Predicting Ethereum Returns Using Machine Learning Techniques

Jonathan Caldwell Christopher Hemm Satya Biswal Zach Peschke
Shreyash Agarwal Srikanth Geedipalli

Abstract

Given the dearth of fundamental asset data for cryptocurrencies, we speculate focusing heavily on technical analysis of financial time-series data may yield predictive capacity. We choose Ethereum as the candidate for analysis, querying price/volume data, ethereum-specific blockchain data, and apply a number of technical indicators to predict returns. We find that, while aggregate performance metrics for our selected Supervised Learning techniques are sub-optimal, selective application of these results to "trading simluations" may nonetheless provide value.

Introduction

Cryptocurrencies have enjoyed immense popularity in recent years, with sentiment varying widely throughout society. Nearly everyone knows of Bitcoin's unprescedented surge in price, and subsequent plummet. This incredible volatility of cryptoassets is enticing to the risk-tolerant investor, and has potential to be lucrative, conditional on correct price predictions. However, unlike traditionally-traded securities with a wealth of fundamental data (for example, company financial statements and press releases), cryptoassets seem to have none. We speculate, given there exist no true "fundamentals" for cryptocurrencies (only quasi-fundamentals, like block-chain data), technical analysis may have significant utility when applied to these assets. We select a liquid, publicly traded cryptocurrency called Ether, which uses Ethereum as its decentralized platform (for purposes of simplicity, we use the term ethereum as the all encompassing concept of both platform and currency, for the remainder of the paper).

In addition to having the second highest market capitalization behind Bitcoin, the Ethereum blockchain supports much more functionality than Bitcoin by enabling Smart Contracts and Distributed Applications. This additional functionality may approximate fundamental data and therefore cause Ethereum's price movements to be more predictable than that of Bitcoin. Ethereum has also shown itself to be a more frequently traded and volatile cryptocurrency than Bitcoin, which may be as a result of 1) Ethereum being a newer currency and 2) Ethereum's trade through contract accounts allowing for accessibility and greater use cases for exchange than Bitcoin. Ethereum's higher volatility and additional functionality motivates our aim to predict directionality of Ethereum price changes through various machine learning models.

The structure of the paper will be as follows: We first introduce the data types we ammassed from various websites and APIs, and detail the considerations/transformations we performed on this data to make it most compatible with state of the art machine learning techniques. We'll then introduce 4 different supervised model techniques implemented, with the aim of predicting directionality of ethereum movement in a given trading day. We discuss the accuracy and results of each in their respective section, then provide some concluding remarks about value-add from an applied perspective.

OHLCV Data

Our primary data source is ethereum time-series data, collected on the Coinbase Platform. This data details the Open, High, Low and Close price for ethereum on a given trading day, as well as the aggregate

volume passing through Coinbase. In using this data, we make two key assumptions:
- Ethereum pricing mechanics are efficient enough that using Coinbase pricing is representative of ethereum as a whole (no significant arbitrage across exchanges)
- Ethereum volume on Coinbase (currently the largest exchange) is at least proportionally indicative of market-wide ethereum trading volume

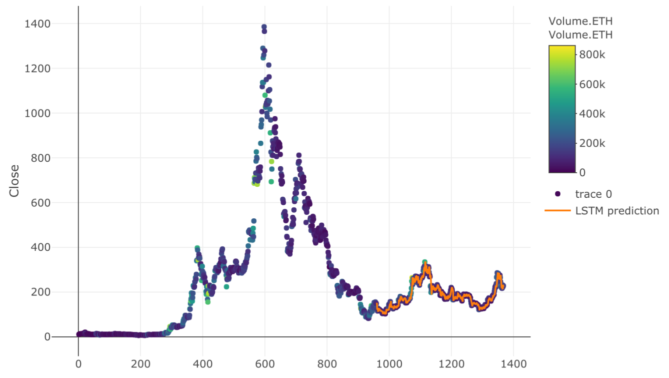


Figure 1: ETH Price, 5/16-3/20 (test sample in orange)

Blockchain Data

Similar to Bitcoin, Ethereum has a blockchain consisting of blocks of data, whether they take the form of transactions and SmartContracts. These blocks are mined by participants, and said blocks are distributed, implicitly validating them.

Blockchain data contains metrics pertaining to the Ethereum blockchain underlying the ether currency. This data is queried from an API provided by CryptoCompare, an online market research company for crypto assets. Sample predictors include:

- Hashrate: The estimated number of tera hashes per second (trillions of hashes per second) the Ethereum network is performing
- Difficulty: A measure of how difficult it is to find a hash below a given target.
- Block Height: The number of blocks in the chain between any given block and the very first block in the blockchain.

We believe, in the absense of traditional fundamental data, these blockchain metrics may provide quasi-fundamental insights, data which may provide information about relative supply/demand dynamics.

Technical Indicators (TIs)

Technical indicators are widely used in the financial sector to illustrate/explain trends in time-series market data. We posit inclusion of technical analysis (which necessitates only OHLCV data) may provide valuable information for predicting ethereum returns. We include a total of 22 technical indicators as predictors. One such example are Bollinger Bands, a TI which details levels one standard deviation above and below a simple moving average of the price. Relative distance of these ‘envelope’ levels and the actual price is informative of the volatility of the asset (visualizaion of BBands in *Figure 2*). For an exhaustive list of Indicators used, refer to our code provided in the appendix.

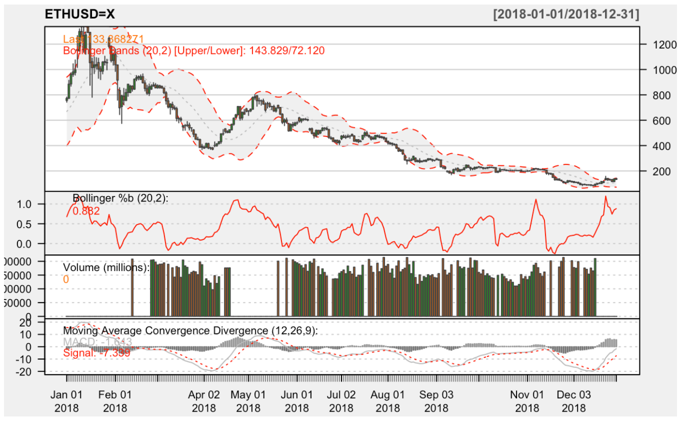


Figure 2: 2018 Ethereum Price with Bollinger Bands

Initial Data Considerations

Data Differencing

As we are utilizing time-series data for our analysis, conventional Machine Learning research dictates data should be stationarized to improve prediction validity. In particular, we apply a differencing function to the price of our asset which, for each observation, represents the difference between the current and last price. Formally:

$$\Delta_t^{ether} = Price_t^{ether} - Price_{t-1}^{ether}$$

Given the two prices are close values for ethereum, we expect Δ to represent the daily return over observed day t.

