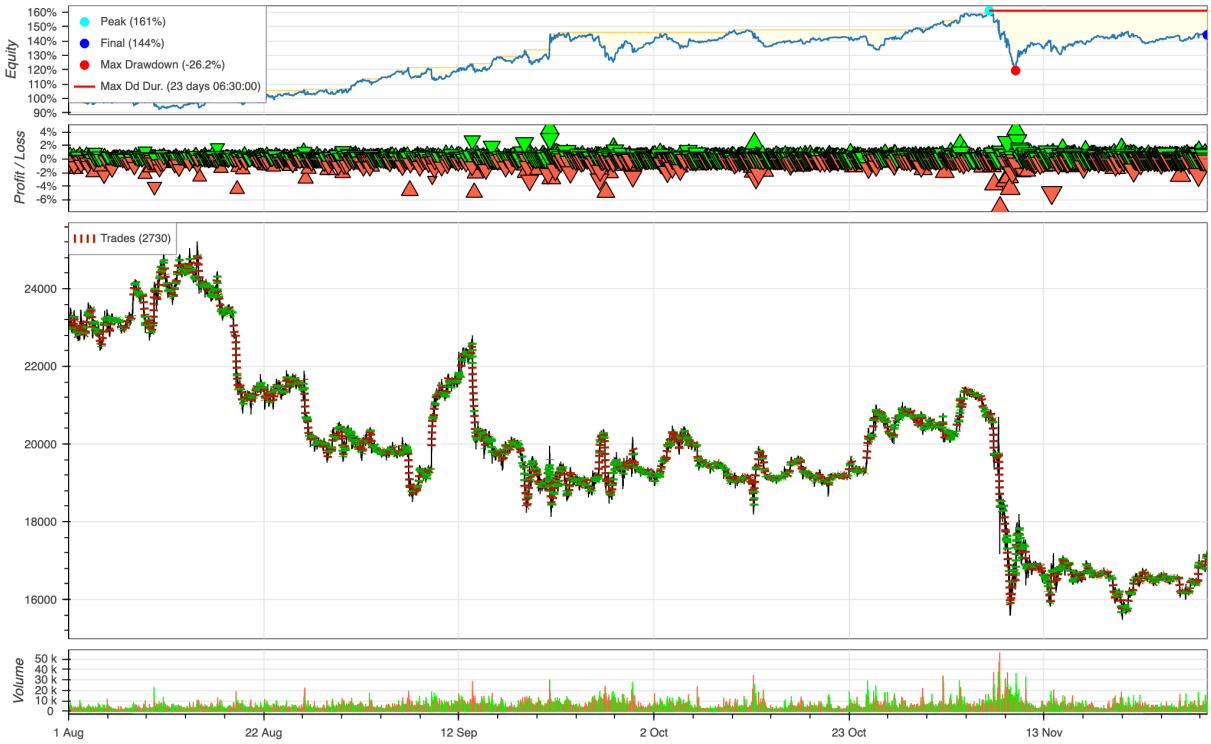


Figure 12: TabNet Simple Strategy Backtesting

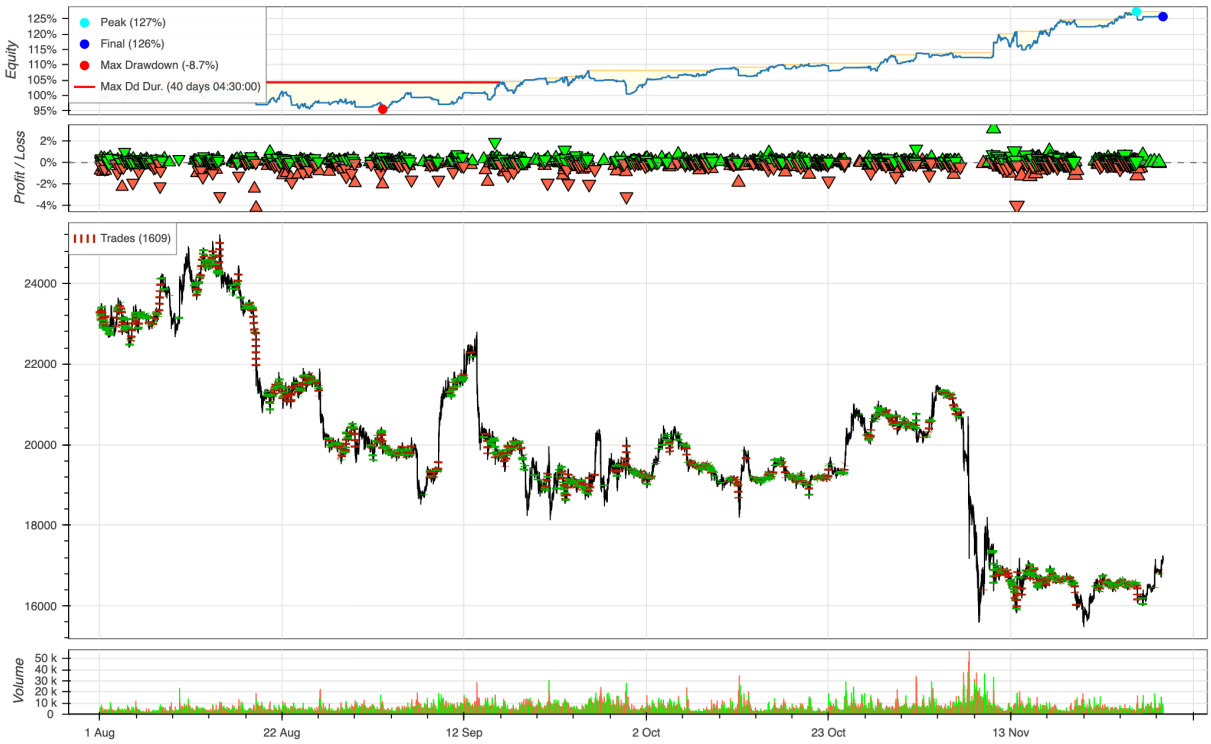


Figure 13: TabNet VA Strategy Backtesting

For DL models, VA Strategy do not reach a higher final value, but realize a higher sharpe ratio than simple strategy. The max drawdown for the VA strategy is much lower than simple strategy, which means VA has a greater impact on risk controlling and shows greater capacity when there is a signal for market downturn. Also, due to the changing threshold by market condition, the numbers of trades for VA strategy also are much less than simple strategy.

Figure 12 and 13 exhibit net value of the simple strategy and VA strategy. Simple strategy performs better in the early time but experiences a sharp drawdown in the later period. However, for VA strategy, even though it does not have a satisfactory performance at beginning, it shows superiority in the later drawdown and avoid losses on that period.

6 Conclusion

In this section we reached to our final conclusions. There are many interesting findings in this paper. Firstly, TabNet, the model with best performance, automatically conducts feature engineering while retain the explainability. This is a very important features since quantitative research are more focus on explainability rather than data mining. Second, we find that DL based models generally perform better in out-of-sample data than ML Baseline models. One thing we want to mention is that LSTM need tremendous computational power, especially for higher-dimensional data. Therefore we use PCA to reduce the dimension for LSTM model and it shows satisfactory results. Thirdly, Lastly, Volatility Adaptive Strategies have great capacity in controlling the market risk and preventing large losses when there is a signal for market downturn. It is a more conservative strategy with large sharpe ratio and small max drawdown.

There are many other things we can do in the future. First of all, since we only use naive label to differentiate the up/down movement of the price, other more advanced labeling method i.e. unsupervised learning might yield better results. Second, When fetching data from Binance, we collected 1-min level high-frequency data, but we only resampled to 10-min. In the future we can resample and explore other frequencies to see whether the results will be better. Lastly, we have data for other top-15 market capitalization cryptos but only use BTCUSDT pair. We can design portfolio strategies on different cryptos other than on single underlying.

