Supervised Classification on Crypto Time Series

Louis Wong
Georgia Tech
lwong64@gatech.edu

Abstract

In this paper, we explored various supervised algorithms, namely, neural networks, decision trees, boosting trees, and k-nearest neighbors, to predict binary classifications in time series analysis for the cryptocurrency financial market data Bitcoin and Ethereum. Given the constant changes in financial time series data, we investigated the ability of machine learning supervised algorithms to make successful predictions against it. This paper discusses the mentioned algorithms, provides an in-depth technical analysis, and explores the predictability, machine learning techniques, and different challenges faced via each method. Finally, we concluded our findings to show which architecture space was successful and how we could improve our design for further research.

1. Introduction

Cryptocurrency has emerged as one of the most vibrant financial markets in recent times. Traditionally, markets comprised only of stock equity markets, option trading, futures, foreign exchange, and commodities market. The new cryptocurrency market has become a hot area for investors and traders to explore. One significant difference in the crypto market, unlike the stock market, is that it operates 24/7 without breaks and halts. This not only makes it more exciting but also provides more opportunities for quantitative analysis to explore possible inefficiencies. In this study, we explored a list of supervised machine learning techniques on the BTC-USD and ETH-USD historical data [11].

2. Approach

2.1. Initial Problem

For this study, we gathered two different datasets to formulate different classification problems to explore. 1) We collected daily historical data for BTC-USD (Bitcoin to US-Dollars). Based on the past t days of data, we attempted to predict a binary classification result, indicating whether the market would go up or down the next day. 2) We collected 15-minute historical data for ETH-USD (Ethereum to US-Dollars). Based on the given t hours of data, we tried to detect whether the next action would be an abnormal positive action. We defined an anomalous action as an extreme positive movement that historically occurred only 10% of the time [3].

For the first problem, we aimed to explore market data as an average investor looking at the price of daily actions: open, high, low, close. Based on past trends, we wanted to predict the binary result of the next trading day. Such information and models are very valuable to short-term traders, whose goal is to gain short-term profit. The second problem is an anomaly detection system for monitoring higher interval daily trading activity. Such detection can be beneficial for larger algorithmic trading systems, typically used by hedge funds and equity firms to exploit market inefficiency. Note that the given look-back periods can be arbitrarily chosen; for this experiment, we explored whether there are optimal time-series look-back periods for such machine learning models.

2.2. Data collection

Numerous sources offer publicly available crypto data. We provided a link that showcases a list of cryptocurrency market financial data here: Rapid API. We used a private access API to gather such data; for convenience in replicating our results, we also provided the dataset as a CSV file, along with our source code.

2.3. Environment Setup

In this study, we primarily used JupyterLab for all the experiments. Besides the standard Python 3 packages, we utilized scikit-learn for both preprocessing and implementation, and TensorFlow Keras for the neural network models [5, 1]. For scikit-learn and Keras documentation, refer to scikit-learn and Keras. We also enabled CUDA to train with the Nvidia GPU to optimize the training performance.

2.4. Dataset and Preprocessing

The first dataset contained daily market data of BTC (Bitcoin) to USD, encompassing about 3 years of data. This dataset comprised 971 rows and 9 columns, featuring time in UTC, OHLC (Open, High, Low, Close) values, as well as the volume, the trade counts from the exchange, and the VWAP (Volume Weighted Average Price), which is commonly used by investors. The second dataset was a short interval intraday dataset of ETH (Ethereum) to USD. Similarly, this dataset included OHLC values, among other information. The data covered a time period of 1 year. Due to the 24-hour nature and intraday intervals of this dataset, it contained a total of 34,929 rows, which was relatively larger compared to the first dataset, even though the total time period was shorter.

2.4.1 K-Fold Cross-Validation

For both dataframes, we initially decided to split the data into a 70/30 ratio for the training and testing sets. However, the recent 30% of the dataset didn't seem to reflect the overall dataset very well. Consequently, any model was highly likely to struggle with predicting such distribution, as shown in Figure 4. To set up our experiment for success, we decided to split the dataset in a manner similar to K-Fold cross-validation [2]. Although we didn't specify the number of K, it was roughly about $K = 6$ on the BTC-USD dataset. Similarly, the ETH-USD dataset setup was in a similar manner, about $K = 10$. More importantly, the goal was to have the distribution of the test set closely resemble that of the training set, giving the model a fair chance at making accurate predictions post-training, as shown in Figure 5.

2.4.2 Log Difference Normalization

The market's time series data was neither normalized nor stationary; the prices in the historical trend exhibited significant skewness towards higher and lower trends. It was critical to normalize such data before proceeding with any supervised machine learning models. There were multiple ways to normalize data, such as using min-max or standard normal methods, but since this was a time series-based model, where prices essentially resembled a random walk Markov chain. It is a common practice to capturing the delta difference and rate of change between t and t-1. To attain some properties and reduce the exponential nature in the final market analysis, calculating the log return was often standard practice. The formula was as follows:

$$r_t = \ln\left(\frac{P_t}{P_{t-1}}\right)$$

$$\begin{aligned} r_t &: \text{Log return at time } t, \\ P_t &: \text{Price at time } t, \\ P_{t-1} &: \text{Price at time } t-1, \\ \ln &: \text{Natural logarithm.} \end{aligned}$$

For the input set, the log difference was applied across the OHLC, while for the y-set (truth label), the binary class was applied to the log return, indicating whether t+1 was positive or negative.