Additionally, we performed a Kwiatkowski-Phillips-Schmidt-Shin test (KPSS), which indicated a differencing of 1 was sufficient to stationarize data (*Figure 3* shows example time-series analysis)

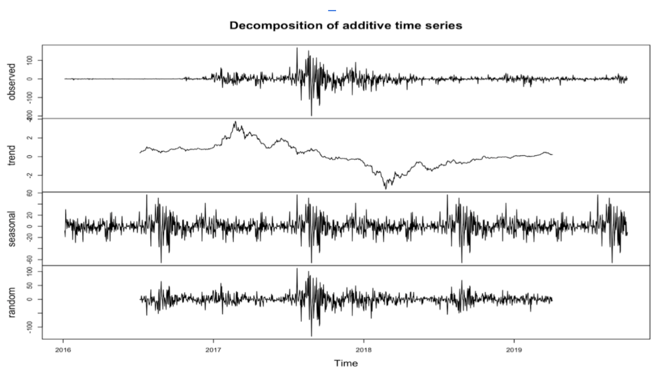


Figure 3: TS Decomposition

A zero is appended to the beginning of the price vector to maintain data-frame length consistency, and the first observation will be discarded from analysis.

Lastly, data is scaled to a normal distribution using base R scale function. Scaler information is maintained to unscale predictions in subsequent sections.

Data Lagging

Given we are attempting to predict asset returns using historic prices, predictors must be lagged relative to the response so as not to confound results. For example, if attempting to predict returns on an asset during trading day t, the close price, $Price_t^{ether}$ is not yet realized, thus we must lag observations one period to provide a realistic learning environment for the Neural Network.

After application of Lagging function to the now-differenced returns, we're left with a Null value at $t = 1$ (generated due to lack of information about $Return_{t-1}^{ether}$ at time $t = 1$). We convert this value to zero, and will discard from analysis.

Data Lagging is performed either explicitly to every time-series input (OHLCV and Blockchain Data), or implicitly by the inputs from which it was transformed (as in the case of Technical Indicators)

Training and Testing Split

For purposes of validating our methodology, we will split the data into a 70:30 Train/Test split, without data randomization. Lack of shuffling is important due to the sequence dependencies present in the data, which we hope to realize with the models. After splitting data, we're left with 957 days of training data, and 410 days of testing data.

Full Data Procedures in R are included in the appendix.

LSTM

We begin with implementation of a Long-Short-Term-Memory Neural Network, a type of Recurrent Neural Network which deals well with long-term dependencies. Specifically, we'll use Keras and Tensorflow libraries to construct the Neural Network.

LSTM-Specific Data Considerations

For compatability with LSTM layers in Keras, data must be coerced into a 3-dimensional array:
- The first dimension corresponds to the number of observations in the sample.
- The second dimension refers to the time-step utilized, which in our case, is the 1-day lag for predictor variables.
- The third dimension represents the number of predictor variables used for the model; in our case, we have with fifty-one predictors for preliminary analysis (OHLCV Data + Blockchain Data + Technical Indicators).

Model Implementation

Specifications for Keras Model:

We use a multivariate model with 5 layers

- The model contains two LSTM layers, each with tanh activation functions

- The model contains two dropout layers, with rates of 0.3 and 0.2, respectively

- The last is a dense layer with one unit (representing the singular output) and a tanh activation function

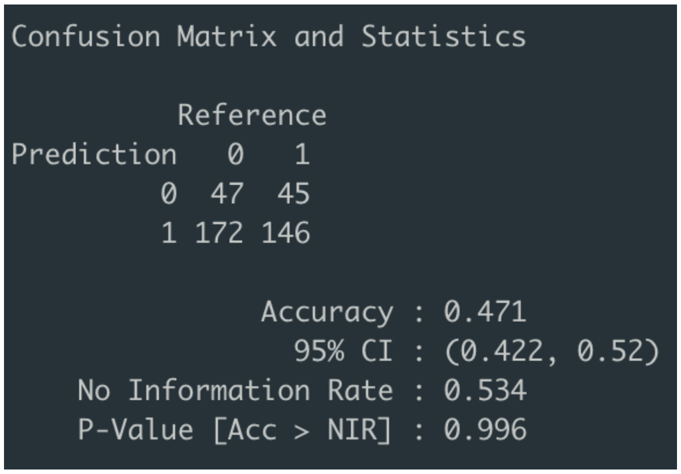
Our compiler uses Mean Average Error for the Loss function, with additional adam technique for the optimization function
- Both specifications were recommended for use in models with time-series data in Keras documentation.

We then fit the model, using a technique simulating multiple epochs (by simply iterating over the Keras.fit() function), but without shuffling, to maintain dependencies. We also reset model state after each iteration, which is recommended with LSTM models.

Prediction and Evaluation

After predicting on the test data, we unscale the predictions, using the same mean and standard deviation applied to standardize the differenced returns vector before feeding into the model.

We then create a binary representation of both the predicted and actual values, for construction of a Confusion Matrix to evaluate the Up/Down daily movement classification (*Figure 4*).



```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0  47  45
         1 172 146

               Accuracy : 0.471
                 95% CI : (0.422, 0.52)
    No Information Rate : 0.534
    P-Value [Acc > NIR] : 0.996
```

Figure 4: LSTM Confusion Matrix

LSTM Results

While we were inially quite optimistic about using RNN for predicted returns, accuracy consistently failed to significantly outperform the no information rate, as evidenced in the confusion matrix above. The ‘no information rate’ represents the effective skew of the data, and the prediction accuracy achieved by simply predicting the most frequent response in the sample at every observation.

Inclusion of Technical Indicators in the model did not significantly increase prediction accuracy. Predicted values for training set relied heavily on the activation function used in the last dense layer, and complexities with data scaling made robust predictions difficult to generate. Accuracy and error rate were actually lower with only the lagged difference in close price; since this univariate data feed is strictly less informative than the multivariate, we fear our analysis overfit the data, which is not suprising given the chronological scarcity for ethereum data.In the next section, we discuss our findings using supervised models.

Random Forest

Next, we utilize random forest to predict ethereum returns. Random Forest is a tree-based machine learning method which uses subsettable bootstrap aggregation, and is known for its powerful out-of-box predictive accuracy. Its handling of unscaled data, and its minimal hyperparameter tuning makes it an ideal candidate for multivariate analyses.