References

[1] O. B. Sezer, M. U. Gudelek, and A. M. Ozbayoglu, "Financial time series forecasting with deep learning: A systematic literature review: 2005-2019," *Applied Soft Computing*, vol. 90, 2020.

[2] L. Alessandretti, A. ElBahrawy, L. M. Aiello, and A. Baronchelli, "Anticipating cryptocurrency prices using machine learning," *Complexity*, 2018.

[3] V. Derbentsev, V. Babenko, K. Khrustalev, H. Obruch, and S. Khrustalova, "Comparative performance of machine learning ensemble algorithms for forecasting cryptocurrency prices," *International Journal of Engineering*, vol. 34, no. 1, pp. 140–148, 2021.

[4] S. Lahmiria and S. Bekiros, "Cryptocurrency forecasting with deep learning chaoti neural networks," *Chaos, Solitons Fractals*, vol. 118, pp. 35–40, 2019.

[5] M. M. Patel, S. Tanwar, R. Gupta, and N. Kumar, "A deep learning-based cryptocurrency price prediction scheme for financial institutions," *Journal of Information Security and Applications*, vol. 55, no. 102583, pp. 2214–2126, 2020.

[6] E. Pintelas, I. E. Livieris, S. Stavroyiannis, T. Kotsilieris, and P. Pintelas, "Investigating the problem of cryptocurrency price prediction: a deep learning approach," *IFIP International conference on artificial intelligence applications and innovations*, pp. 99–110, 2020.

[7] S. Biswas, M. Pawar, S. Badole, N. Galande, and S. Rathod, "Cryptocurrency price prediction using neural networks and deep learning," *2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS)*, vol. 1, pp. 408–413, 2021.

[8] M. Ortu, N. Uras, C. Conversano, S. Bartolucci, and G. Destefanis, "On technical trading and social media indicators for cryptocurrency price classification through deep learning," *Expert Systems with Applications*, vol. 198, p. 116804, 2022.

[9] M. Chen, N. Narwal, and M. Schultz, "Predicting price changes in ethereum," *International Journal on Computer Science and Engineering (IJCSE) ISSN*, pp. 0975–3397, 2019.

[10] E. Akyildirim, A. Goncu, and A. Sensoy, "Prediction of cryptocurrency returns using machine learning," *Annals of Operations Research*, vol. 297, no. 1, pp. 3–36, 2021.

[11] D. Mayo and H. Elgazzar, "Predicting cryptocurrency price change direction from supply-side factors via machine learning methods," *2022 IEEE World AI IoT Congress (AIIoT)*, pp. 330–336, 2022.

[12] A. Ibrahim, "Forecasting the early market movement in bitcoin using twitter's sentiment analysis: An ensemble-based prediction model," *2021 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS)*, pp. 1–5, 2021.

[13] D.-H. Kwon, J.-B. Kim, J.-S. Heo, C.-M. Kim, and Y.-H. Han, "Time series classification of cryptocurrency price trend based on a recurrent lstm neural network," *Journal of Information Processing Systems*, vol. 15, no. 3, pp. 694–706, 2019.

[14] S. Tanwar, N. P. Patel, S. N. Patel, J. R. Patel, G. Sharma, and I. E. Davidson, "Deep learning-based cryptocurrency price prediction scheme with inter-dependent relations," *IEEE Access*, vol. 9, pp. 138 633–138 646, 2021.

[15] T. Shintate and L. Pichl, "Trend prediction classification for high frequency bitcoin time series with deep learning," *Journal of Risk and Financial Management*, vol. 12, no. 1, 2019.

[16] S. Arik and T. Pfister, "Tabnet: Attentive interpretable tabular learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 8, pp. 6679–6687, 2021.