2.4.3 Anomaly Positive Outlier

For the high interval ETH-USD dataset, we were interested in detecting anomalous positive outliers. In this case, we defined an anomalous return as one that was above the 90th percentile, which, on average, only occurred about 10% of the time.

2.4.4 Pearson Correlation

Our basic correlation analysis on the auto time series indicated that there was almost no correlation between returns from t to t-1, suggesting that this was a rather difficult and non-trivial problem [12]. The Pearson correlation was -0.0737 for the first dataset and -0.0754 for the second. These results appeared random and noisy, which was consistent with many other quantitative analyses. The market seemed quite random, with relationships between future movements and past historical events often being complex and non-linear. However, the distribution graph showed that the log return resembled a normal distribution with higher density around the mean, zero, suggesting that the market was what most people would describe as a zero-sum game. Rather than saying the market return was random, it would be more accurate to describe it as normally distributed randomness.

2.4.5 Imbalance Classification

After analyzing the distribution of binary log returns in the BTC-USD dataset, we found that both the positive and negative classes were approximately balanced, with a near 50/50 split, which was ideal for a binary classification problem. However, the ETH-USD dataset exhibited a high skew, given our focus on the data in the 90th percentile of log returns. To achieve a balanced class distribution in the training data, we needed to apply a balancing technique. Various strategies were available, such as upsampling, downsampling, or employing methods like SMOTE or class-weighted loss mechanisms such as focal loss [4, 9]. SMOTE

creates a synthetic class based on the minority class, generating new data by combining attributes from the minority data group. Although this approach could rectify the balance issue, it might lead the model to overfit to synthetic data, which doesn't represent real-world trends, potentially resulting in noise overfitting. Focal loss, however, was more specifically designed for neural network models. Given that we were experimenting with a variety of different types of supervised models, a more general approach was preferable. We developed a unique solution that combined both upsampling and downsampling, which will be elaborated upon in the experiment section.

3. Experiments and Results

3.1. Initial Comparison

Initially, we utilized 5 different algorithms to fit the training data. The algorithms are as follows: Decision Tree, Neural Network, Boosting, SVM, and K-Nearest Neighbors.

3.1.1 Loss and Error

Given that each algorithm is quite different, they use distinct loss metrics to fit the model. For instance, Neural Networks use binary cross-entropy, Decision Trees employ Gini Impurity, and SVMs utilize hinge loss, among others. For the BTC-USD experiment, we compared them using the standard metric of accuracy, which is relatively simple to interpret in this binary classification problem. The formula for accuracy is given by

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

The error or loss defined for each model will be $1 -$ Accuracy. Using generalized approaches, we can compare the performance of the different models.

On the other hand, the ETH-USD experiment cannot simply use accuracy to define the success of the model due to the imbalanced nature of the anomaly detection algorithm, which takes into account different factors such as true and false positives, precision, and recall. We used the F1 score to compare different algorithms, which is the preferred practice when dealing with imbalanced classes. The formula can be written as follows:

$$F1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

where

$$\text{precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

and

$$\text{recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

3.1.2 Over-fitting

For the BTC-USD experiment, we partitioned our training data into fractional sets, gradually introducing more data, from 10% to 100% of the training set. The results in Figure 6 show that some models tend to overfit more easily, while others continue to improve. For example, the neural network (LSTM) seems to continually reduce its training and validation errors as the amount of training data increases. In contrast, Boosting and SVM are only fitting to the training set, with the testing error worsening after utilizing 80% of the training data. This seems to suggest that there are inherent architectural biases when training these models. The Neural Network, especially the LSTM network used in our experiment, is relatively more complex compared to the other supervised algorithms. This initial result shows that, by default (without much tuning), the neural network can accommodate larger datasets for training.

3.1.3 Hybrid Resampling

As mentioned previously, the ETH-USD experiment requires balanced preprocessing. However, we do not want to create additional synthetic samples using the conventional SMOTE technique. For this experiment, we devised unique approaches that utilize both upsampling and downsampling, guided by an adjustable hyperparameter r, where r represents the ratio of resampling used. As r increases, it amplifies the amount of upsampling used, whereas a lower r value reduces the majority sample instead (downsampling). We tested various r values against 5 algorithms and obtained the validation results. The best validation F1 score was achieved at $r = 0.4$, leaning towards the downsampling technique. However, the testing set revealed that the optimal score was at $r = 0.1$. Without access to the test set, there would have been no way to ascertain whether upsampling or downsampling is generally better for our case. Our validation results at least somewhat suggest a preference for the lower side, indicating that this technique was successful.

3.1.4 Initial Performance Results

Looking at the initial performance BTC-USD results at the Table 1, we must admit the predictive difficulty of this problem. Almost all models do not perform better than random chance. This indicates the noise in the market and the random nature of Markov Chain time series. However, a small

Model	Train Accuracy	Test Accuracy	Time Used (Min:Sec)
Neural Network	0.693	0.533	20:23
SVM	0.705	0.493	00:42
Decision Tree	1.0	0.49	00:28
k-NN	0.671	0.49	0:55
Boosting	0.958	0.463	8:24

Table 1: Initials performance results on BTC-USD.

increase in predictability can often amplify to huge profits. Therefore, even a small percentage of improvement can have significant implications in financial rewards. That said, we continue to diagnose this analysis.

Unsurprisingly, we noticed that the best-performing model that stood out the most is the neural network model LSTM, which evaluated with an accuracy of 53%, while the rest of the models fell below 50%. SVM and KNN came in closely at 2nd and 3rd place, respectively, with about 49% accuracy. The decision tree tends to be highly overfit to the training data, with a score of 1 in accuracy and 47% in the testing set. Hyper-tuning may benefit significantly in such cases, which will be our next step.