We begin with a random forest implementation using OHLCV, Blockchain, and our list of aforementioned Technical Indicators. We begin with ntree = 500 and nodesize = 15, both arbitrary and unoptimized parameters to mitigate overfitting. Variable Importance on the training set included in *Figure 5*.

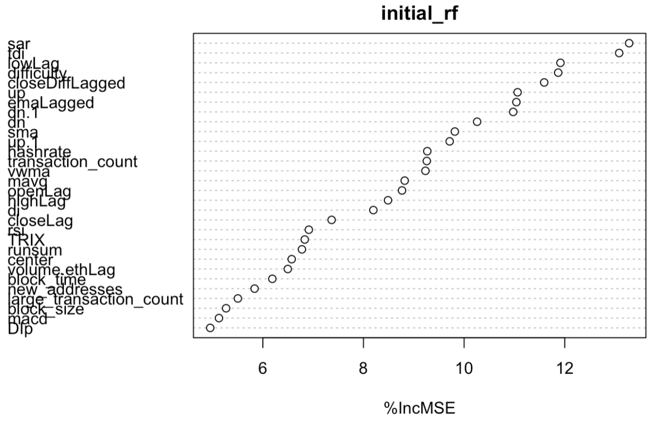
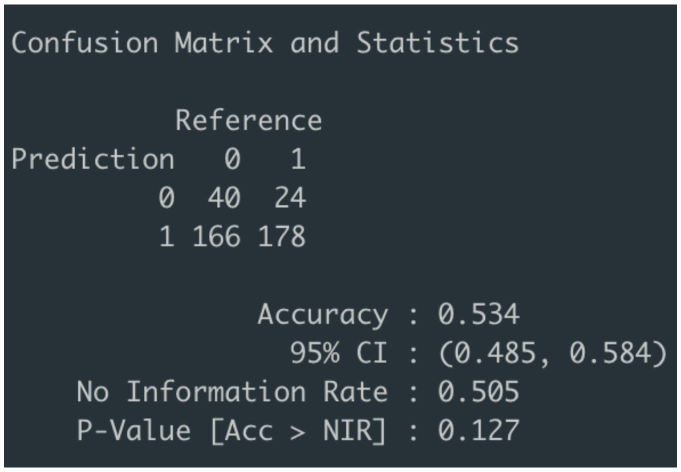


Figure 5: RF Variable Importance

Note that in the top 10 most informative predictors (by % increase in MSE, a metric detailing the estimated increase in error had the predictor not been present), 1 is OHLC (Low), one is Blockchain (Difficulty), with the other 8 being Technical Indicators.

For Random Forest, to discourage overfitting, we decided to leave hyperparameters unoptimized; *Figure 6 and 7* detail the confusion matrix, and accuracy statistics for the final model.

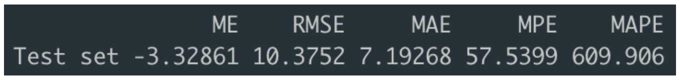


```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0  40  24
         1 166 178

               Accuracy : 0.534
                 95% CI : (0.485, 0.584)
    No Information Rate : 0.505
    P-Value [Acc > NIR] : 0.127
```

Figure 6: RF Confusion Matrix

We see aggregate accuracy is near margin (not statistically greater than 50% using a 95% CI). RMSE is significantly above MAE, likely indicating divergence in prediction on larger values.

To quantify prediction results on another dimension, we also utilized transformations of the binary prediction vector to simulate different trading strategies over the window of the test sample.



```
              ME     RMSE      MAE      MPE     MAPE
Test set -3.32861 10.3752 7.19268 57.5399 609.906
```

Figure 7: RF Accuracy Metrics

- Long only strategies would use the traditional (1,0) classification, product of this traditional vector with the actual returns would yield long-only returns each day.
- Long/Short strategies would use a (1,-1) classification, simulating taking a directional position every trading day. Vector product of this classifier and the daily returns yields the long-short returns each day.

Cumulative summation functions are used to calculate the historic returns, graphical representation is provided in *Figure 8*.

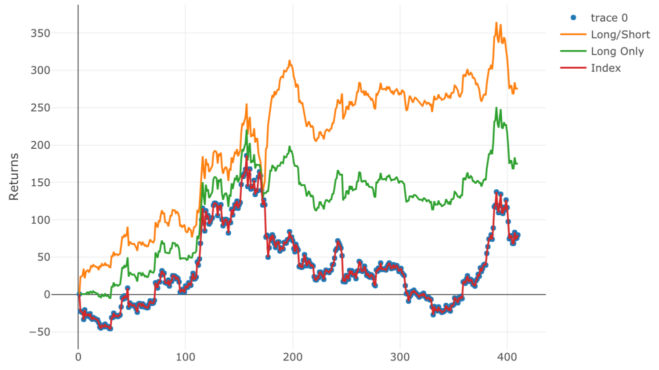
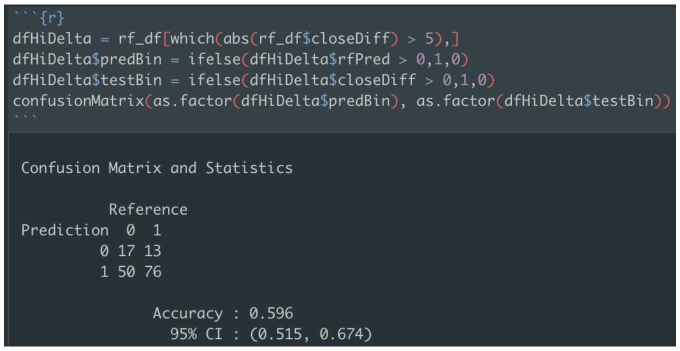


Figure 8: RF Returns

We see even at near-margin aggregate prediction rates, simply by using a directional signal strategy, we can perform significantly better than the index. While these results may at first seem at odds with one another, we hypothesize the abnormal 'returns' are a product of Random Forest's prediction efficacy on high-volatility trading days. To illustrate this point, we construct a confusion matrix for all trading days with a realized delta (daily ether return) greater than $5 USD (*Figure 9*).



~~~
```{r}
dfHiDelta = rf_df[which(abs(rf_df$closeDiff) > 5),]
dfHiDelta$predBin = ifelse(dfHiDelta$rfPred > 0,1,0)
dfHiDelta$testBin = ifelse(dfHiDelta$closeDiff > 0,1,0)
confusionMatrix(as.factor(dfHiDelta$predBin), as.factor(dfHiDelta$testBin))
```
~~~

