Predicting Tomorrow

APPLYING MACHINE LEARNING TO TRADE BITCOIN

Aaron Childress

INVESTMENT THESIS | PANTERA SUBMISSION

OBJECTIVE

Develop a bitcoin trading strategy with accuracy score significantly higher than can be attributed to random chance. The corollary is the trading strategy should generate alpha with respect to a HODL strategy.

MOTIVATION

Bitcoin price is highly volatile; a trading strategy that can consistently limit downside deviation could generate outsized returns.

DEFINING THE STRATEGY

Build machine learning processes to predict whether tomorrow's return (t + 1) will be positive or negative. The strategy is simple:

- ONE security: bitcoin is the asset; bitcoin return is the benchmark
- ONE binary signal: 0 = short, +1 = long (buy or hold depending on current state)
- ONE-step-ahead: forecast only tomorrow's return

The underlying assumption is that the *full position* is either bought/held or sold at the close of each day based on trade signal.

FEATURE ENGINEERING

Features can be grouped into three categories – lags, technical indicators, and cointegrated assets. There are a few features outside of these categories included in non-linear models, but they turned out to be relatively unimportant (see Figure 4).

Lags. The acf and pacf plot on log-returns showed significant lags at 6 and 10 at the 95% confidence level; however, there was no clear AR or MA pattern, so I grid searched ARMA orders to select the order with the lowest AIC score, ARMA(8,5).

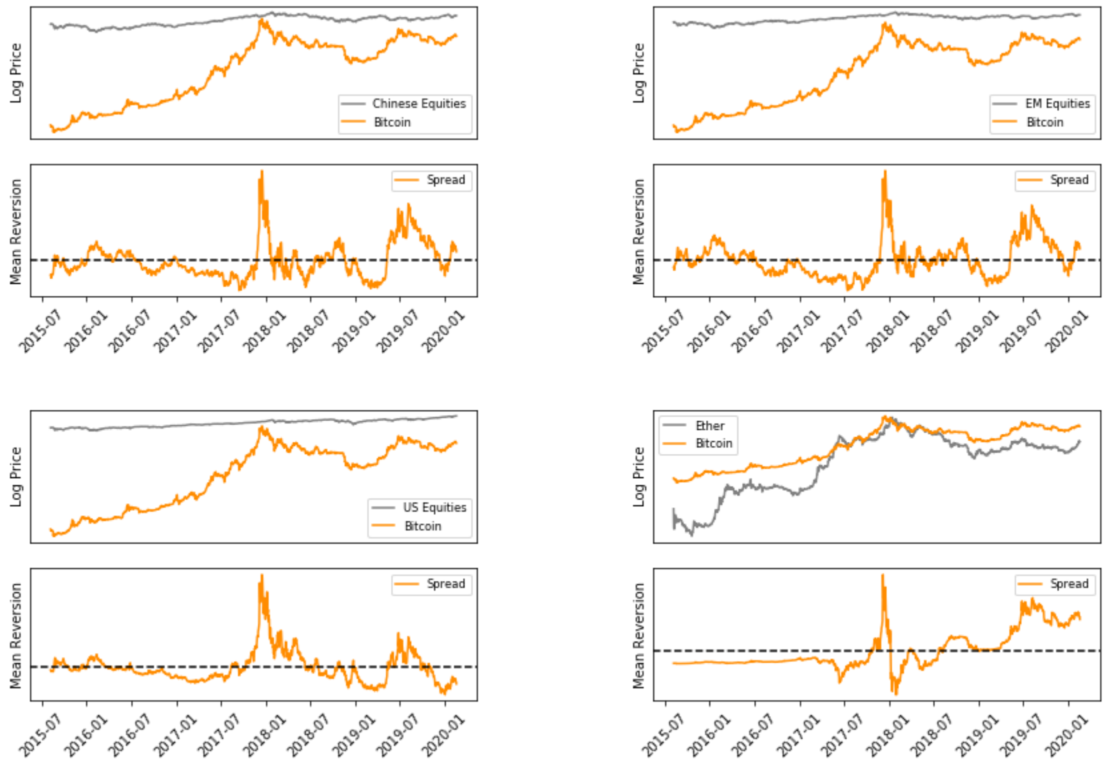
Technicals. I included 7, 14, 20, and 50-day SMA and RSI features.

Cointegrated Assets. Even if two prices follow a random walk, it's possible that their linear combination does not follow a random walk. Moreover, even though the individual prices themselves may not be forecastable, the linear combination is. I tested for cointegration of various asset prices with bitcoin prices[1].