[17] Y. Zheng, "Order flow, technical analysis and neural network: Predicting short-term direction of futures contract," *Institute of Electrical and Electronics Engineers (IEEE)*, 2022.

[18] S. Yang, X. Yu, and Y. Zhou, "Lstm and gru neural network performance comparison study: Taking yelp review dataset as an example," pp. 98–101, 2020.

A Appendix: Technical Indicators

Symbol	Indicator Meaning
Overlap Studies	
BBANDS	Bollinger Bands
DEMA	Double Exponential Moving Average
EMA	Exponential Moving Average
HT_TRENDLINE	Hilbert Transform - Instantaneous Trendline
KAMA	Kaufman Adaptive Moving Average
MA	Moving average
MAMA	MESA Adaptive Moving Average
MAVP	Moving average with variable period
MIDPOINT	MidPoint over period
MIDPRICE	Midpoint Price over period
SAR	Parabolic SAR
SAREXT	Parabolic SAR - Extended
SMA	Simple Moving Average
T3	Triple Exponential Moving Average (T3)
TEMA	Triple Exponential Moving Average
TRIMA	Triangular Moving Average
WMA	Weighted Moving Average
Momentum Indicators	
ADX	Average Directional Movement Index
ADXR	Average Directional Movement Index Rating
APO	Absolute Price Oscillator
AROON	Aroon
AROONOSC	Aroon Oscillator
BOP	Balance Of Power
CCI	Commodity Channel Index
CMO	Chande Momentum Oscillator
DX	Directional Movement Index
MACD	Moving Average Convergence/Divergence
MACDEXT	MACD with controllable MA type
MACDFIX	Moving Average Convergence/Divergence Fix 12/26
MFI	Money Flow Index
MINUS_DI	Minus Directional Indicator
MINUS_DM	Minus Directional Movement
MOM	Momentum
PLUS_DI	Plus Directional Indicator
PLUS_DM	Plus Directional Movement
PPO	Percentage Price Oscillator
ROC	Rate of change : ((price/prevPrice)-1)*100
ROCP	Rate of change Percentage: (price-prevPrice)/prevPrice
ROCR	Rate of change ratio: (price/prevPrice)
RSI	Relative Strength Index
STOCH	Stochastic
STOCHF	Stochastic Fast
STOCHRSI	Stochastic Relative Strength Index
TRIX	1-day Rate-Of-Change (ROC) of a Triple Smooth EMA
ULTOSC	Ultimate Oscillator
WILLR	Williams' %R
Volume Indicators	
AD	Chaikin A/D Line
ADOSC	Chaikin A/D Oscillator
OBV	On Balance Volume
Volatility Indicators	
ATR	Average True Range
NATR	Normalized Average True Range
TRANGE	True Range
Cycle Indicators	
HT_DCPERIOD	Hilbert Transform - Dominant Cycle Period
HT_DCPHASE	Hilbert Transform - Dominant Cycle Phase
HT_PHASOR	Hilbert Transform - Phasor Components
HT_SINE	Hilbert Transform - SineWave
HT_TRENDMODE	Hilbert Transform - Trend vs Cycle Mode