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 17 13
         1 50 76

               Accuracy : 0.596
                 95% CI : (0.515, 0.674)
```

Figure 9: RF, High Delta Confusion Matrix

Therefore, we speculate while Random Forest's aggregate prediction accuracy is near margin, it is able to predict large movement days with significance. Thus, when using such a signal in a naive trading stragegy, returns from correct prediction on high-volatility days outweigh the potentially near-margin prediction when absolute price movement is relatively low.

This motivated the creation of another trading strategy option: Long/Short/Abstain. We use similar rules as the Long/Short strategy before, but if the magnitude of RF prediction is less than 0.5, we simply abstain from taking a position (*Figure 10*). We see such a strategy marginally outperforms our existing Long/Short strategy.

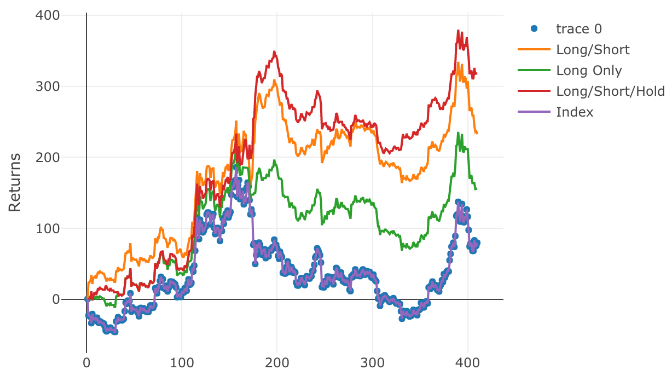


Figure 10: RF Returns

ROC and AOC are quite marginal, even with our optimized RF model, as seen in *Figure 11*. AOC is approx. 0.533. These results do not show convincing evidence for random forest's ability to predict directionality of movement over the entire test set.

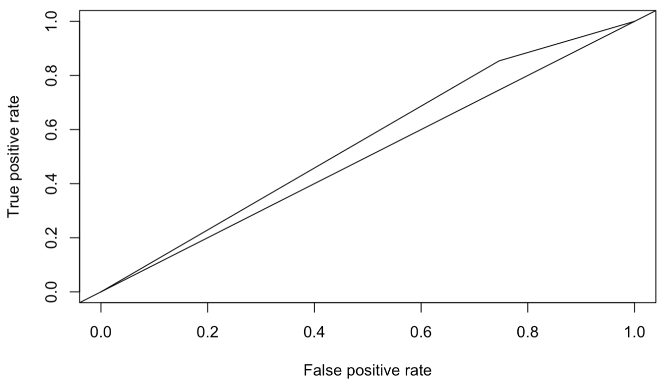
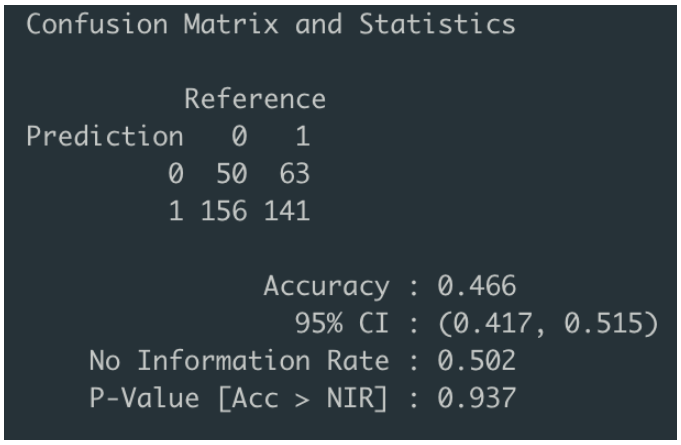


Figure 11: RF ROC

Boosting

Next, we implement a Gradient Boosting Model to predict ethereum returns. Boosted models uses a large ensemble of weak decision trees to inform predictions; moreso than Random Forest models, boosting models may be significantly improved by optimizing over hyperparameters. We begin with a naive implementation of a GBM, starting with an initial 1000 trees, with tree depth unspecified (*Figure 12* provides the confusion matrix for the initial implementation).

After running the initial naive model, we conduct variable selection, excluding all variables scoring below a variable importance threshold of 0.8; this excludes a large number of near-zero importance variables in the model (visualization of Variable Importance in *Figure 13*). Similar to the Random Forest variable importance analysis, we see a large proportion of the most informative variables are technical indicators, such as TDI, a trend indicator.



```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0  50  63
         1 156 141

               Accuracy : 0.466
                 95% CI : (0.417, 0.515)
    No Information Rate : 0.502
    P-Value [Acc > NIR] : 0.937
```

Figure 12: RF, High Delta Confusion Matrix

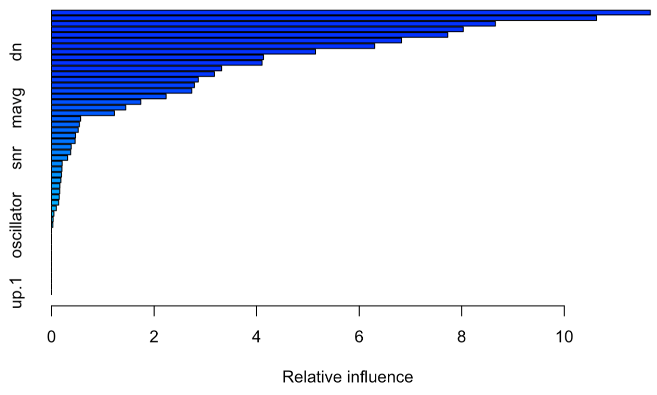
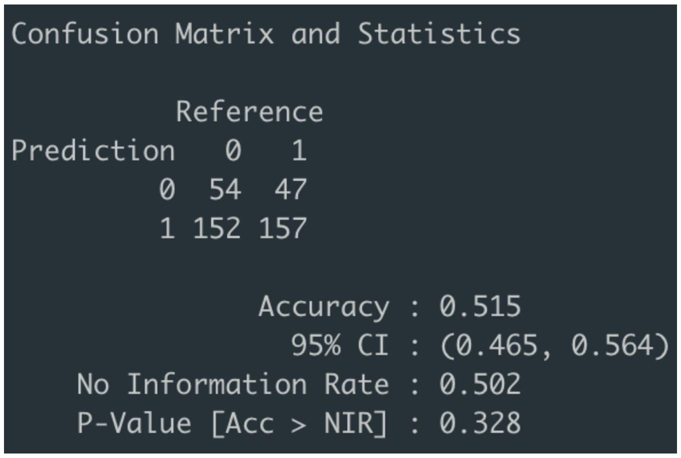


Figure 13: GBM Relative Importance

We then rerun the boosting model, now iterating over a number of hyperparameters to optimize on the training set, using RMSE as the objective optimizer (as this is fundamentally still a regression, rather than a classification model). *Figure 14* shows the resultant test-set confusion matrix for the optimized model. We see percent accuracy significantly improves, but not enough to say with confidence that prediction exceeds 50% over the entire test-set.