In terms of time performance, although the LSTM shows significant performance, it is almost 10-20 times slower than the other models. If analyzing a huge dataset with limited computational resources is a constraint, then neural network types of models may not be the best choice. In this case, a single decision tree is extremely fast, taking only 0.28 seconds to train on the stacking dataset. For the ETH-USD results at the Table 2.

Model	Train F1	Test F1	Time Used (Min:Sec)
Boosting	0.797	0.225	5:25
SVM	0.750	0.213	11:44
Neural Network	0.668	0.179	1:32
Decision Tree	1.000	0.176	0:23
k-Nearest Neighbors	0.676	0.165	0:16

Table 2: Initial performance results on ETH-USD.

In this experiment, we observe that the speed is consistent between both datasets, despite the differences in the number of data points and the nature of the datasets. The decision tree emerged as the fastest, completing in 23 seconds, while the neural network (LSTM) took longer, finishing in 1 minute and 32 seconds. It is evident that the complexity of these algorithms operates on different scales. Noticeably, SVM computation time is significantly worse than the first experiment, indicating that the model is not efficient at scale. There are the inherent challenges of this detection problem, the initial results of this algorithm are quite poor, failing to capture patterns successfully. Generally, an F1 score below 0.5 is considered to indicate poor performance. The task of short-interval detection is inherently difficult. Over the next few phases, we plan to continue optimizing the algorithm, with the hope of improving its performance.

3.2. Technical Indicators

Technical indicators are often suggested by traditional quantitative analysis and many other financial traders and investors. However, do they actually offer any predictive value, or are they just an old-school practice that is no longer relevant in today's sophisticated machine learning and deep neural network applications? Based on our empirical study, we conclude that traditional indicators are indeed quite beneficial even on machine learning models.

We incorporated some of the popular technical indicators, such as SMA (Simple Moving Average), which can be defined as:

$$\text{SMA} = \frac{1}{n}\sum_{i=1}^{n} P_i$$

RSI (Relative Strength Index), defined as:

$$\text{RSI} = 100 - \left(\frac{100}{1 + RS}\right)$$

where $RS = \frac{\text{Average Gain}}{\text{Average Loss}}$ over a specified period, and Bollinger Bands, which involve a set of trendlines plotted two standard deviations (positively and negatively) away from a simple moving average of a security's price.

3.2.1 Normalization

Given that the nature of these indicators is to observe the relationship between the technical indicators and price movements, instead of performing an auto log difference, we normalized them using the indicator value at time t divided by the close price at time t. The result is a value ranging from -1 to 1, which is a similar outcome to a standard normalization operation, while preserving the structural relationship between price and indicators. The RSI, however, usually ranges from 0 to 100 in value; we divided it by its mean of 50. The min-max normalization can also work well in this situation since both upper and lower bound is fixated.

3.2.2 Experiment Results

We tested the results both with and without the indicators on the BTC-USD, as shown in Table 3.

Model	Test Accuracy	Test Accuracy w/o Indicators	Indicators Gain
Neural Network	0.533	0.517	0.017
SVM	0.493	0.490	0.003
Decision Tree	0.490	0.497	-0.007
k-Nearest Neighbors	0.490	0.49	0.0
Boosting	0.463	0.493	-0.03

Table 3: Comparison of Model Performance with and without Technical Indicators

We noticed that two models showed positive gains, while one remained neutral. Although it was not beneficial in every single model, some model certainly improved. For instance, the Neural Network experienced a boost from 51.7% to 53.3% in accuracy.

Contrarily, on the ETH-USD dataset, the technical indicators did not improve the model performance; in fact, they showed a clear decrease in performance across all five algorithms. We believed this was because the intraday algorithm was already fitting to noise without capturing much of a pattern, and the indicators were creating extra noise. On the other hand, the daily interval BTC-USD dataset had a lot less noise and benefited from the technical indicators as they provided additional feature descriptions.

3.3. Hyperparameters Optimization

After the initial results and comparison, we believed there was a need to improve our model performance. First, we optimized the data process in general across all models, then we tuned each model one by one based on their architecture design and the hyperparameters of those models.

3.3.1 Optimizing Lookback Periods

The difference between a normal dataset and a time series dataset is that the latter has time-dependent data denoted by t, each $t-1$, $t-2$, ..., to $t-n$, and so on. The algorithm aimed to predict $t+1$. Given that most architectures have a fixed input size, n had to be a fixed number in $t-n$. However, the selection of n was not straightforward. In auto time series data analysis, people often used autocorrelation to eliminate non-correlated $t-n$ values, but in our case, even $t-1$ had a very low correlation, close to zero, making this technique unfeasible due to the non-linear nature of our dataset. The only way to choose the best $t-n$ was through empirical results.

Considering this as a hyperparameter tuning process, we should not have performed it on the final testing results, as that would have amounted to data leakage and look-ahead bias. In real-time, we would never have access to such future data to choose the best n. Therefore, we further split our training set into a validation set using random sampling, allocating 80% to the training set and 20% to the validation set to evaluate the best $t-n$. Then, we could compare the actual testing evaluation. However, we had to remind ourselves that this was merely for experimental interpretation; we could not use testing evaluation to select our hyperparameter values.

