Short-term prediction of Crypto-currencies using Machine Learning

Abhishek Kumar

Worldquant University

Abstract

The purpose of this paper is to apply machine learning techniques to predict the movement of cryptocurrencies on an intraday scale and to develop a trading strategy based on the model. A variety of machine learning algorithms like AdaBoost, RandomForest, XGBoost and Neural Networks has been used and their suitability is judged for the task. This work tries to use different labels and unique features including prediction from forecasts of econometric models like GARCH, volume and trade data and their interaction with returns. Finally, a trading strategy based on the model is proposed and back-tested on unseen data.

Keywords: *Machine Learning, cryptocurrencies, RandomForest, AdaBoost, intraday trading.*

Introduction

Machine Learning has found its application on nearly every aspect of our lives. Knowingly or unknowingly, we interact with such algorithms multiple times a day from browsing internet, navigating in our cars or buying groceries at supermarket. Pertaining to its success in solving complex issues of image recognition, search optimization and mastering games like chess and GO, it's a natural inquisition to apply it for predicting asset prices.

Multiple notable work has been done academically in this regard largely in stocks, currencies and commodities. There has also been evidence of it being used by large hedge funds and institutional banks for profitable trading. This work will apply the same principles and improve upon their work in the nascent field of cryptocurrencies. Specifically, the attempt is to bridge the gap between theoretical models and practical constraints and couple them to realize an industrial trading strategy. This has to be accomplished by not only applying state of the art machine learning algorithms but also improve the general process of model selection in terms of using different kind of labels to help the algorithm learn better, using relevant features for training and making customized features based on intuition, knowledge of data generating process and even borrowing features from work done on other asset classes.

An attempt to overcome challenges faced by other researchers in the same field has also been made, some of which include non- interpretability of the final model, poor translation of high accuracy of model to realized profits (at- least in backtest), and realistic assumptions about live market conditions.

Literature Review

The current literature on Application of Machine learning models to crypto-currency can be described as "scattered" at best. There has been attempt by numerous individual authors and papers to model price formation in crypto markets by machine learning models. Some of the earlier works concentrated entirely on bitcoin prediction. Most of the papers has been written by students and hence as such not many key people exist due to this being a nascent topic.

The first work on this direction was by Shah and Zhang (2014) who used a Bayesian regression model to predict the direction of Bitcoin. They used only past price data sampled at different frequencies and order imbalance data. They reported yearly returns of 85% and Sharpe ratio of 4.10 from their model forecasts. However, these numbers where before accounting for any transaction costs or slippage.

Another one of the earliest works was in the *paper* **Automated Bitcoin Trading via Machine Learning Algorithms** by Isaac Madan, Shaurya Saluja and Aojia Zhao (2014) of Department of Computer Science at Stanford University in 2014.

They used three different horizons: 1-day, 10-minutes and 10-seconds and attempted to explain the price change in bitcoin using a custom GLM and Random-Forest Model with good accuracy.

They used a unique set of features for daily horizon which was a combination of bitcoin price and bitcoin networks data. The bitcoin networks data included 16 different features like confirmation time, Block Size and hash rate achieving an extremely high accuracy of 94% from Random-Forest Model. The models based on 10-minute forecasts also had a decent accuracy of 57% for the random forest model and 54% for the GLM model.

The paper however did not delve into back-testing a strategy based on the model and thus was incomplete. Another problem was a little technical – the results may very well be due to the rising long-term prices in the market during the phase—a naive buy-and-hold strategy achieved a similar result.

In another paper titled **Statistical Arbitrage in Cryptocurrency Markets by** Thomas Günter Fischer, Christopher Krauss and Alexander Deinert (2017) they developed a stat arb strategy similar to market neutral strategies ran by long short equity funds.

They used past prices over different horizons to estimate a ML model based on RandomForest and a simple logistic regression model. Instead of predicting the direction they trained the algorithm to learn the probability of outperforming the median return of the cross – section. They then evaluated the performance of a trading strategy which goes long on the top 3 coins having highest probability of outperformance and short 3 coins with lowest probability of outperformance. Their results had realistic transaction cost and bid ask slippage which results indicated a return of 25% per annum after accounting for them.

Other notable works include modelling volume data of bitcoin to predict future returns by Balcilar, Bouri, Gupta and Roubaud (2017) and Combining Technical Indicators to generate profitable signals by Sungjoo, and Moon (2018).

The general consensus is that on multiple horizons ranging from 1 minute to 1 day it is possible to predict the direction of the cryptocurrency markets with better than random chance. However, the transformation of accuracy to viable trading strategy has seen mixed results after accounting for transaction costs and liquidity levels.

There has been considerable creativity demonstrated in selecting features for the model including, feature pertinent to bitcoin networks data (Isaac Madan, Shaurya Saluja and Aojia Zhao), twitter sentiments data (Colianni, Rosales, and Signorotti), Web -forum sentiments data (Kim, Y. Bin, Jun G. Kim, Wook Kim, Jae H. Im, Tae H. Kim, Shin J. Kang, and Chang H. Kim) and google trends data (Garcia and Schweitzer),

However, in this paper we employ econometric models for price and volatility and simple technical indicators for trend detection. These classical statistical methods standalone work better as features and have been successfully incorporated in machine learning models as features for asset classes such as cash equities and currencies.

Another major issue is the number of trades per day by the Strategy: An empirical analysis of major crypto currency exchanges Binance, CoinBase and Huobi reveals that the lowest transaction cost ranges from 10 bps to 20 bps per trade. So, even if a strategy trades only 5 times a day it would result in a transaction costs of 365% per year. To achieve a return greater than 365% consistently is impossible to say the least.

Hence for an Intraday strategy, we have to achieve a balance in horizon selection, trading frequency and accuracy which means we have to sit idle at most of the times with or without a position in the underlying. This fact renders the binary classification task (+1 long, -1 short) defunct which has to be addressed by a three-state classification (+1 long, -1 short ,0 hold). This has not been done by any paper upon review of freely available papers on the subject in multiple databases.

Methodology

The project has been divided into three separate tasks. The first is modelling, second is Strategy development and third conclusion. The modelling component includes applying a machine learning algorithm which translates to collection of data, generating labels, selecting and designing features, training a ML model on the features to predict labels, cross-validate the model and judge the model performance on unseen test set.

The second task includes using the model to design a trading strategy and judge the performance of the model on metrics like (Sharpe Ratio, Annualized returns and) using practical constraints of transaction costs and slippages.

The final task is to learn and report from the inferences of model and provide conclusive remarks.

a. Modelling

The main components of this task include:

- **Collection of Data**: The data has been downloaded from Binance exchange of the top 5 market cap cryptocurrencies (BTC, ETH, XRP, LTC, BCH) with base currency as tether. The data contains field representing (OPEN, HIGH, LOW, CLOSE, VOLUME, TRADES, UNIX TIME) field at 30 minutes interval and 15 minutes interval. The R script to download the data has been provided in the GitHub link. This data ranges from 1st January 2019 till 20th November 2020 for all the currencies except BCHUSDT which starts from 28th November 2019 03:30 till 20th November 2020. The train validation test split ratios were 60%, 15% and 25% respectively.

- **Generating Labels**: The problem has been modelled as classification problem so the labels would be the direction of next period ((future) return (+1 or -1). However due to low volatility of assets in intraday framework a threshold of 30 bps has been applied to make the labels as (+1, -1, 0). The results for binary classification are presented for BTCUSDT in Appendix 2. Sample results for 15 minutes data has also been provided for BTCUSDT in Appendix 3.

- **Selection and Designing Features**: The feature space has been segregated into 6 categories as follows:

  Lag 1 percent change: The percent change in the assets open, high, low, close, volume and trades from previous values.

  Technical Indicators: A total of 14 different technical indicators are used, some with multiple parameters. The implementation of technical indicators was from "ta" library in python. Appendix 1a provides a complete list of Technical indicators used.