```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0  54  47
         1 152 157

               Accuracy : 0.515
                 95% CI : (0.465, 0.564)
    No Information Rate : 0.502
    P-Value [Acc > NIR] : 0.328
```

Figure 14: GBM Confusion Matrix

Boosting ROC curve suggests non-significance of predictions, with AUC at approximately 0.52 (*Figure 15*).

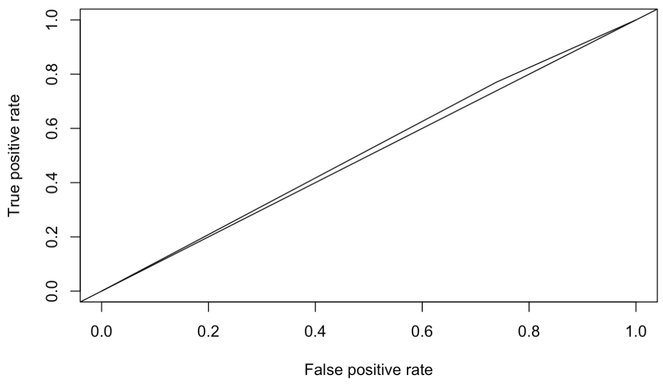
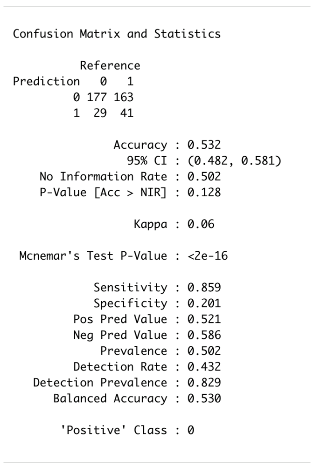


Figure 15: GBM ROC

ARIMA

Finally, we implement an ARIMA (Autoregressive Integrative Moving Average) Model to predict ethereum returns. ARIMA transforms the time series data into a stationary time series, and predictions are a linear regression upon the time differences as well as additional exogenous variables, which include volume traded, total currency supply, and transaction count on the blockchain. We lag and difference the data by an order of 1. We include less exogenous variables than the predictor variables in the boosting and random forest models to avoid perfect multicollinearity (*Figure 16, 17, 18* provide the confusion matrix, returns, and ROC curve respectively for the ARIMA implementation).

While the ARIMA model exhibits adequate predictability, it is unsuccessful relative to the Random Forest model in predicting large movement days, resulting in only marginally better returns over the index in



```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 177 163
         1  29  41

               Accuracy : 0.532
                 95% CI : (0.482, 0.581)
    No Information Rate : 0.502
    P-Value [Acc > NIR] : 0.128

                  Kappa : 0.06

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.859
            Specificity : 0.201
         Pos Pred Value : 0.521
         Neg Pred Value : 0.586
             Prevalence : 0.502
         Detection Rate : 0.432
   Detection Prevalence : 0.829
      Balanced Accuracy : 0.530

       'Positive' Class : 0
```

Figure 16: ARIMA, Confusion Matrix

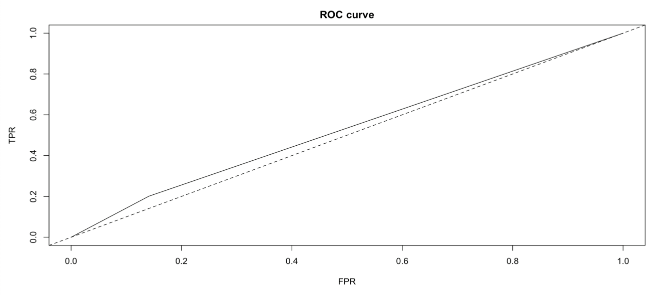


Figure 17: ARIMA, ROC

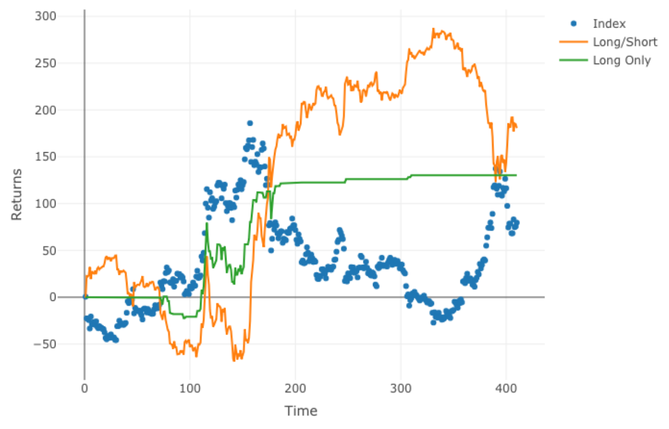


Figure 18: ARIMA, Returns

both trading strategies. Finally, the ARIMA ROC curve also implies non-significant predictive accuracy, with AUC at approximately 0.53.

Conclusion

Our primary goal at the onset of the project was to test the hypothesis that, given the absence of fundamental securities data, returns on cryptocurrencies, such as ethereum, are predictable using their daily price/volume data. We applied numerous technical indicators to this price/volume data (as is common with traders of traditional financial assets), included quasi-fundamental blockchain data, and applied a variety of machine learning models to test the efficacy of this hypothesis.

We conclude that Supervised models, particularly Random Forest, perform best at predicting directional daily returns of ethereum. Boosting, though heavily optimized, failed to perform as well as the out of box predictions generated by RF, leading us to believe attempts to Gradient Boost on this short window of time-series data may simply encourage overfitting. Overfitting is a similar issue facing the neural network used, as LSTM produced consistently suboptimal predictive accuracy.

With regards to conventional machine learning and regression metrics, even our best model (Random Forest) performs poorly over aggregate Out Of Sample window, as RMSE, MSE, and ME were clearly quite high. Discretizing our predicted values to test directional predictive accuracy yielded % accuracy near 50% margin, with statistics pointing towards non-significance.

Given however, that the objective of the exercise (predicting daily returns) extends beyond simply error mitigation, we adopt performance evaluation on an alternative dimension, particularly that of strategic market returns. Application of these discrete predictions for naive trading strategies provide a more promising picture on Supervised Learning's ability to provide value. Utilizing Random Forest's predictions, we see a number of strategies (Long/Short/Hold, Long/Short, Long-only) have hypothetical returns exceeding that of the ethereum index. We assert that these returns persist, even in the face of potentially marginal accuracy rates, due to the capacity of Random Forest to predict high-volatility days with non-marginal significance. ARIMA yielded comparable accuracy and AOC values to RF, but failed to consistently correctly predict high-delta trading days, as evidenced from its strategies' high volatility of returns. This may suggest RF's capacity for high-dimensional data gives it an advantage in prediction of high-delta days over traditional autoregressive models. Obviously, these quasi-return calculations exclude transaction costs, as well as other potential trading costs; however, the margin by which RF outperforms the Ethereum index is promising.

References

- https://cran.r-project.org/web/packages/TTR/TTR.pdf
- https://pdfs.semanticscholar.org/ceff/65e02b2b9b6b181cfc956350351b8e284a01.pdf
- https://min-api.cryptocompare.com
- https://www.cryptodatadownload.com/data/coinbase/
- http://rwanjohi.rbind.io/2018/04/05/time-series-forecasting-using-lstm-in-r/
- https://keras.io/examples/imdb_lstm/

Appendix

Data Procedures