Figure 1 presented the results obtained using a lag of 5 to a lag of 45, which ranged from $t-5$ to $t-45$. Based on this chart, we selected the maximum validation accuracy at $t-40$. Compared to our arbitrary choice of $t-10$, we observed improvements in both the validation and testing sets. In a real-time perspective, this equated to 40 days.

Similarly, we experimented with the interval on the ETH-USD dataset, achieving maximum validation F1 at $t-25$, a result that was consistent with the test set. In real-time, this corresponded to approximately 6 hours and 15 minutes. Since the validation and testing results were consistent in both experiments, especially with the intraday ETH-USD data, we concluded that this optimization technique was successful. It demonstrated the existence of an optimal lookback period, as evidenced by our experiments.

3.3.2 k-Nearest Neighbors (k-NN)

The k-Nearest Neighbors (k-NN) is a supervised algorithm known as instance-based learning, where the algorithm learns and makes predictions using just local approximations. It is one of the simplest algorithms; it classified new samples based on similarity to the k closest data points in the training set using a distance metric, such as Euclidean distance. However, it could be quite sensitive to the choice of k.

One popular k-NN optimization technique was known as the "elbow method". By examining the loss curve, there was a point where increasing the number of k no longer reduced the loss sharply. For this reason, we chose the left corner value of k as the chosen hyper-parameter. It was also common practice to select an odd number for k to avoid ties. Our loss curve seemed noisy, Figure 3, as the error rate fluctuated up and down. However, there were two clear local minima at $k=5$ and $k=15$. In this case, we opted for the smaller value of k, since increasing the complexity often led to overfitting the training or validation set. Given that the testing set was often not equivalent to the training set, a generalized approach was often the optimal strategy in machine learning.

For the ETH-USD experiment, we observed a more pro-

nounced resemblance to a discrete curve on Figure 7. If we selected the hyperparameter value using the "elbow method", that would have been $k = 5$. Interestingly, upon examining the test set results, it revealed that this was indeed the best hyperparameter choice, as the testing results continued to decrease after $k = 5$, indicating it was becoming overfitted. That said, the "elbow method" definitely held its place for hyperparameter tuning in k-NN algorithms.

3.3.3 Decision Tree

The decision tree is a supervised learning model that utilized information theory. It calculated the minimum entropy to split features apart. The algorithm resembled a tree-like structure, arranging features in a hierarchical order to maximize information gain, denoted as IG. The formula for information gain was given by:

$$\mathbf{IG} = \text{Entropy(parent)} - \sum(\text{proportion of samples in child} \times \text{Entropy(child)})$$

The more pure the nodes were, the higher the order of importance was ranked. The algorithm iterated to rank the features until every feature's rank was assigned. However, if a dataset contained numerous features, not all of them were particularly helpful for prediction. Therefore, some optimization was needed, known as pruning. One conventional pruning technique was cost complexity pruning, which aimed to minimize the cost complexity by recursively finding the node with the weakest link. This was regulated by a parameter known as ccp-alpha, which could be described by the formula:

$$\mathbf{CostComplexity} = \text{node's impurity} + \alpha \times (\text{number of leaves in the tree} - 1)$$

Simply put, the higher the value of α, the more aggressive the pruning was.

Our BTC-USD experiment, as depicted in the ccp-alpha error chart in Figure 2, showed that the best validation occurred at $\alpha = 0.017919$, where the error rate was the lowest. The testing dataset revealed that the optimal value was $\alpha = 0.0$, indicating that any pruning led to a decrease in accuracy. Since this was a single tree, the splits seemed to be quite essential for prediction. On the ETH-USD dataset, the best score was $\alpha = 0.002745$. Figure 8, the graphs representing the validation F1 and testing F1 scores looked remarkably similar.

3.3.4 Boosting

Boosting is a supervised machine learning ensemble technique. It differs from bagging, which created subsets of the training data by randomizing samples with replacement. Instead, boosting focused on sequentially adjusting the weights of misclassified samples to improve the performance of subsequent models. It employed n weak classifiers to assemble a strong classifier. This technique could offer a generalized approach to problem-solving, aiming to reduce both bias and variance. However, since multiple classifiers were utilized, pruning was often preferred to prevent overfitting the data. Additionally, the optimal choice for the hyperparameter n was not predetermined and was usually determined through empirical evaluation [7].

Similarly, applying the observations from the ccp-alpha error chart, we found that the best cross-validation result once again matched the best result in the testing set. Remarkably, the best validation set result, with the choice of hyperparameter $\alpha = 0.011752$, was the same as that of the test set, indicating the success of this hyperparameter tuning strategy. The ETH-USD data showed that the best validation occurred at $\alpha = 0.001826$, where the error rate was the lowest. The testing dataset revealed that the optimal value was $\alpha = 0.0$. The technique for validation was a relatively close reflection, but it wasn't always certain that the best hyperparameter for validation would be the best for the test set.

We also tested various values for n for the estimator. Once again, we observed a general pattern where, on the ETH-USD dataset, the validation performance resembled the testing performance. For the BTC-USD dataset, we achieved the best validation with $n = 100$, and for the ETH-USD dataset, the best result was obtained with $n = 50$.