  Price Derivations: Statistical transforms of Price and volume data is done and used as features**.** These include mean returns, order flow, skewness, kurtosis and standard deviation. Appendix 1b provides a complete list of Price derived features used.

  Previous Returns: This includes returns of the asset at various time horizons including lag-2, lag-2, previous 5 period returns, previous 10 period returns and previous 20 period returns. Appendix 1c provides complete list of previous return features used.

  Econometric Features: This includes volatility models like GARCH and bipower variation based on rolling window with fixed parameters, and difference between various volatility models to capture jump components. Appendix 1d provides complete list of econometric features used.

  Seasonal Factors: This category tries to capture intraday seasonality patterns. This includes an indicator for given time interval, close returns 24 hrs ago, close returns 12 hrs ago, average of last 5 days volume and trades during same time interval, % change between current volume(trades) and average volume(trades) mentioned above and product of return and volume (trades) should the % change exceeds a threshold of (50, 100, 200) to capture large abnormal volume with direction. Appendix 1e provides complete list of seasonal factors used.

This resulted in total number of features as 93.

Although Decision trees can handle non normalized data we use two standard methods for scaling the data. The first uses min max scaler and the other uses standard normalization. The results were found to be similar for both approaches hence only report for standard scaler are displayed.

- **Model Training**: Random Forests, AdaBoost, XGBoost and Neural Networks will be used for training separately.

  The description is as follows:

  AdaBoost Classifier: AdaBoost classifier with Decision tree classifier as base learners are used.

  The hyperparameters to be optimized for the base learner were *max_depth* of the decision tree, *min_sample_split* and *min_sample_leaf. Class weight "balanced"* was used due to large proportion of class 0 as compared to -1, 1.

  To limit overfitting *max_depth* has been varied as (1,2) and *min_sample_leaf* and *min_sample_split* has not been optimized.

  The main algorithm has *learning_rate* and *number of estimators* as hyperparameters. The *learning_rate* was varied between 0.1 – 0.9 in increments of 0.2. The *number of estimators* was set to 50.
  Only the best results are reported.

  The best results for BTCUSDT were obtained at max_depth = 2 and learning_rate = 0.5 with train accuracy of 0.61 and validation accuracy of 0.55. The results for test set are provided at the results section.

  The most important features were 40 period standard deviation, time indicator, volume, volume % change, 40 period kurtosis and stochastic oscillator.

  The annualized returns of the strategy were 300% with sharpe of 5.11 for training and -62% and sharpe of -0.89 for validation set after transaction costs of 10 bps.

  RandomForest Classifier: RandomForest classifier with Decision tree classifier as base learners are used.

  The hyperparameters to be optimized were *max_depth* of the decision tree, *min_sample_split, and min_sample_leaf. Class weight "balanced"* was used due to imbalanced dataset.
  To limit overfitting *max_depth* has been varied as (1,2) and *min_sample_leaf* and *min_sample_split* has not been optimized.

  The other parameters are *max_features* to be used while deciding the next learner and *number of estimators* as hyperparameters. The max_features were set to 93. The number of estimators was set to 50.
  Only the best results are reported.

  The best results for BTCUSDT were obtained at max_depth = 2 and with train accuracy of 0.56 and validation accuracy of 0.34. The results for test set are provided at the results section.

The most important features were 40 period standard deviation, 20 period standard deviation, 20 period bipower variation, trades, 5 period bipower variation.

The annualized returns of the strategy were 150% with sharpe of 0.88 for training and -128% and sharpe of -1.18 for validation set after transaction costs of 10 bps.

XGBoost Classifier: XGBoost classifier with gbtree as base learners is used.

The hyperparameters to be optimized for the base learner was *max_depth*, *reg_lambda*, *reg_alpha*.
To limit overfitting max_depth has been varied as (1,2) and lambda and alpha has not been optimized.

The main algorithm has *learning_rate* and *number of estimators* as hyperparameters. The *learning_rate* was varied between 0.1 – 0.9 in increments of 0.2. The *number of estimators* was set to 50.
Only the best results are reported.

The best results for BTCUSDT were obtained at max_depth = 2 and learning_rate = 0.3 with train accuracy of 0.73 and validation accuracy of 0.59. The results for test set are provided at the results section.

The most important features were 10 period standard deviation, 5 period bipower variation, 20 period bipower variation, trades, difference between 10 period GARCH and bipower variation and difference between 20 period bipower variation and standard deviation.

The annualized returns of the strategy were 249% with sharpe of 6.74 for training and 257% and sharpe of 3.00 for validation set after transaction costs of 10 bps.

NeuralNetwork Classifier: Multi-Layer Perceptron classifier was used.

The hyperparameters to be optimized are the number and *hidden layer sizes*, *activation function* and *max iterations*.

It was observed that *relu* performed better than other activation function all of the times. The hidden layers tried were (5,5), (5,5,5),
(10,10) and (10,10,10). Not much value addition was observed trying number of hidden layers greater than 3.

The *max iterations* were set to 1000.

The best results for BTCUSDT were obtained at hidden layer size of (10,10) with train accuracy of 0.73 and validation accuracy of 0.59. The results for test set are provided at the results section.

The annualized returns of the strategy were 190% with sharpe of 4.71 for training and 284% and sharpe of 3.66 for validation set after transaction costs of 10 bps per trade.

Based on training and validation set xgboost classifier had the best results in terms of accuracy and performance the results of that were also stable for varied values of hyperparameters.

Results:

TESTING SET PERFORMANCE

ADABOOST

	BTCUSDT	ETHUSDT	XRPUSDT	LTCUSDT	BCHUSDT
ACCURACY	0.75	0.46	0.31	0.49	0.3
RETURNS	-61%	-198%	-124%	-141%	-125%
SHARPE RATIO	-4.2	-3.15	-1.82	-2.46	-2.29

RANDOM FOREST

	BTCUSDT	ETHUSDT	XRPUSDT	LTCUSDT	BCHUSDT
ACCURACY	0.61	0.49	0.52	0.54	0.46
RETURNS	51%	81%	54.30%	18%	427%
SHARPE RATIO	1.35	1.26	0.86	0.28	5.25