```r
set.seed(123)
#loading in prive/volume data
ethereum = read.csv("Coinbase_ETHUSD_daily.csv")
ethereum = ethereum[seq(dim(ethereum)[1],1),]

#Creation of Response Variables
ethereum$closeDiff = append(0,diff(ethereum$Close, differences = 1))
ethereum$Y = ethereum$closeDiff
ethereum$Yscaled = scale(ethereum$closeDiff)

#OHLCV lag
ethereum$openLag = Lag(ethereum$Open,1)
ethereum$highLag = Lag(ethereum$High,1)
ethereum$lowLag = Lag(ethereum$Low,1)
ethereum$closeLag = Lag(ethereum$Close,1)
ethereum$volume.ethLag = Lag(ethereum$Volume.ETH,1)

ethereum$Date = as.Date(mdy(as.character(ethereum$Date)))

ethereum$closeDiffLagged = Lag(ethereum$closeDiff, 1)
ethereum$emaLagged = EMA(ethereum$closeLag, 1)

#loading in and cutting to appropriate window
eth_chain = read.csv('chain_data_ETH.csv')
eth_chain$Date = as.Date(as.POSIXct(eth_chain$time, origin="1970-01-01"))
eth_chain = eth_chain[eth_chain$Date >= as.POSIXct("2016-05-27"), ]
eth_chain = eth_chain[eth_chain$Date <= as.POSIXct("2020-03-05"), ]

#lag columns of eth_chain
z = colnames(eth_chain)
z = z[!z %in% c('id', 'symbol', 'time', 'Date')]
for (i in 1:length(z)) {
  eth_chain[, z[i]] = Lag(eth_chain[, z[i]], 1)
}
eth_chain = eth_chain[-c(1), ]
data = merge(ethereum, eth_chain, by="Date", all = T)

#remove missing dates
data = data[-c(1028:1039), ]
data

# creation of technical indicators, using ttr library
eth_hlc = data[, c('highLag', 'lowLag', 'closeLag')]
eth_hl = data[, c('highLag', 'lowLag')]
eth_close = data[, c('closeLag')]
data$snr = SNR(eth_hlc, 5)
data$obv = OBV(data[, c('closeLag')], data[, c('volume.ethLag')])
data$runsum = runSum(data[, c('closeLag')], n=5)
data$vwma = VWMA(data[, c('closeLag')], 5, volume=data[, c('volume.ethLag')])
data$rsi = RSI(eth_close, n=5)
data$roc = ROC(eth_close)
```

```r
data$percentrank = runPercentRank(eth_close, n =5)
data$mfi = MFI(eth_hlc, data[, c('volume.ethLag')], n=5)
data$cci = CCI(eth_hlc,n=7)
data$clv = CLV(eth_hlc)
data$sma = SMA(eth_close, n=5)

#cbinding the multiple-column technical indicators
aroon = as.data.frame(aroon(eth_hl, 4))
data = cbind(data, aroon)
bbands = BBands(eth_hlc, 5)
data = cbind(data, bbands)
trix = TRIX(eth_close, n=2, nSig=9)
data = cbind(data, trix)
tdi = TDI(eth_close, n=3)
data = cbind(data, tdi)
gmma = GMMA(eth_close, short = c(3, 5, 8, 10, 12, 15), long = c(30, 35, 40, 45, 50, 60))
data = cbind(data, gmma)
pbands = PBands(eth_close, n=3)
data = cbind(data, pbands)
sar = SAR(eth_hl, accel = c(0.02, 0.2))
data = cbind(data, sar)
stoch = stoch(eth_hlc, nFastK = 5)
data = cbind(data, stoch)
macd = MACD(eth_close, nFast = 2, nSlow = 5, nSig = 5)
data = cbind(data, macd)
adx = ADX(eth_hlc, n = 5)
data = cbind(data, adx)

#data partitioning
N = nrow(data)
n = round(N * 0.7, digits = 0)
train_new = data[1:n,]
train_new = train_new[-c(1:7), ]
test_new = data[(n+1):N,]

mask = colnames(train_new)
#removing all variables that have NAs, or any variable which may confound predictions
mask = mask[!mask %in% c('Date', 'Symbol', 'id', 'symbol', 'time',
                         'Open', 'High', 'Low', 'Close', 'Volume.ETH','Volume.USD',
                         'short lag 10', 'short lag 12', 'short lag 15', 'short lag 30',
                         'short lag 35','short lag 8', 'long lag 30', 'long lag 35', 'long lag 40',
                         'long lag 45', 'long lag 50', 'long lag 60', 'slowD','ADX', 'short lag 3',
                         'short lag 5','signal','obv','closeDiff','Yscaled')]
#create master train/test dataframes, for usage in rf, boosting, and lstm
train_boost = train_new[, mask]
test_boost = test_new[, mask]
```

Boosting Procedures

```r
library(gbm)
set.seed(123)
boost_diff_fit = gbm(Y~., data=train_boost, n.trees = 1000)
diff_model_boost <- predict(boost_diff_fit, newdata = test_boost, n.trees = 1000)
```