3.3.5 Neural Network

Neural networks represented the latest branch of machine learning, drawing inspiration from biological neural networks. They offered a variety of architectural designs, each finding success in different applications. Mathematically speaking, they employed a technique known as backpropagation, which is a gradient optimization technique used to iteratively reduce the loss function. In this study, we utilized a successor to the recurrent neural network (RNN) architecture known as LSTM (Long Short-Term Memory) [10]. LSTM employed various gate mechanisms to mitigate the shortcomings of vanilla RNNs, including the vanishing gradient problem, which was a significant issue where gradients tended to get smaller and smaller during training, making learning extremely slow or even causing it to halt

entirely. Optimizing a deep neural network model (a network with one or more hidden layers) was not a straightforward task. Conventionally, once the network architecture was defined, various hyperparameters such as learning rate, batch size, width, and depth needed to be fine-tuned. Unfortunately, there wasn't a rigorous technique for optimizing deep learning models, as these combinations could lead to a massive search space, invoking the curse of dimensionality [8].

Epochs were hyperparameters in neural network models that dictated the number of cycles through which the data was trained. Typically, a neural network required multiple cycles to converge. However, an excessive number of epochs could often lead to overfitting. Our BTC-USD experimental plots, which depicted both binary cross-entropy loss and accuracy, offered different insights. The cross-entropy plot suggested that the model began to overfit much sooner than what was suggested by the accuracy plot. For this study, since we were also comparing with other models, we based our results on accuracy. The optimal number of epochs, as determined by the maximum accuracy on the validation set, was $epochs = 2$. Generally speaking, in both charts, we observed a significant decrease in performance after 10 epochs, despite an improvement in training performance. This indicated that the model became highly overfit beyond 10 epochs. For the ETH-USD experiment, the best validation occurred at $epochs = 21$, which was higher than that for BTC-USD. This made intuitive sense, since intraday data was much more complex and noisy, benefiting from additional epochs, while the simpler daily BTC-USD dataset could be easily overfit.

Our LSTM implementation first passed through h number of hidden units before proceeding to the fully connected layers. We empirically tested different values of hidden units, ranging from 50 to 600, in increments of 50. Both the BTC-USD and ETH-USD experiments showed that the optimal number of LSTM hidden units was 100. However, while the BTC-USD test set revealed an optimal value of 200, the ETH-USD test set also indicated that 100 was the optimal number of units. Additional LSTM hidden units seemed to reduce the performance of the LSTM models, indicating overfitting, as training accuracy continued to improve.

The most effective batch size during validation was 32 for the BTC-USD experiment and 64 for the ETH-USD experiment. Both batch sizes were quite commonly used in practice. However, the testing set revealed an optimal value of 128 for BTC-USD, while the ETH-USD test set indicated that our validation value was indeed the optimal choice.

The *mlp-depth* was a hyperparameter we devised for the model. Each increment added another layer to the fully connected layer. Although most of the fitting should have been performed at the LSTM units, it was possible that classification could benefit from having additional MLP layers. Given that our final activation function was sigmoid, a sensitive function that balanced between both binary sides, it was plausible to anticipate benefits from additional layers. However, both our experiments showed the best results with an *mlp-depth* of 1, which was the same as our initial setup with only a single layer. It was demonstrated that there was no additional benefit to adding more MLP layers after the LSTM in this case. Both experiments showed consistency between the validation and test results, allowing us to conclude that in general, there was no additional benefit to having extra MLP layers in this financial time-series data, particularly following the LSTM cells.

Lastly, we fine-tuned the learning rate based on the defined architecture. We postponed this step until the end, despite it being one of the most important hyperparameters in neural networks, because different widths and depths in the network often necessitated readjusting the learning rate. Generally, a larger network might have been more suited to a lower learning rate, although it was never straightforward. In our case, we found that a learning rate of 1×10^{-3} yielded the best performance during validation in both the BTC-USD and ETH-USD experiments. However, testing revealed that the optimal choices were actually 1×10^{-2} and 1×10^{-4}. In both cases, the results were not exact but were relatively close, deviating by one unit.

3.3.6 SVM (Support Vector Machine)

SVM (Support Vector Machines) is a supervised learning technique that projected data into a high-dimensional space. In classification tasks, it used a defined mathematical approach to find a hyperplane that separated and distinguished each class. The formula for the decision function, which helped in finding the optimal hyperplane, was given by:

$$f(x) = \langle w, x \rangle + b$$

where $f(x)$ was the decision function, x was an input vector, w was the weight vector, and b was the bias term. However, due to the nature of projecting data into high dimensionality, it could be computationally intensive when dealing with large datasets with many features. There were numerous different kernels, each offering a different way to map data to various types of spaces. These included transformations such as linear, polynomial, Gaussian, and RBF (Radial Basis Function), each with its own set of advantages and disadvantages. In SVM, the hyperparameter C was a regularization parameter often used to control the trade-off between maximizing the margin and fitting the training data

closely. In general, the larger the value of C, the stronger the fit to the training data [6].

We tested various different kernels, such as 'linear', 'poly', 'rbf', and 'sigmoid', with the results shown in Table 6 and Table 7. For the BTC-USD dataset, the best validation kernel was 'linear', as evaluated from accuracy metrics; however, testing revealed that the 'rbf' kernel was a better choice. The 'rbf' kernel was often the default choice for SVM; however, in this experiment, our validation set was not able to reveal this information before testing. Although it wasn't the worst choice, the table suggested it was still the second-best option. For the ETH-USD dataset, evaluated based on the F1 score, the 'linear' kernel demonstrated consistent performance across both the validation and test sets. Both experiments showed that the 'poly' kernel performed the worst while having the highest training score, indicating that, for this application, the 'poly' kernel was highly overfit to the training data and unable to generalize well.

Lastly, we tested different values of C, with both datasets showing the best validation results at $C = 10$. The true optimal values were revealed to be one unit away, either higher or lower, which was relatively consistent. The value of $C = 10$ was often used as the default in SVM, indicating that conventional values often aligned well, albeit with slight tuning needed.