XGBOOST

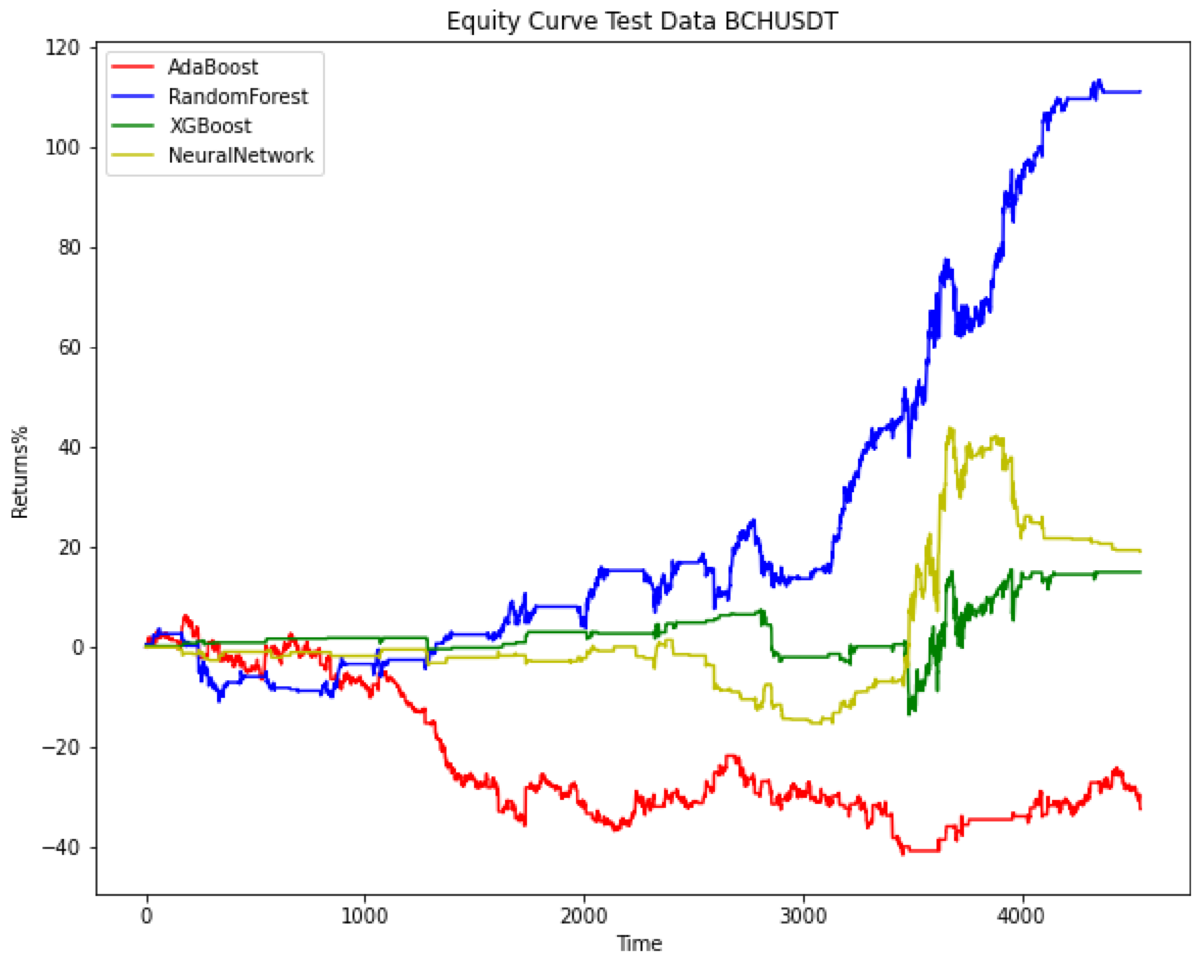
	BTCUSDT	ETHUSDT	XRPUSDT	LTCUSDT	BCHUSDT
ACCURACY	0.78	0.61	0.63	0.58	0.55
RETURNS	12.45%	42%	67%	73.86%	57%
SHARPE RATIO	1.47	0.91	1.56	2.28	1.01