```r
boosting_df_diff <- data.frame(test_new$Date, test_new$Y, diff_model_boost)
colnames(boosting_df_diff) <- c('time', 'Y', 'model')

ggplot(boosting_df_diff, aes(time)) +
  geom_line(aes(y = Y, colour = "actual")) +
  geom_line(aes(y = diff_model_boost, colour = 'predicted'))

binaryPred.diff.boost = ifelse(diff_model_boost > 0,1,0)
binaryPredSig.diff.boost = ifelse(diff_model_boost > 0,1,-1)
binaryTest = ifelse(test_new$Y > 0,1,0)

imp =summary(boost_diff_fit)
boost_vars = imp[imp$rel.inf >= 0.8, ]
boost_vars = as.vector(boost_vars[, 1])
boost_vars = append('Y', boost_vars)
train_final = train_boost[, names(train_boost) %in% boost_vars]
test_final = test_boost[, names(train_boost) %in% boost_vars]

boost_final = gbm(Y~., data=train_final, n.trees = 5000, interaction.depth = 5)
boostPredFinal <- predict(boost_final, newdata = test_final, n.trees = 5000)

finalBoostDF <- data.frame(test_new$Date, test_new$Y, boostPredFinal)
colnames(finalBoostDF) <- c('time', 'Y', 'model')

ggplot(finalBoostDF, aes(time)) +
  geom_line(aes(y = Y, colour = "actual")) +
  geom_line(aes(y = boostPredFinal, colour = 'predicted'))

binPredBoostFinal = ifelse(boostPredFinal > 0,1,0)
binPredSigBoostFinal = ifelse(boostPredFinal > 0,1,-1)
binaryTest = ifelse(test_new$Y > 0,1,0)

confusionMatrix(as.factor(binPredBoostFinal),as.factor(binaryTest))

#ROC curve
predBoost = prediction(binPredBoostFinal,binaryTest)
roc.perfB = performance(predBoost,measure = 'tpr',x.measure = 'fpr')
plot(roc.perfB)
abline(a=0,b=1)

auc.perfB = performance(predBoost, measure = "auc")
auc.perfB@y.values
```

Random Forest Procedures

```r
library(forecast)
library(randomForest)
set.seed(123)
initial_rf <-randomForest(Y~., data=train_boost, ntree=500,nodesize=15, importance=TRUE)
varImpPlot(initial_rf, type=1)

## naive accuracy results

set.seed(123)
```

```r
rfPred = predict(initial_rf, test_boost)
accuracy(rfPred,test_boost$Y)
rf_df <- data.frame(test_new$Date, test_new$Y, rfPred)
colnames(rf_df) <- c('time', 'closeDiff', 'rfPred')
indexHighDelta = which(test_boost$Y>5,arr.ind = TRUE)

binaryPred.rf = ifelse(rfPred > 0,1,0)
binaryPredSig.rf = ifelse(rfPred > 0 , 1, -1)
binaryTest.rf = ifelse(test_boost$Y > 0,1,0)

confusionMatrix(as.factor(binaryPred.rf),as.factor(binaryTest.rf))

#testing high delta predivtive accuracy
dfHiDelta = rf_df[which(abs(rf_df$closeDiff) > 5),]
dfHiDelta$predBin = ifelse(dfHiDelta$rfPred > 0,1,0)
dfHiDelta$testBin = ifelse(dfHiDelta$closeDiff > 0,1,0)
confusionMatrix(as.factor(dfHiDelta$predBin), as.factor(dfHiDelta$testBin))

#Simlulating Returns:

#Simulates long-only strat
binaryPred.rf[is.na(binaryPred.rf)] = 0
binaryPredSig.rf[is.na(binaryPredSig.rf)] = 0
returnLong = binaryPred.rf * test_boost$Y

# long/short/hold strat
magPredSig.rf = ifelse(abs(rfPred) < 0.5,0,ifelse(rfPred > 0.5,1,-1))
magPredReturns = magPredSig.rf * test_boost$Y
agReturnsMagStrat = cumsum(magPredReturns)
sum(magPredReturns)
#control returns
#Vector which simulates a long/short strategy consideration
binaryPredSig = ifelse(rfPred > 0,1,-1)
#Simulates buying or selling (long/short strat)
returnStrat = binaryPredSig.rf * test_boost$Y

agReturnsStrat = cumsum(returnStrat)
agReturnsLong = cumsum(returnLong)
Returns = cumsum(test_boost$Y)
agDF = data.frame(Returns,agReturnsStrat,agReturnsLong,agReturnsMagStrat)
DaysElapsed = 1:410
#plotting returns
plot_ly(agDF, x = DaysElapsed, y = ~Returns, type = "scatter", mode = "markers") %>%
 add_trace(y = agReturnsStrat, x = 1:410, name = "Long/Short", mode = "lines")%>%
  add_trace(y = agReturnsLong, x = 1:410, name = "Long Only", mode = "lines")%>%
  add_trace(y = agReturnsMagStrat, x = 1:410, name = "Long/Short/Hold", mode = "lines")%>%
  add_trace(y = ~Returns, x = 1:410, name = "Index", mode = "lines")

#variable importance analysis
varImpRF =initial_rf$importance
import = initial_rf$importanceSD
importVar = import[import > 2]
rf_vars = names(importVar)
rf_vars = append('Y', rf_vars)
rf_train_final = train_boost[, names(train_boost) %in% rf_vars]
rf_test_final = test_boost[, names(train_boost) %in% rf_vars]
```

```
final_rf <-randomForest(Y~., data=rf_train_final, ntree=500,nodesize=15, importance=TRUE)

rfPredFinal = predict(final_rf, rf_test_final)
accuracy(rfPredFinal,rf_test_final$Y)
rf_df <- data.frame(test_new$Date, test_new$Y, rfPredFinal)
colnames(rf_df) <- c('time', 'Y', 'rfPred')

binaryPred.rf.final = ifelse(rfPredFinal > 0,1,0)
binaryPredSig.rf.final = ifelse(rfPredFinal > 0 , 1, -1)
binaryTest.rf = ifelse(test_boost$Y > 0,1,0)

confusionMatrix(as.factor(binaryPred.rf.final),as.factor(binaryTest.rf))

#construction of ROC curve

pred = prediction(binaryPred.rf,binaryTest)
roc.perf = performance(pred,measure = 'tpr',x.measure = 'fpr')
plot(roc.perf)
abline(a=0,b=1)

auc.perf = performance(pred, measure = "auc")
auc.perf@y.values
```

LSTM Procedures

```
#we define a lag of 1
dLag = 1
#define batch size, must be a factor of the lagged train/test length
batch.size = 1
#now basically want everything but the close diff, think a=
x.train = array(
  data = as.matrix(scale(train_boost[,-1])),
  dim = c(nrow(train_boost),1,51))
y.train = array(data = scale(train_boost$Y), dim = c(nrow(train_boost), 1))

x.test = array(
  data = as.matrix(scale(test_boost[,-1])),
  dim = c(nrow(test_boost),1,51))
y.test = array(data = scale(test_boost$Y), dim = c(nrow(test_boost), 1))