Symbol	Indicator Meaning
Pattern Recognition	
CDL2CROWS	Two Crows
CDL3BLACKCROWS	Three Black Crows
CDL3INSIDE	Three Inside Up/Down
CDL3LINESTRIKE	Three-Line Strike
CDL3OUTSIDE	Three Outside Up/Down
CDL3STARSINSOUTH	Three Stars In The South
CDL3WHITESOLDIERS	Three Advancing White Soldiers
CDLABANDONEDBABY	Abandoned Baby
CDLADVANCEBLOCK	Advance Block
CDLBELTHOLD	Belt-hold
CDLBREAKAWAY	Breakaway
CDLCLOSINGMARUBOZU	Closing Marubozu
CDLCONCEALBABYSWALL	Concealing Baby Swallow
CDLCOUNTERATTACK	Counterattack
CDLDARKCLOUDCOVER	Dark Cloud Cover
CDLDOJI	Doji
CDLDOJISTAR	Doji Star
CDLDRAGONFLYDOJI	Dragonfly Doji
CDLENGULFING	Engulfing Pattern
CDLEVENINGDOJISTAR	Evening Doji Star
CDLEVENINGSTAR	Evening Star
CDLGAPSIDESIDEWHITE	Up/Down-gap side-by-side white lines
CDLGRAVESTONEDOJI	Gravestone Doji
CDLHAMMER	Hammer
CDLHANGINGMAN	Hanging Man
CDLHARAMI	Harami Pattern
CDLHARAMICROSS	Harami Cross Pattern
CDLHIGHWAVE	High-Wave Candle
CDLHIKKAKE	Hikkake Pattern
CDLHIKKAKEMOD	Modified Hikkake Pattern
CDLHOMINGPIGEON	Homing Pigeon
CDLIDENTICAL3CROWS	Identical Three Crows
CDLINNECK	In-Neck Pattern
CDLINVERTEDHAMMER	Inverted Hammer
CDLKICKING	Kicking
CDLKICKINGBYLENGTH	Kicking - bull/bear determined by the longer marubozu
CDLLADDERBOTTOM	Ladder Bottom
CDLLONGLEGGEDDOJI	Long Legged Doji
CDLLONGLINE	Long Line Candle
CDLMARUBOZU	Marubozu
CDLMATCHINGLOW	Matching Low
CDLMATHOLD	Mat Hold
CDLMORNINGDOJISTAR	Morning Doji Star
CDLMORNINGSTAR	Morning Star
CDLONNECK	On-Neck Pattern
CDLPIERCING	Piercing Pattern
CDLRICKSHAWMAN	Rickshaw Man
CDLRISEFALL3METHODS	Rising/Falling Three Methods
CDLSEPARATINGLINES	Separating Lines
CDLSHOOTINGSTAR	Shooting Star
CDLSHORTLINE	Short Line Candle
CDLSPINNINGTOP	Spinning Top
CDLSTALLEDPATTERN	Stalled Pattern
CDLSTICKSANDWICH	Stick Sandwich
CDLTAKURI	Takuri (Dragonfly Doji with very long lower shadow)
CDLTASUKIGAP	Tasuki Gap
CDLTHRUSTING	Thrusting Pattern
CDLTRISTAR	Tristar Pattern
CDLUNIQUE3RIVER	Unique 3 River
CDLUPSIDEGAP2CROWS	Upside Gap Two Crows
CDLXSIDEGAP3METHODS	Upside/Downside Gap Three Methods
Statistic Functions	
BETA	Beta
CORREL	Pearson's Correlation Coefficient (r)
LINEARREG	Linear Regression
LINEARREG_ANGLE	Linear Regression Angle
LINEARREG_INTERCEPT	Linear Regression Intercept
LINEARREG_SLOPE	Linear Regression Slope
STDDEV	Standard Deviation
TSF	Time Series Forecast
VAR	Variance