3.4. Final Comparison

Finally, for the BTC-USD experiment, after we fine-tuned each model, we compared it to its previous initial model, as can be seen in Table 4.

Model	Test Accuracy	Test Accuracy (Initial)	Optimization Gain
Neural Network	0.547	0.517	0.030
k-Nearest Neighbors	0.523	0.490	0.033
Boosting	0.490	0.493	-0.003
SVM	0.470	0.490	-0.020
Decision Tree	0.463	0.497	-0.033

Table 4: Comparison of Optimization Gains for Different Models

Overall, the neural network (LSTM) model and k-NN significantly improved after fine-tuning. However, some models, namely SVM and Decision Tree, did not seem to improve much. Given the non-linear nature of the problem and the massive amount of noise in the financial market, these models were prone to overfitting too easily, especially the decision tree. The decision tree achieved a training accuracy of 1, which, from the model's perspective, explained everything going on from the training set. As we know, market noise is inherent in the data, and it is implausible for a model to fully interpret the data. The LSTM appeared to be more robust to the noise, generalizing across its cells. Interestingly, another model known to be simple, the k-NN algorithm, exhibited quite excellent performance after fine-tuning the number of K. This was quite surprising. It is believed that since we used technical indicators to enhance the feature description of the state of the time series at time t, k-NN was likely able to generalize some aspects from it, like momentum and RSI. In general, the neural network LSTM performed very well against other models, both in the initial setup and after fine-tuning.

Finally, for the ETH-USD experiment, after we fine-tuned each model, we compared it to its previous initial model, as can be seen in Table 5. Although the optimization technique was successful in the first experiment, it failed to improve the models in the anomaly detection application. It seems inherently difficult to detect a small amount of positive movement, even when we applied extra techniques like hybrid balance, lookback period optimization, and specific techniques for each algorithm. The success of one use case does not always apply to another.

Model	Test F1	Test F1 (Initial)	Optimization Gain
Boosting	0.225	0.235	-0.010
SVM	0.216	0.226	-0.010
Decision Tree	0.185	0.183	0.002
Neural Network	0.176	0.180	-0.003
k-Nearest Neighbors	0.146	0.175	-0.029

Table 5: Comparison of Hyperparameters Optimization Gain

If we look beyond the F1 score and interpret the results from a Bayesian perspective, where we are only concerned about the accuracy of the prediction given that it is an anomaly action, we can see from the confusion matrix that the decision tree actually performed quite poorly, predicting no anomaly actions; Figure 9. This approach essentially gives up the goal of reducing loss in favor of avoiding false positives. Interestingly, algorithms such as boosting and neural networks attempt to predict these anomaly movements with 44% to 48% accuracy under given conditions. Further exploration might turn this challenging task around.

4. Conclusion and Future Improvement

In conclusion, predicting asset movements such as Bitcoin or Ethereum is a challenging task. In the first case, BTC-USD, where we attempted to predict upcoming trends, we achieved accuracy results slightly better than random

chance (0.5); although the difference is only a few percentage points, the results can still have a significant impact in the long run, akin to betting on a consistently weighted coin. This phenomenon is a manifestation of the law of large numbers. Moreover, in modern markets where leverage options like futures and option trading are available, wins and losses can amplify significantly. In terms of supervised algorithms, the neural network once again proved to achieve top-ranking results, unmatched by traditional machine learning algorithms. However, this comes at the cost of considerably increased computation time, creating a trade-off between computational efficiency and optimal prediction results. As the complexity of the deep learning model increases, with additional layers and width, the interpretability of the model decreases, which can be a concern for risk management in financial decisions.

In this study, we explored various machine learning setups and handcrafted feature engineering for supervised learning, successfully showcasing the value of traditional quantitative technical indicators. We demonstrated methods to optimize various types of models and highlighted the superior performance of deep learning. In future works, we aim to explore alternative feature engineering methods for supervised learning, interpret feature importance, and possibly also weigh features in time series with decay hyperparameters. We plan to introduce more classic technical indicators and evaluate their effectiveness, include sentiment analysis, and incorporate fundamental models like market size and market caps. The potential architecture space in machine learning for finance is essentially limitless.

Lastly, we must acknowledge that predicting a small number of positive labels in the dataset is extremely challenging. While the detection seems logical, essentially all the algorithms we used for this study did not show promising results. Selecting the right loss metric is another significant factor for success. We understand that accuracy is not the most important metric; therefore, we chose the F1 score to balance both sensitivity and accuracy. However, upon further analysis of the algorithms' performance, it appears that they are primarily trying to predict the majority class to reduce loss, which was not the intention of our study. Generally speaking, there are still many undesirable false detections, even from this perspective. Therefore, we conclude that this detection algorithm was not as successful as our first experiment. In machine learning, choosing the right project can sometimes be as important as the execution itself. Some projects can be too ambitious to perform successfully. However, if we continue to tackle outlier detection algorithms, we will need to define our own custom loss function to reduce ambiguity in our objectives. Given that most classic supervised learning algorithms have their associated loss functions, customizing the boosting algorithm or a neural network-based model can be a good choice for implementation, with custom architecture and loss function design.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, and Andy Davis. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. 2016. 1

[2] Jason Brownlee. A gentle introduction to k-fold cross-validation, 2018. 2

[3] Vitalik Buterin. Ethereum white paper, 2013. 1

[4] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: Synthetic minority oversampling technique. In *Journal of Artificial Intelligence Research*, 2002. 2