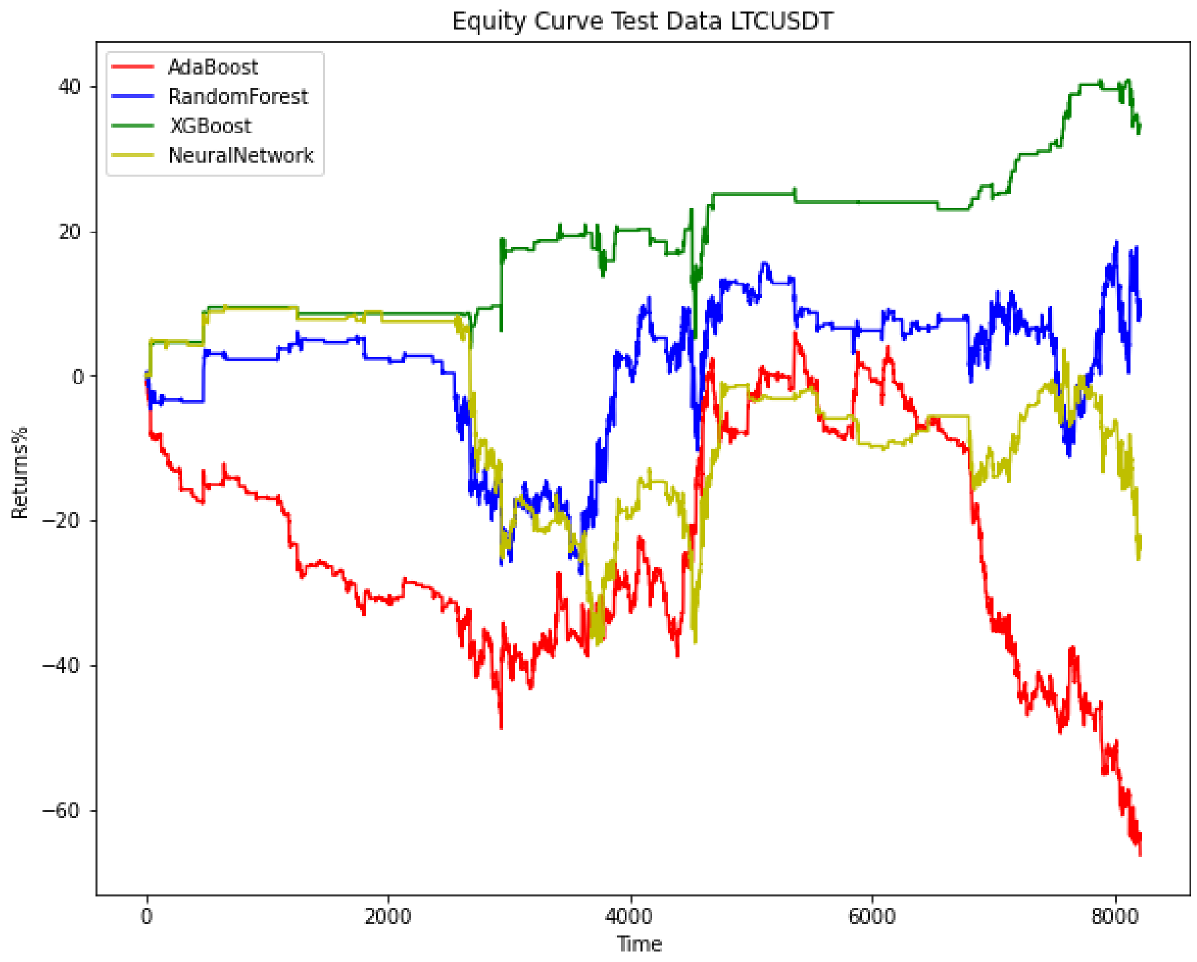
MLP CLASSIFIER

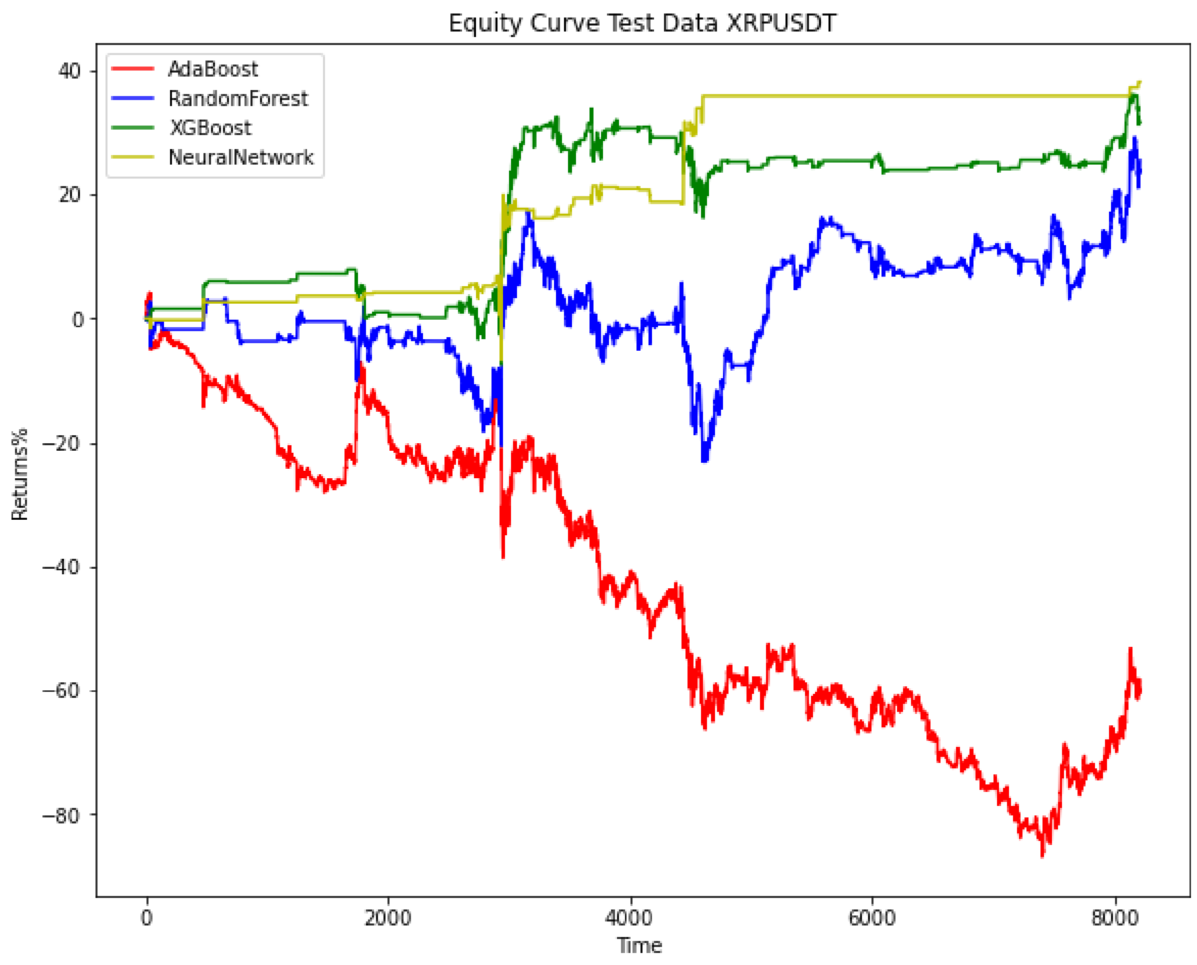
	BTCUSDT	ETHUSDT	XRPUSDT	LTCUSDT	BCHUSDT
ACCURACY	0.77	0.55	0.62	0.57	0.54
RETURNS	-17%	-114%	-30%	-124%	106.51%
SHARPE RATIO	-1.13	-1.97	-0.87	-2.43	1.67

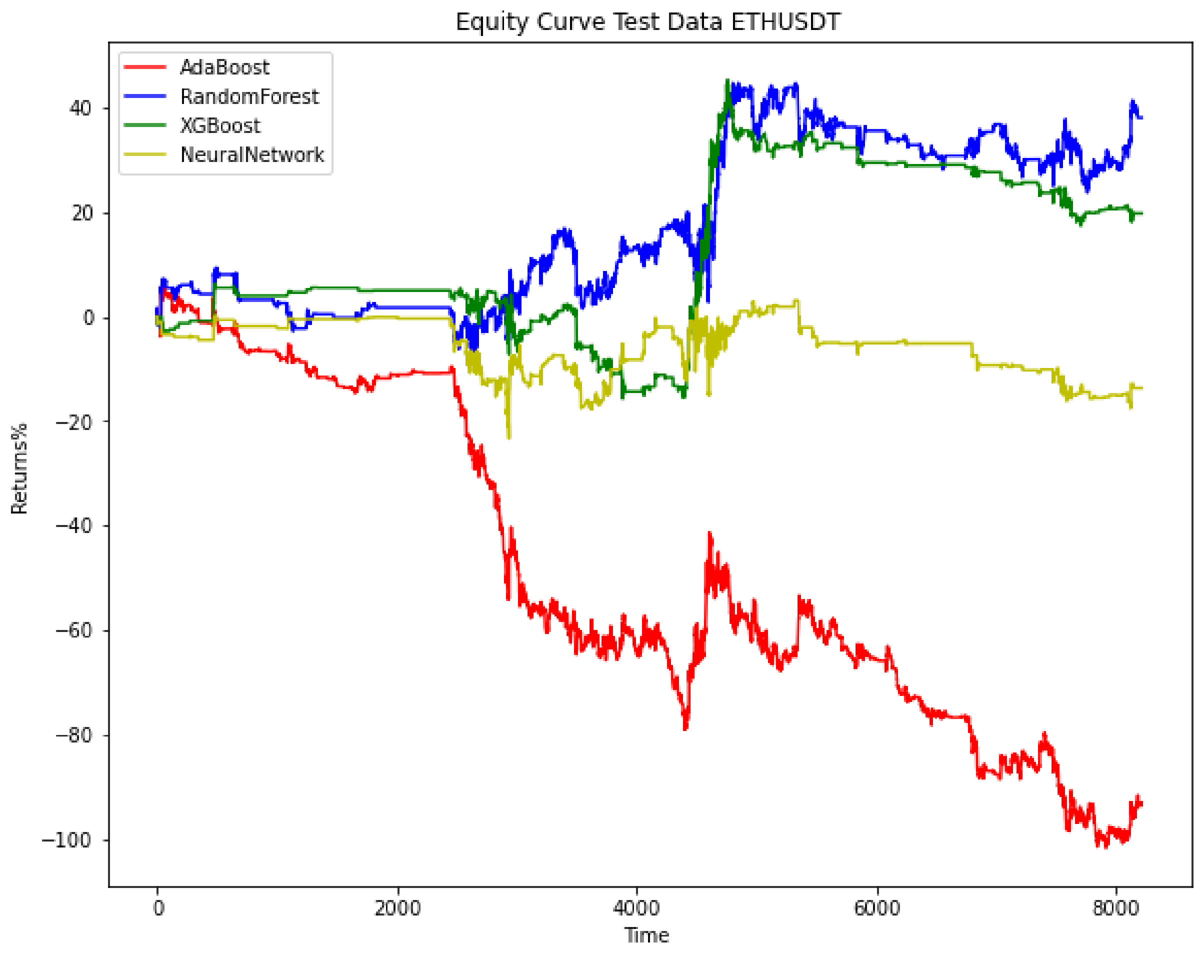
As we can see the out of sample performance of MLP classifier and AdaBoost classifier are very poor when compared to their training performance. This may be due to overfitting.

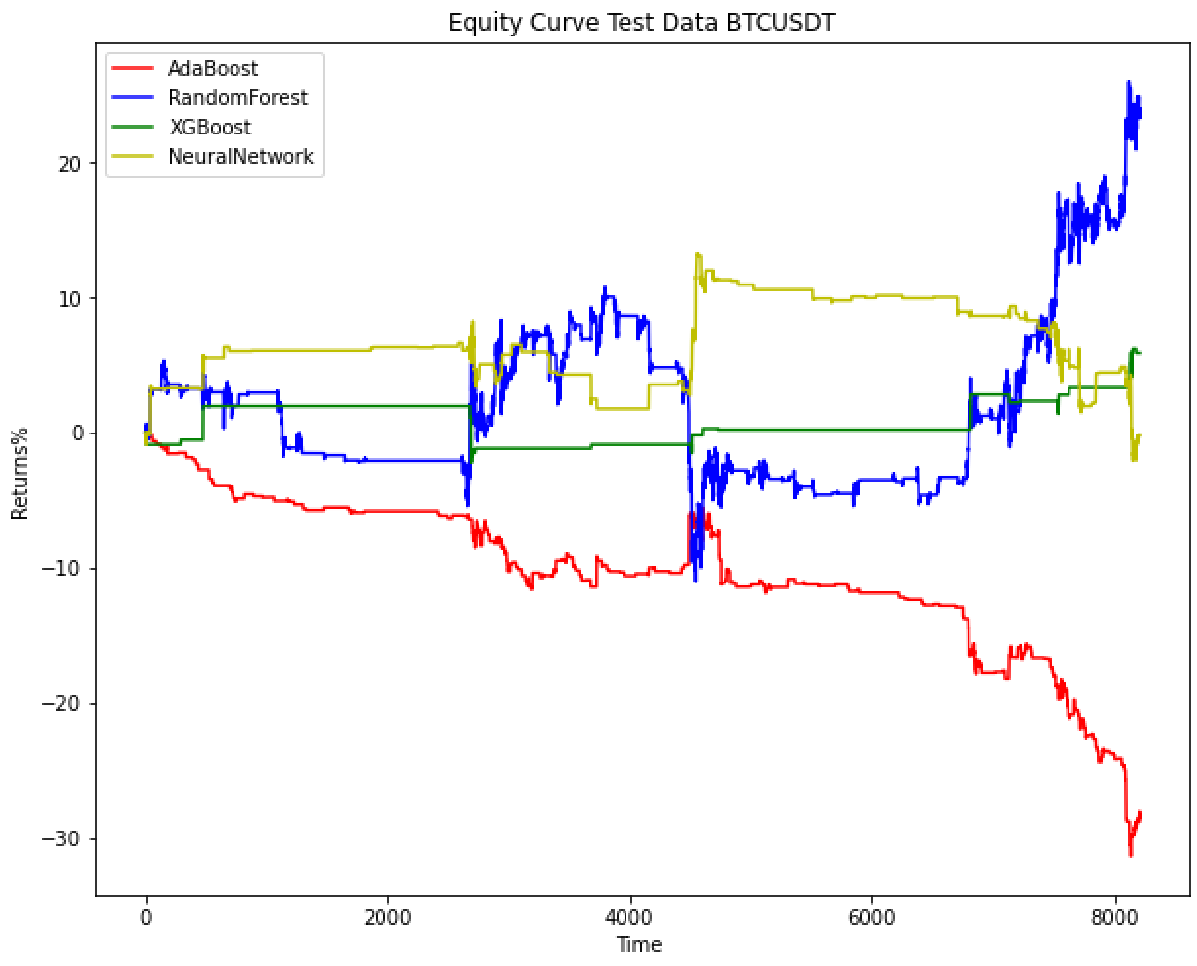
Random forests had worst performance in training set but has done surprisingly well in out of sample. XGBoost was best performing in training as well as testing set data. It beats every other classifier in terms of accuracy for all the assets. It also generates positive returns for all the currencies along with RandomForest.