B Appendix: Backtesting Results

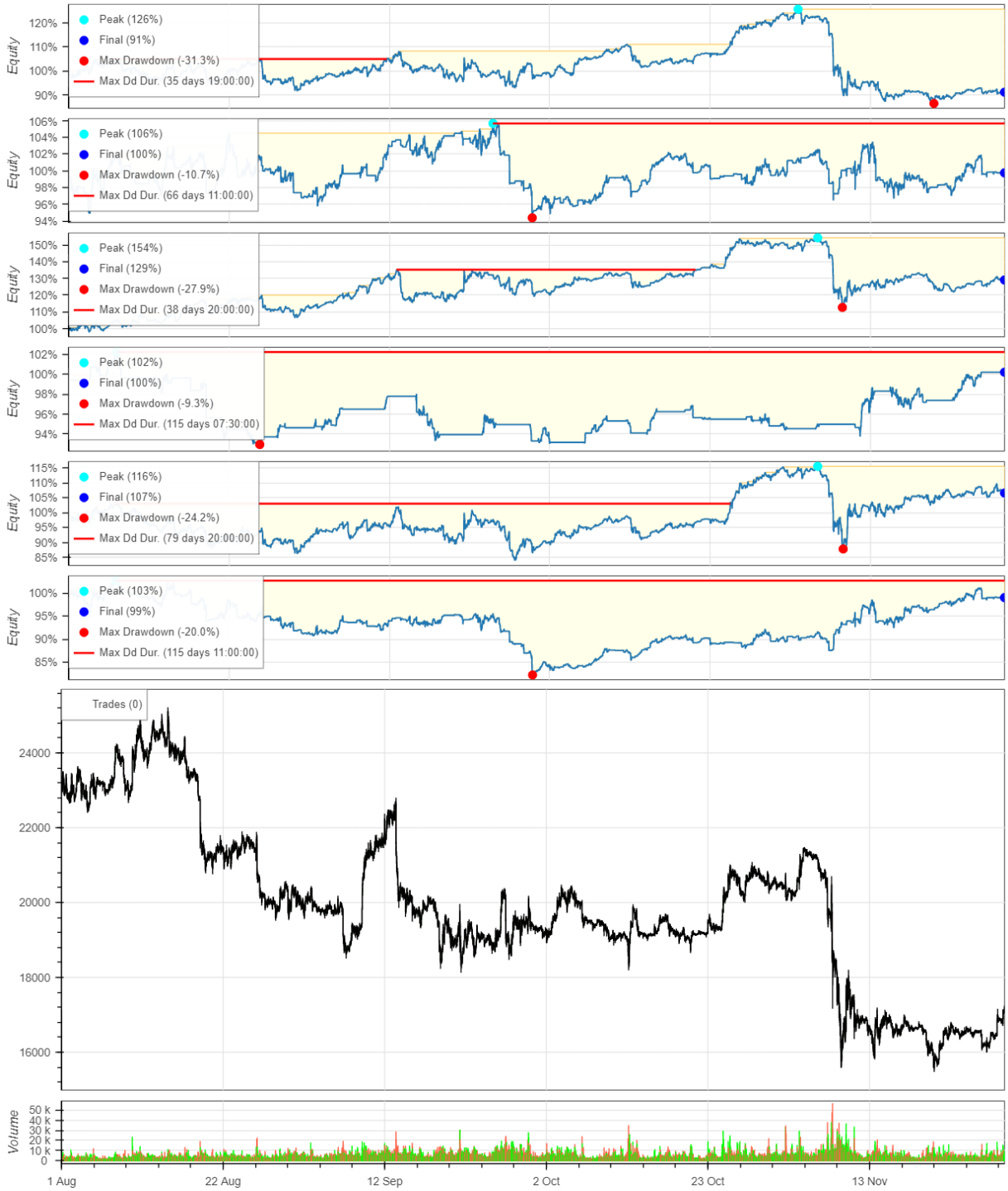


Figure 14: Backtest result of ML models

From top to bottom: (1) Performance of Random Forest Model with simple strategy; (2) Performance of Random Forest Model with volatility adapting strategy; (3) Performance of AdaBoost Model with simple strategy; (4) Performance of AdaBoost Model with volatility adapting strategy; (5) Performance of XGBoost Model with simple strategy; (6) Performance of XGBoost Model with volatility adapting strategy; (7) Price movement of BTCUSDT pair (8) BTCUSDT Trading volume