[5] François Chollet. Keras, 2015. 1

[6] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 1995. 8

[7] Yoav Freund and Robert E. Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. 1995. 6

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning, 2016. 7

[9] Haibo He and Edwardo A. Garcia. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*. 2

[10] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 1997. 6

[11] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *www.bitcoin.org*, 2008. 1

[12] Karl Pearson. Notes on regression and inheritance in the case of two parents. *Proceedings of the Royal Society of London*, 1895. 2

Appendix

Kernel	Train Accuracy	Validation Accuracy	Test Accuracy
linear	0.878	0.514	0.497
poly	0.854	0.472	0.490
rbf	0.788	0.444	0.507
sigmoid	0.531	0.472	0.497

Table 6: Kernel Accuracy Comparison

Kernel	Train F1	Validation F1	Test F1
linear	0.665	0.326	0.277
poly	0.823	0.291	0.018
rbf	0.796	0.302	0.230
sigmoid	0.550	0.256	0.036

Table 7: Comparison of F1 Scores for Different Kernels

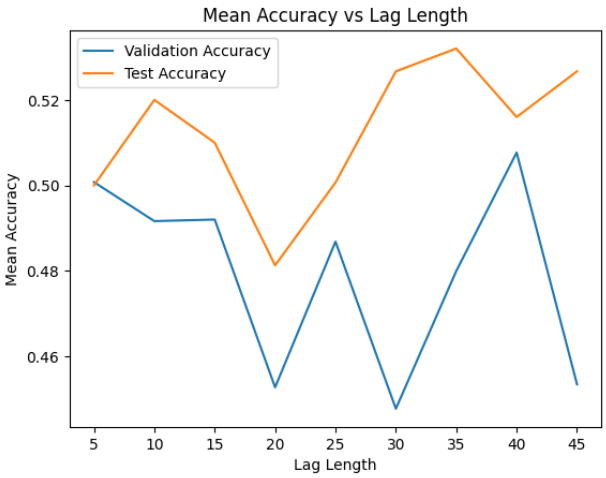

Figure 1: Optimizing Lookback Periods

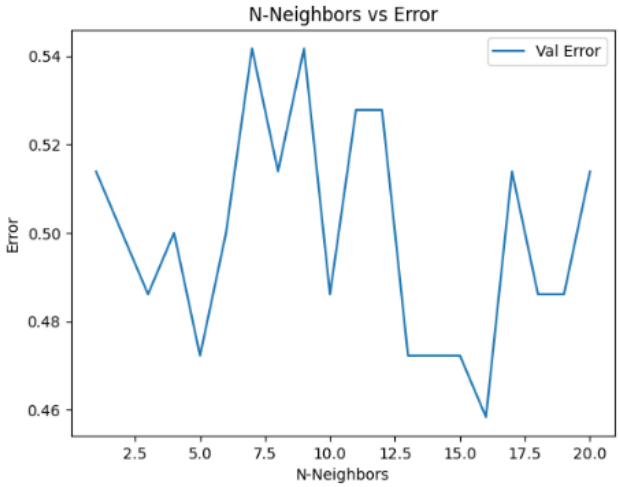

Figure 3: The k-NN error chart.

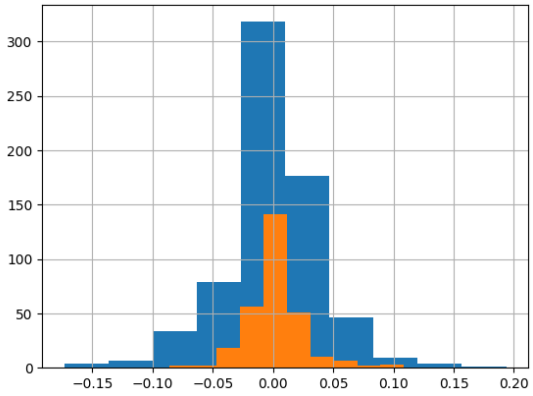

Figure 4: Simple train-test splits distribution.

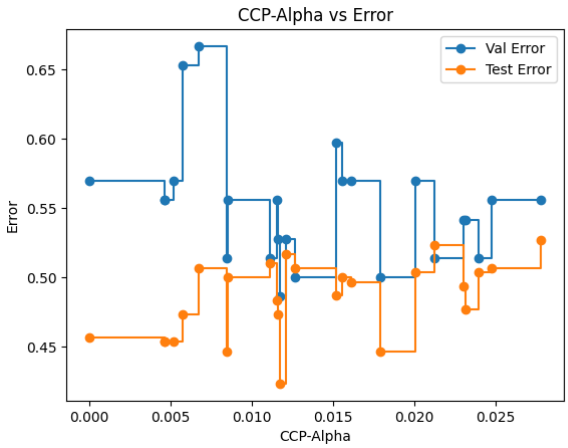

Figure 2: The ccp-alpha error chart.

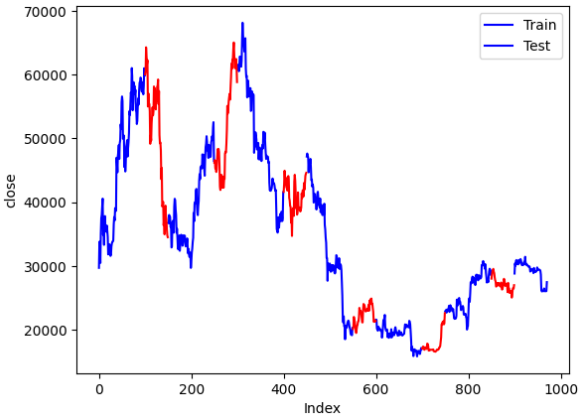

Figure 5: K-fold cross validation.