Equity Curve Test Data BCHUSDT
AdaBoost
RandomForest
XGBoost
NeuralNetwork
Returns%
Time
120
100
80
60
40
20
0
−20
−40
0
1000
2000
3000
4000


Equity Curve Test Data LTCUSDT
AdaBoost
RandomForest
XGBoost
NeuralNetwork
Returns%
Time
40
20
0
−20
−40
−60
0
2000
4000
6000
8000

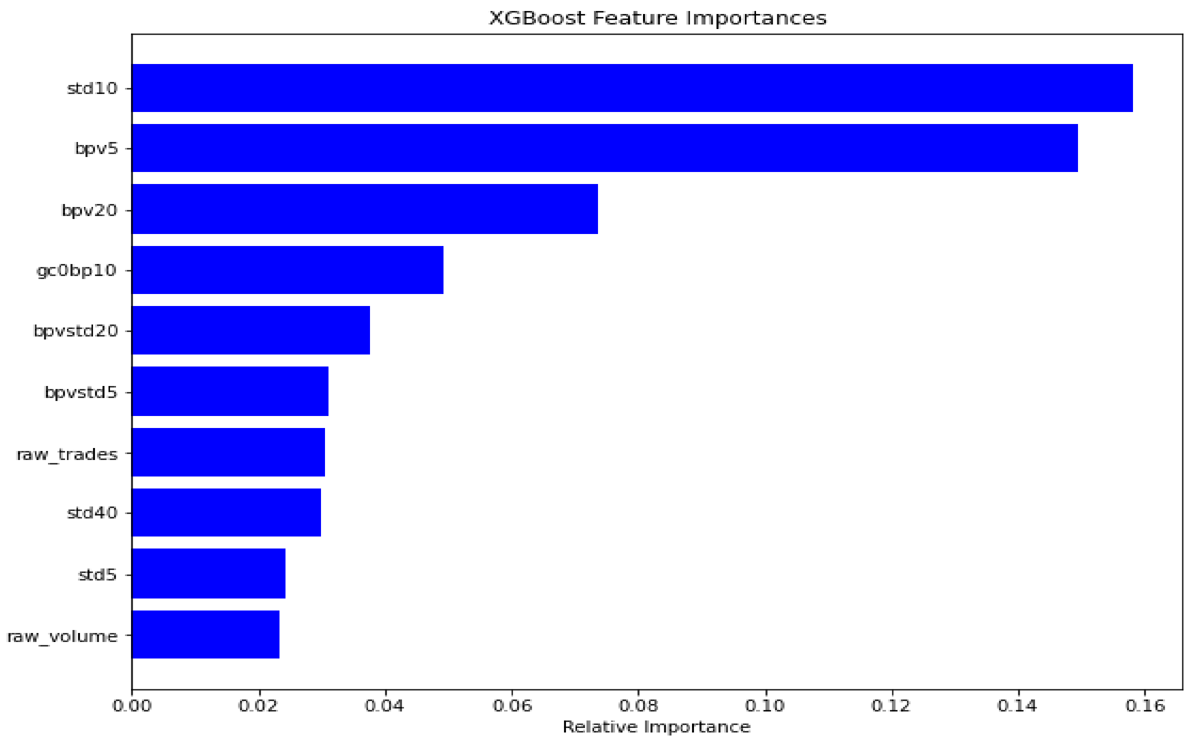

Equity Curve Test Data XRPUSDT
AdaBoost
RandomForest
XGBoost
NeuralNetwork
40
20
0
−20
−40
−60
−80
Returns%
0
2000
4000
6000
8000
Time


Equity Curve Test Data ETHUSDT
AdaBoost
RandomForest
XGBoost
NeuralNetwork
40
20
0
−20
−40
−60
−80
−100
Returns%
0
2000
4000
6000
8000
Time