#keras_model_sequential construction
set.seed(123)
model <- keras_model_sequential()
model %>%
 layer_lstm(units = 100,
 input_shape = c(dLag, 51),
 batch_size = batch.size,
 return_sequences = TRUE,
 stateful = TRUE, activation = 'tanh') %>%
 layer_dropout(rate = 0.3) %>%
 layer_lstm(units = 50,
 return_sequences = FALSE,
 stateful = TRUE, activation = 'tanh') %>%
 layer_dropout(rate = 0.2) %>%
```

```r
 layer_dense(units = 1, activation = 'linear')

model %>%
 compile(loss = 'mae', optimizer = 'adam')

set.seed(123)
for(i in 1:5){
 model %>% fit(x = x.train,
 y = y.train,
 batch_size = batch.size,
 epochs = 1,
 verbose = 0,
 shuffle = FALSE)
 model %>% reset_states()
}

set.seed(123)
pred_out <- model %>% predict(x.test, batch_size = batch.size)
model %>% evaluate(x.test, y.test,verbose = 0, batch_size = batch.size)

#discretizing predictions
scaleFactor = scale(test_boost$Y)
unscaled_pred = unscale(pred_out,scaleFactor)
binaryPred = ifelse(unscaled_pred > 0,1,0)
binaryPredSig = ifelse(unscaled_pred > 0,1,-1)
binaryTest = ifelse(y.test > 0,1,0)

diffPred = diffinv(unscaled_pred,1) + ethereum$Close[n]

L = length(unscaled_pred)
predictionsR = numeric(L)

for(i in 1:L){

     # invert scaling
     # invert differencing
     yhat  = unscaled_pred[i] + ethereum$Close[n-1+i]
     # store
     predictionsR[i] <- yhat
}

resid = unscaled_pred - y.test

confusionMatrix(as.factor(binaryPred),as.factor(binaryTest))

#Calculating Returns:
#simulates buying or selling (long/short strat)
returnStrat = binaryPredSig * test_boost$Y
#simulates long-only strat
returnLong = binaryPred * test_boost$Y
#control returns
agReturnsStrat = cumsum(returnStrat)
agReturnsLong = cumsum(returnLong)
Returns = cumsum(test_boost$Y)
agDF = data.frame(Returns,agReturnsStrat,agReturnsLong)
DaysElapsed = 1:410
```

```r
#plotting returns
plot_ly(agDF, x = DaysElapsed, y = ~Returns, type = "scatter", mode = "markers") %>%
 add_trace(y = agReturnsStrat, x = 1:410, name = "Long/Short", mode = "lines")%>%
  add_trace(y = agReturnsLong, x = 1:410, name = "Long Only", mode = "lines")%>%
  add_trace(y = ~Returns, x = 1:410, name = "Index", mode = "lines")
```

ARIMA Procedures

```r
eth_ts = ts(ethereum$Close, start = c(2016,5,27), frequency = 365)
eth_diff_ts = ts(ethereum$closeDiff, start = c(2016,5,27), frequency = 365)
plot(eth_ts)
plot(eth_diff_ts)
components.ts = decompose(eth_ts)
components.diff.ts = decompose(eth_diff_ts)
plot(components.ts)
plot(components.diff.ts)
#stationarize
library("fUnitRoots")
urkpssTest(eth_ts, type = c("tau"), lags = c("short"),use.lag = NULL, doplot = TRUE)
ndiffs(eth_ts)
ndiffs(eth_diff_ts)
#train/test split
arima_data = arima_data[, c('closeDiff', 'transaction_count', 'current_supply', 'Volume.ETH', 'Volume.USD')
N = nrow(arima_data)
n = round(N * 0.7, digits = 0)
train_arima = arima_data[1:n,]
test_arima = arima_data[(n+1):N,]
#create ts object
train_diff_ts = ts(train_arima$closeDiff, start = decimal_date(as.Date("2016-05-27")), frequency = 365)
train_arima = as.matrix(train_arima[, c('transaction_count', 'current_supply', 'Volume.ETH', 'Volume.USD')]
eth_arima = auto.arima(train_diff_ts, trace=TRUE, xreg= train_arima)
#predictions
test_arima = test_arima[, c('transaction_count', 'current_supply', 'Volume.ETH', 'Volume.USD')]
futurVal = forecast(eth_arima,h=410, level=c(99.5), xreg = as.matrix(test_arima))
plot(futurVal, main="Arima Price Difference Forecast")
#predictions and returns
binaryPred_arima = ifelse(futurVal$mean > 0,1,0)
binaryPredSig_arima = ifelse(futurVal$mean > 0,1,-1)
binaryTest = ifelse(test_new$closeDiff > 0,1,0)

confusionMatrix(as.factor(binaryPred_arima),as.factor(binaryTest))

returnStrat_arima = binaryPred_arima * test_new$closeDiff
returnStrat_short_arima = binaryPredSig_arima * test_new$closeDiff

sum(returnStrat_arima)
sum(returnStrat_short_arima)
sum(test_new$closeDiff)

agReturnsLong_arima = cumsum(returnStrat_arima)
agReturnsControl = cumsum(test_new$closeDiff)
agReturnsStrat_arima = cumsum(returnStrat_short_arima)

agDF = data.frame(agReturnsControl,agReturnsLong_arima, agReturnsStrat_arima)
```

```
head(agDF)
#plotting returns
plot(1:410,agReturnsControl)
plot(1:410,agReturnsLong_arima)
plot(1:410,agReturnsStrat_arima)

arima_returns = plot_ly(agDF, x = ~1:410, y = ~agReturnsControl, type = "scatter", mode = "markers", name="
  add_trace(y = agReturnsStrat_arima, x = 1:410, name = "Long/Short", mode = "lines")%>%
  add_trace(y = agReturnsLong_arima, x = 1:410, name = "Long Only", mode = "lines")

arima_returns <- arima_returns %>% layout(xaxis = list(title="Time"), yaxis = list(title="Returns"))
arima_returns
#roc curve
library(ROCR)
plot(c(0,1),c(0,1),xlab='FPR',ylab='TPR',main="ROC curve",cex.lab=1,type="n")
pred = prediction(as.vector(binaryPred_arima), as.vector(binaryTest))
perf = performance(pred, measure = "tpr", x.measure = "fpr") #calculate performance in lines(perf@x.values
aucnow=performance(pred, measure = "auc")
lines(perf@x.values[[1]], perf@y.values[[1]])
abline(0,1,lty=2)
```