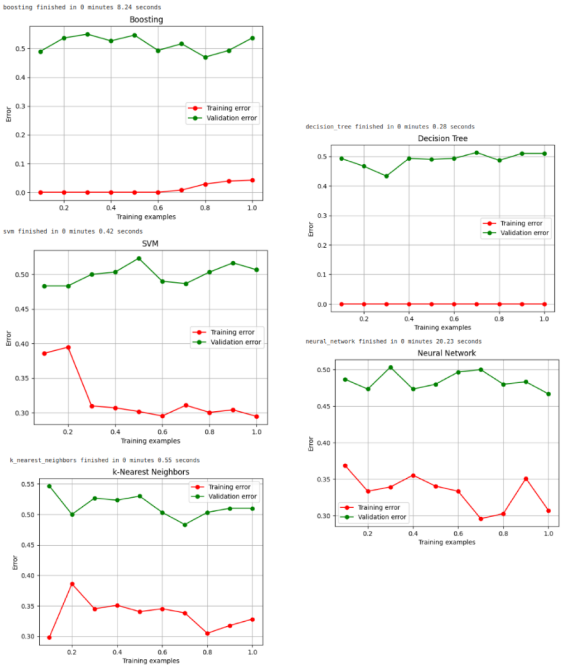

Figure 6: Learning curve of various algorithms.

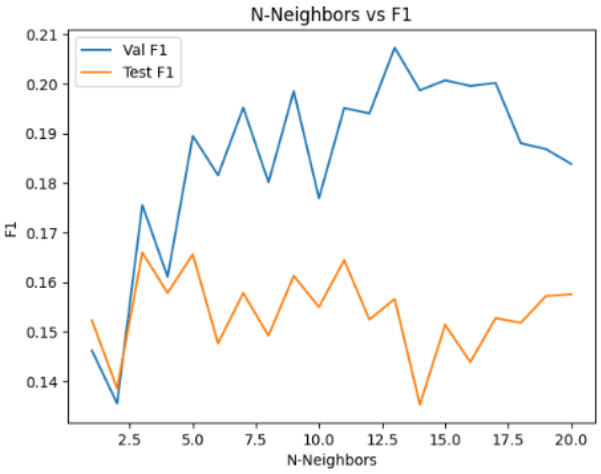

Figure 7: The k-NN validation F1 vs test F1.

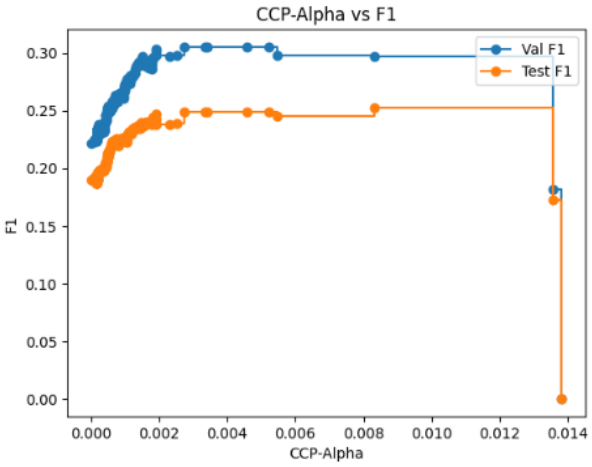

Figure 8: The ccp-alpha validation F1 vs test F1.

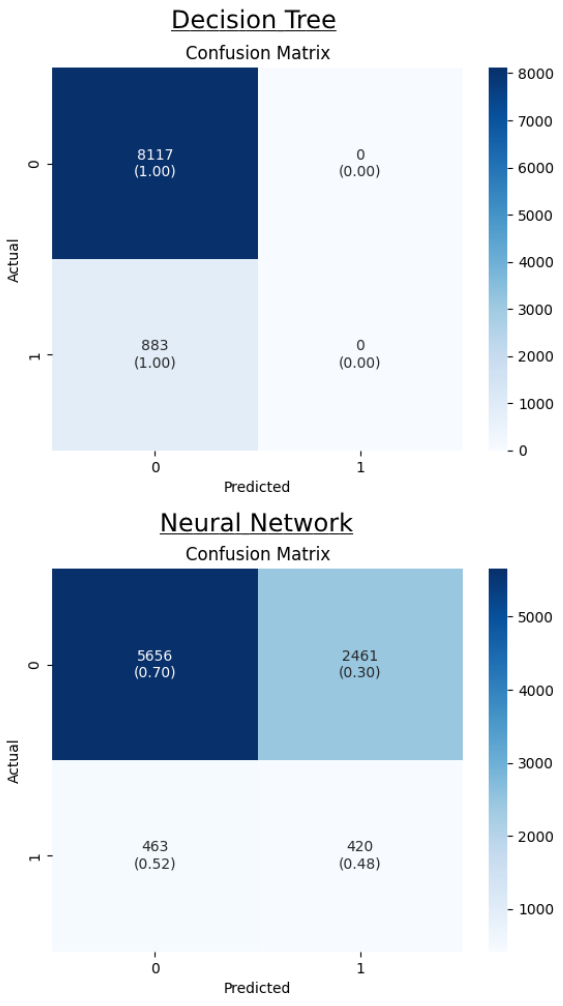

Figure 9: Confusion Matrix.