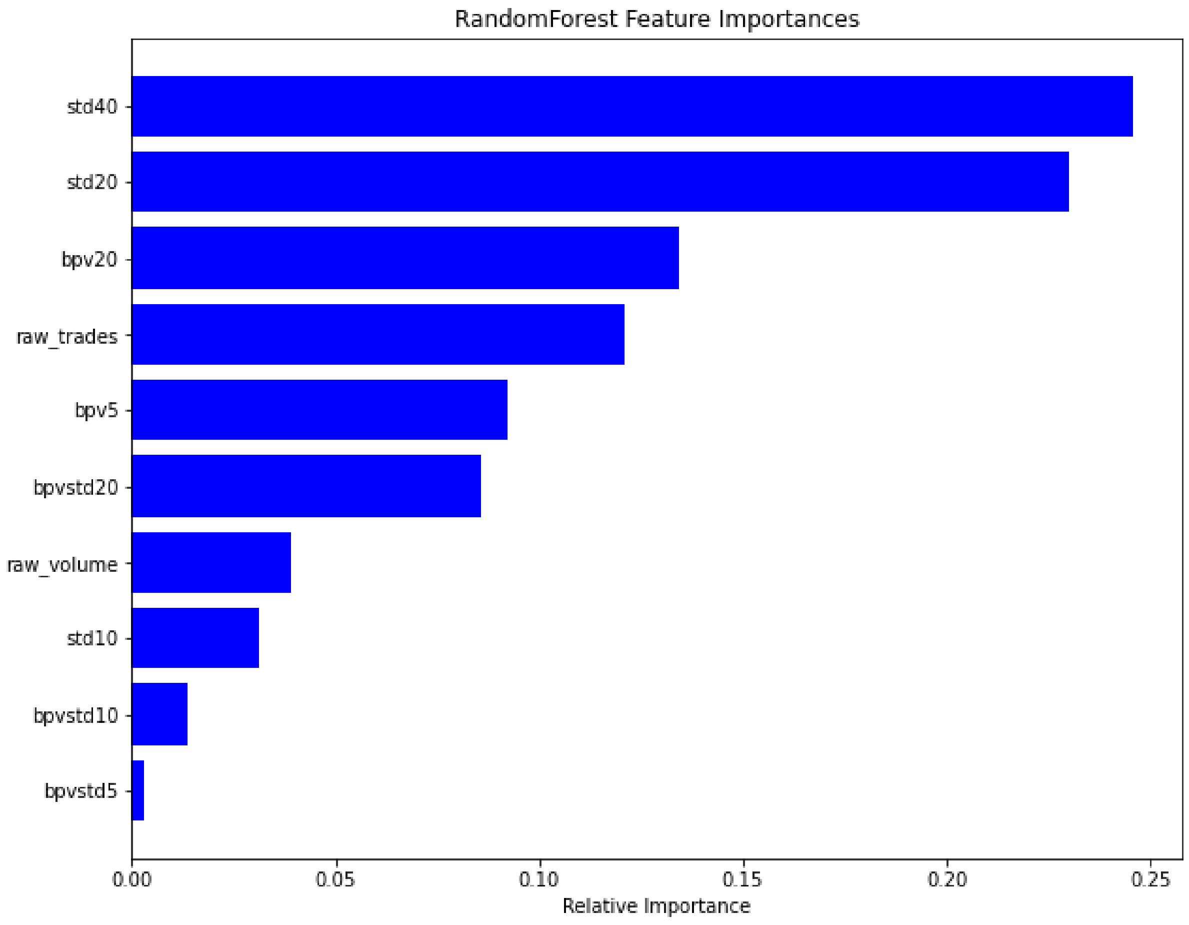

Equity Curve Test Data BTCUSDT
AdaBoost
RandomForest
XGBoost
NeuralNetwork
Returns%
Time
20
10
0
−10
−20
−30
0
2000
4000
6000
8000

FEATURE IMPORTANCE

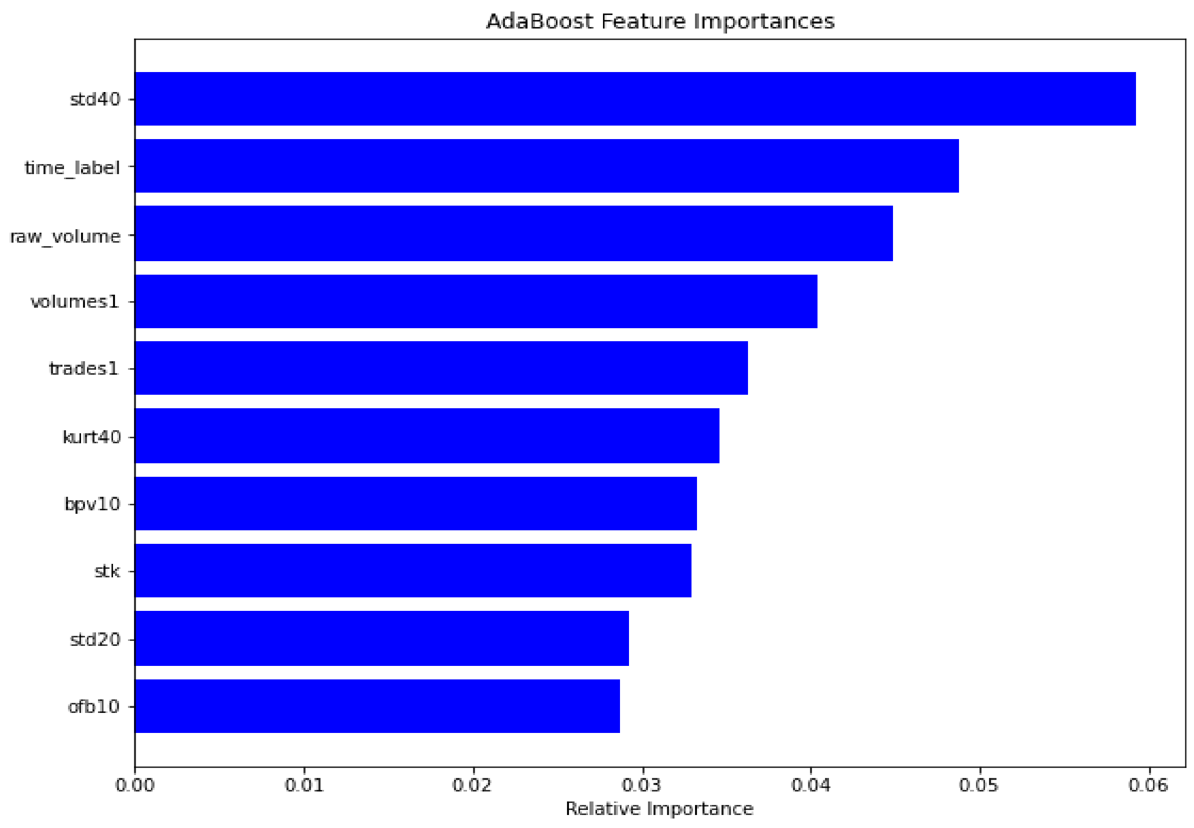
A common theme among the various models for various assets was use of volatility-based indicators as important features



TOP 10 XGBOOST FEATURE IMPORTANCE
(REFER APPENDIX FOR VARIABLE DEFINITION)



TOP 10 RANDOMFOREST FEATURE IMPORTANCE
(REFER APPENDIX FOR VARIABLE DEFINITION)



TOP 10 ADABOOST FEATURE IMPORTANCE
(REFER APPENDIX FOR VARIABLE DEFINITION)

All the three tree-based models used standard deviation and bipower variation as the top features. This does makes sense intuitively as higher chances of observing a move greater than 30 bps is only possible in high volatility environment.

CONCLUSIVE REMARKS

This project demonstrated the successful application of machine learning algorithms for trading. The problem was modelled as a classification problem and various features were used for the prediction task. Some hyperparameter tuning was done for the best results. It also provided evidence that successful intraday trading can be done even after accounting for transaction costs based on machine learning models.

Future work in this area will involve applying other algorithms like LSTM based neural networks, SVM based boosters among others. A proper treatment of transaction costs and slippages based on order book data is also desired for more confidence in trading results.

References

11.1. Journal Article

[1] Shah and Zhang (2014) ***Bayesian regression and Bitcoin.*** Devavrat Shah, Kang Zhang.

[2] Madan, Saluja and Zhao (2015) ***Automated Bitcoin Trading via Machine Learning Algorithms.*** Issac Madan, Shaurya Saluja and Aojia Zhao

[3] Garcia and Schweitzer (2015) ***Social signals and algorithmic trading of Bitcoin.***

[4] Balcilar, Bouri, Gupta and Roubaud (2017) ***Can Volume Predict Bitcoin Returns and Volatility? A Quantiles-Based Approach*** Mehmet Balcilar, Elie Bouri, Rangan Gupta and David Roubaud

[5] Koutmos (2018) ***Bitcoin returns and transaction activity*** Dimitri Koutmos

[6] ***Predicting fluctuations in cryptocurrency transactions based on user comments and replies***. Kim, Y. Bin, Jun G. Kim, Wook Kim, Jae H. Im, Tae H. Kim, Shin J. Kang, and Chang H. Kim (2016)

[7] Sungjoo, and Moon (2018). ***Finding attractive technical patterns in cryptocurrency markets.*** Ha, Sungjoo, and Byung-Ro Moon.

[8] Colianni, Rosales, and Signorotti (2015). ***Algorithmic Trading of Cryptocurrency Based on Twitter Sentiment Analysis*** Stuart Colianni, Stephanie Rosales, and Michael Signorotti

[9] Keywan Christian Rasekhschaffe & Robert C. Jones (2019): ***Machine Learning for Stock Selection, Financial Analysts Journa***l,
DOI: 10.1080/0015198X.2019.1596678

APPENDIX 1

This contains features used for the training of neural network. Python code-based notation has been left for ease of understanding. The variables in the left side in square brackets inside apostrophes are the features. The right-hand side contains definition/ formulas for the same.