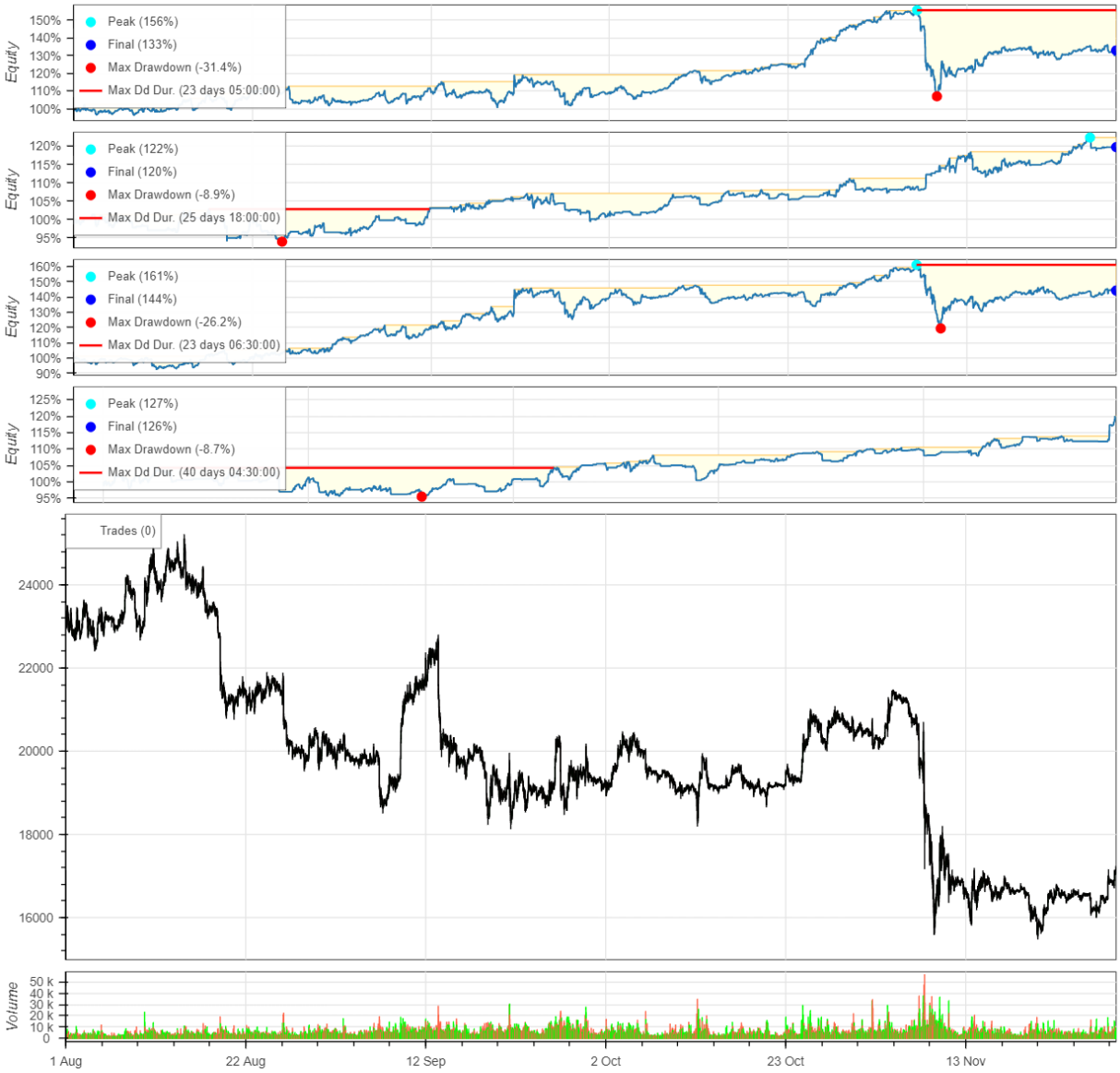


Figure 15: Backtest result of Deep Learning models

From top to bottom: (1) Performance of LSTM Model with simple strategy; (2) Performance of LSTM Model with voaltility adapting strategy; (3) Performance of Tabnet Model with simple strategy; (4) Performance of Tabnet Model with volatility adapting strategy; (5) Price movement for BTCUSDT pair; (6) BTCUSDT Trading volume