APPENDIX 1a

LIST OF TECHNICAL INDICATORS USED AS FEATURES
(MARKED INSIDE SQUARE BRACKETS)

```
['sma5']    = ["close"]/sma_indicator(close=["close"], window=5, fillna = False) -1
['sma10']  = ["close"]/sma_indicator(close=["close"], window=10, fillna = False) -1
['sma20']  = ["close"]/sma_indicator(close=["close"], window=20, fillna = False) -1

['ema5']    = ["close"]/ema_indicator(close=["close"], window=5, fillna = False) -1
['ema10']  = ["close"]/ema_indicator(close=["close"], window=10, fillna = False) -1
['ema20']  = ["close"]/ema_indicator(close=["close"], window=20, fillna = False) -1

['rsi14']    = (["close"]/rsi(close=["close"], window=14, fillna = False) -50)/100
['rsi28']    = (["close"]/rsi(close=["close"], window=28, fillna = False) -50)/100

['aro10']    = AroonIndicator(close=["close"], window=10, fillna = False).aroon_indicator()
['aro20']    = AroonIndicator(close=["close"], window=20, fillna = False).aroon_indicator()

['bb_bbm'] = ["close"]/indicator_bb.bollinger_mavg()  -1
['bb_bbh']  = ["close"]/indicator_bb.bollinger_hband() -1
['bb_bbl']   = ["close"]/indicator_bb.bollinger_lband() -1

['cci10']     = CCIIndicator(high = ["high"],low = ["low"],close = ["close"] , window= 10 , fillna = False).cci()
['cci20']     = CCIIndicator(high = ["high"],low = ["low"],close = ["close"] , window= 20 , fillna = False).cci()

['chk10']    = ChaikinMoneyFlowIndicator(high = ["high"],low = ["low"],close = ["close"],volume =
                 ["volume"] , window= 10 , fillna = False).chaikin_money_flow()

['chk20']    = ChaikinMoneyFlowIndicator(high = ["high"],low = ["low"],close = ["close"],volume =
                 ["volume"] , window= 20 , fillna = False).chaikin_money_flow()

['macd']     = MACD(["close"],fillna = False).macd()

['stk']        = StochasticOscillator(high = ["high"],low = ["low"],close = ["close"],window= 10 , fillna
                  =False).stoch()

['adi']        = AccDistIndexIndicator(high = ["high"],low = ["low"],close = ["close"],volume = ["volume"]  ,
                   fillna = False).acc_dist_index()

['nvi']         = NegativeVolumeIndexIndicator(close = ["close"],volume = ["volume"] , fillna =
                  False).negative_volume_index()

['obv']        = OnBalanceVolumeIndicator(close = ["close"],volume = ["volume"] , fillna =
                     False).on_balance_volume()

['emi10']    = EaseOfMovementIndicator(high = ["high"],low = ["low"],volume = ["volume"] , window= 10 ,
                   fillna = False).ease_of_movement()

['emi20']    = EaseOfMovementIndicator(high = ["high"],low = ["low"],volume = ["volume"] , window= 20 ,
                       fillna = False).ease_of_movement()

['vwap5']   =  ["close"]/VolumeWeightedAveragePrice(high = ["high"],low = ["low"],close =  ["close"],
                  volume = ["volume"],window = 5  , fillna = False).volume_weighted_average_price() -1

['vwap10'] = ["close"]/VolumeWeightedAveragePrice(high = ["high"],low = ["low"],close = ["close"],
                   volume = ["volume"],window = 10  , fillna = False).volume_weighted_average_price() -1

['vwap20'] = ["close"]/VolumeWeightedAveragePrice(high = ["high"],low = ["low"],close =["close"],
                     volume = ["volume"],window = 20  , fillna = False).volume_weighted_average_price() -1

['vwap40'] = ["close"]/VolumeWeightedAveragePrice(high = ["high"],low = ["low"],close =["close"],
                   volume = ["volume"],window = 40  , fillna = False).volume_weighted_average_price() -1
```

APPENDIX 1b

LIST OF PRICE DERIVED FEATURES
(MARKED INSIDE SQUARE BRACKETS)

```
return_series          =  ["close"]/["close"].shift(1) -1
return_volume          =  ["volume"]/["volume"].shift(1) -1
close                  =  ["close"]
raw_volume             = ["volume"]
raw_trades             = ["trades"]

rets                   = return_series
volrets                = np.sign(rets)*np.sign(return_volume)

sign                   = np.sign(rets)
of1                    = sign*raw_volume
of2                    = sign*np.log(raw_volume)

["raw_volume"]         = raw_volume
["raw_trades"]         = raw_trades
["reversion"]          = close.rolling(20).apply(reversion)

["ret5mean"]           = rets.rolling(5).mean()
["ret10mean"]          = rets.rolling(10).mean()
["ret20mean"]          = rets.rolling(20).mean()

["volret5mean"]        = volrets.rolling(5).mean()
["volret10mean"]       = volrets.rolling(10).mean()
["volret20mean"]       = volrets.rolling(20).mean()

["ofa5"]               = of1.rolling(5).mean()
["ofb5"]               = of2.rolling(5).mean()
["ofa10"]              = of1.rolling(10).mean()
["ofb10"]              = of2.rolling(10).mean()
["ofa20"]              = of1.rolling(20).mean()
["ofb20"]              = of2.rolling(20).mean()
["ofa40"]              = of1.rolling(40).mean()
["ofb40"]              = of2.rolling(40).mean()

["skew10"]             = return_series.rolling(10).skew()
["skew20"]             = return_series.rolling(20).skew()
["skew40"]             = return_series.rolling(40).skew()

["kurt10"]             = return_series.rolling(10).kurt()
["kurt20"]             = return_series.rolling(20).kurt()
["kurt40"]             = return_series.rolling(40).kurt()
```

APPENDIX 1c

LIST OF PREVIOUS RETURNS BASED FEATURES
(MARKED INSIDE SQUARE BRACKETS)

```
return_series     = ["close"]/["close"].shift(1) -1
close             = ["close"]

["retl11"]   = ["close"].shift(1)/["close"].shift(2) -1
["retl12"]   = ["close"].shift(2)/["close"].shift(3) -1
["retl50"]   = ["close"].shift(0)/["close"].shift(5) -1
["retl51"]   = ["close"].shift(1)/["close"].shift(6) -1
["retl100"] = ["close"].shift(0)/["close"].shift(10) -1
["retl101"] = ["close"].shift(1)/["close"].shift(11) -1
["retl200"] = ["close"].shift(0)/["close"].shift(20) -1
["retl201"] = ["close"].shift(1)/["close"].shift(21) -1
```

APPENDIX 1d

LIST OF ECONOMETRICS BASED FEATURES
(MARKED INSIDE SQUARE BRACKETS)

```
return_series   = ["close"]/["close"].shift(1) -1
close           = ["close"]
raw_volume      = ["volume"]
raw_trades      = ["trades"]
volat_series    = return_series*return_series

bp0             = np.abs(return_series)*np.abs(return_series.shift(1))
sd0             = return_series*return_series
gc0             = np.sqrt(volat_series.ewm(span = 2.0/0.9-1).mean())

["gc0"]         = gc0

["std5"]        = return_series.rolling(5).std()
["std10"]       = return_series.rolling(10).std()
["std20"]       = return_series.rolling(20).std()
["std40"]       = return_series.rolling(40).std()


["bpv5"]        = bp0.rolling(5).mean()
["bpv10"]       = bp0.rolling(10).mean()
["bpv20"]       =  bp0.rolling(20).mean()

["bpvstd5"]     = ["bpv5"] - ["std5"]
["bpvstd10"]    = ["bpv10"] - ["std10"]
["bpvstd20"]    = ["bpv20"] - ["std20"]

["gc0bp5"]      = ["gc0"] - ["bpv5"]
["gc0bp10"]     = ["gc0"] - ["bpv10"]
["gc0bp20"]     = ["gc0"] - ["bpv20"]

["gc0std5"]     = ["gc0"] - ["std5"]
["gc0std10"]    = ["gc0"] - ["std10"]
["gc0std20"]    = ["gc0"] - ["std20"]
```

APPENDIX 1e

LIST OF SEASONAL FACTORS BASED FEATURES
(MARKED INSIDE SQUARE BRACKETS)

```
return_series    = ["close"]/["close"].shift(1) -1
close            = ["close"]
raw_volume       = ["volume"]
raw_trades       = ["trades"]
rets             = return_series

["time_label"]   = [0,1,2,….47]  (based on half hour intervals)
["rets1"]        = rets.shift(48)
["rets0"]        = rets.shift(24)

vol_s1           = (raw_volume.shift(1*48) + raw_volume.shift(2*48) + raw_volume.shift(3*48) +
                    raw_volume.shift(4*48) + raw_volume.shift(5*48))/5

trd_s1           = (raw_trades.shift(1*48) + raw_trades.shift(2*48) + raw_trades.shift(3*48) +
                    raw_trades.shift(4*48) + raw_trades.shift(5*48))/5

volumes1         = raw_volume/vol_s1 - 1
trades1          = raw_trades/trd_s1 - 1

["volumes1"]     = volumes1
["trades1"]      = trades1
["volumei1"]     = np.where(volumes1 > 0.5,1,0)*rets
["tradesi1"]     = np.where(trades1 > 0.5,1,0)*rets
["volumei2"]     = np.where(volumes1 > 1,1,0)*rets
["tradesi2"]     = np.where(trades1 > 1,1,0)*rets
["volumei3"]     = np.where(volumes1 > 2,1,0)*rets
["tradesi3"]     = np.where(trades1 > 2,1,0)*rets
```

APPENDIX 2

Modelling the problem as binary classification with only 2 labels :**1** if the price goes up, **-1** if the price goes down. If the price doesn't change then we simply put the previous direction as label.

AdaBoost Classifier: AdaBoost classifier with Decision tree classifier as base learners are used.

RandomForest Classifier: RandomForest Classifier is used.

XGBoost Classifier: XGBoost classifier with gbtree as base learners is used.

NeuralNetwork Classifier: Multi-Layer Perceptron classifier was used.

The methodology and hyperparameters to optimize were similar to the one used in 3- way classification. Again, the parameters with best results in Validation set data were used to infer the performance in the test set.

BTCUSDT				
	ADABOOST	RANDOM FOREST	XGBOOST	MLP
ACCURACY	0.54	0.56	0.55	0.51
RETURNS	-349%	-498%	-451%	-433%
SHARPE RATIO	-7.3	-10.41	-9.41	-9.05

The performance in terms of returns and Sharpe is worse than 3-way classification. However, it was mostly due to large amount of transactions cost incurred as a result of capturing small movements.

We can hence conclude that binary classification is not the way to go at least in such smaller time frames where volatility is not enough to cover transaction costs. An interesting application can be to try to devise trend following systems using current price above some "historical average" as labels.

APPENDIX 3

This section discusses the results of applying ML algos on 15 minutes dataset for BTCUSDT only. The following Models were used:

AdaBoost Classifier: AdaBoost classifier with Decision tree classifier as base learners are used.

RandomForest Classifier: RandomForest Classifier is used.

XGBoost Classifier: XGBoost classifier with gbtree as base learners is used.

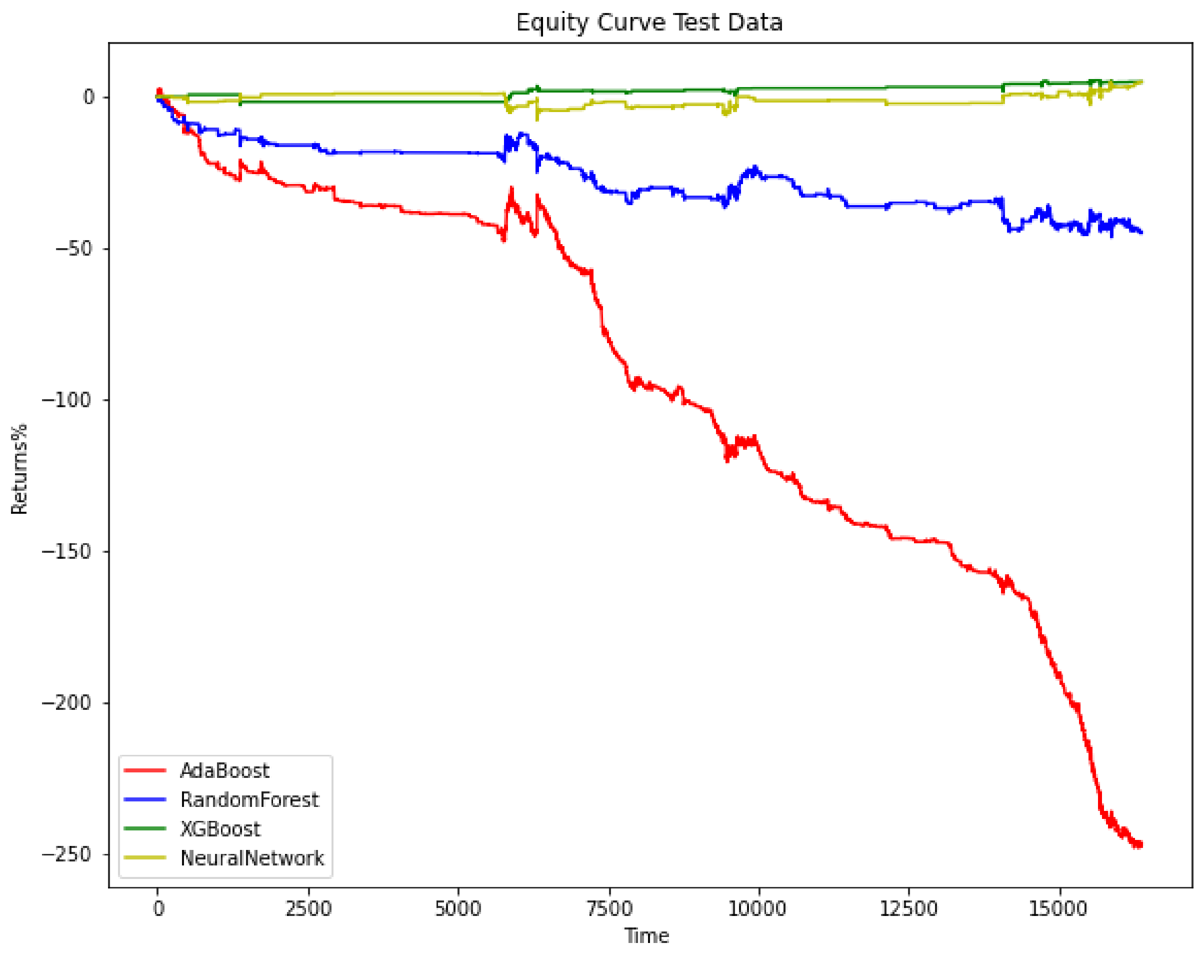
NeuralNetwork Classifier: Multi-Layer Perceptron classifier was used.

The methodology and hyperparameters to optimize were similar to the one used in 3- way classification on 30 minutes data. Again, the parameters with best results in Validation set data were used to infer the performance in the test set.

BTCUSDT				
	ADABOOST	RANDOM FOREST	XGBOOST	MLP
ACCURACY	0.64	0.73	0.87	0.87
RETURNS	-271%	-50%	6.10%	7.85%
SHARPE RATIO	-9.5	-2.07	0.8	1.46

Adaboost and RandomForest have performed significantly worse than 30-minute time period. XGBoost has performed well on this data set as well with an accuracy of 87% and so has MLP.

One noticeable factor was extremely large number of “0” classification as compared to other classes. Hence the large accuracy can be explained by tendency of the models to classify most of the labels to dominant class. This will result in no economic gain from running the strategy live.


Equity Curve Test Data
Returns%
Time
AdaBoost
RandomForest
XGBoost
NeuralNetwork