Each plot in Figure 1 shows clear mean reversion. This implies that there is an economic relationship between these assets and bitcoin such that including them in the model should improve the forecast. Last, looking at Chinese equities and emerging market equities, the spread to bitcoin is almost identical. Because Chinese equities and emerging market equities are strongly correlated and because China accounts for 2/3rd of bitcoin mining alone, I used Chinese equities only (dropping EM equities)

[1] I tested for cointegration in two parts: First, I regressed bitcoin prices on other asset prices to get the slope coefficient (m). Second, I ran the Augmented Dickey Fuller test on the linear combination of each series pair to determine whether it's a random walk. The null hypothesis is no cointegration. I set a 10% significance level for rejecting the null. I included assets that passed the test in Figure 1. The top chart is log-price of the assets and the bottom is the spread (bitcoin price – m*asset price).

Figure 1: Cointegration Plots



Note: Spread is mean subtracted (zero centered)

MODEL DEVELOPMENT

Daily returns are challenging to predict, so I applied some custom solutions while taking care to properly handle time series data. The core of my model is an iterative process. I take one-step forecasts when predicting on the validation set. At each iteration, I fit the model on *all* data from time 0 to time t to predict time t + 1. Everyday, the train set expands by one day and I *only* forecast the next day. From what I can tell, scikit-learn has no built-in methods for this, so I wrote a program to do it. Every model discussed below implements this program when predicting on the validation set.

LINEAR MODELS

Step 1: Transform Target

I transformed bitcoin price to log returns to achieve relative stationarity

Step 2: Model Selection

Base ARMA model (8,5) was selected because of lowest AIC.

Step 3: Fit base ARMA model with no exogenous variables on train set

Autocorrelation. Ljung-Box (Q) Null Hypothesis is there are no correlations in the residuals. With Prob(Q) 0.62, FTR null – no issues with autocorrelation.

Heteroskedasticity. Jarque-Bera (JB) Null Hypothesis is the residuals are normally distributed. With Prob(JB) 0.0, reject null. Residuals are not normally distributed. Heteroskedasticity in the error terms suggests that there are exogenous variables that should be added (as expected).

Step 4: Predict base ARMA with no exogenous variables on train data to establish baseline

I made one-step-ahead forecasts over the last 90 days of training data. Scikit-learn has get_prediction method for this. However, because ARIMA are regression models and my goal is classification, I built functions to map the output[2].

Accuracy for base ARMA model on last 90 days of train data was 47%.

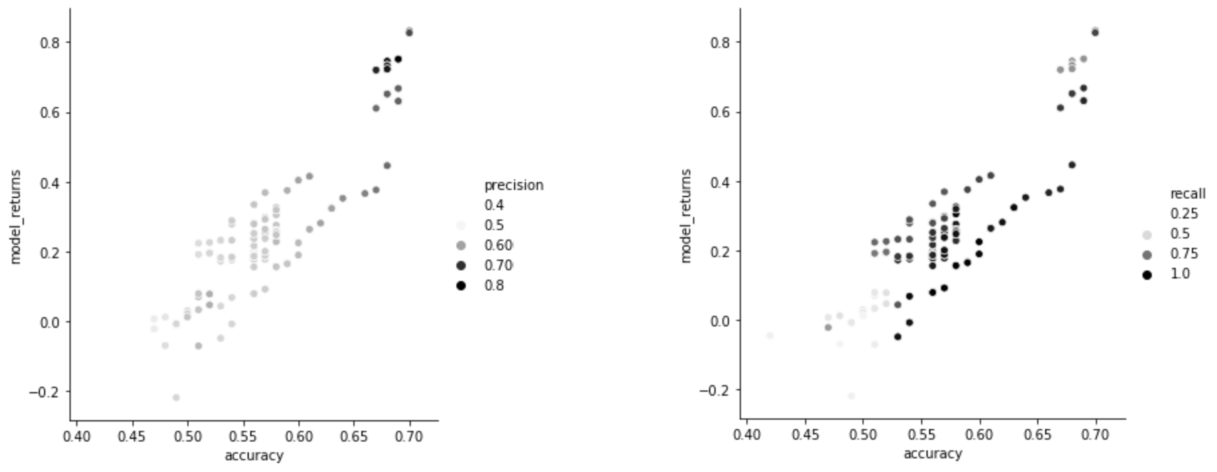
Step 5: Predict base ARMA with no exogenous variables on test set

Accuracy score was 52%. Train and test accuracy scores are not significantly different from random chance.

Step 6: Repeat steps 3-5 with exogenous variables

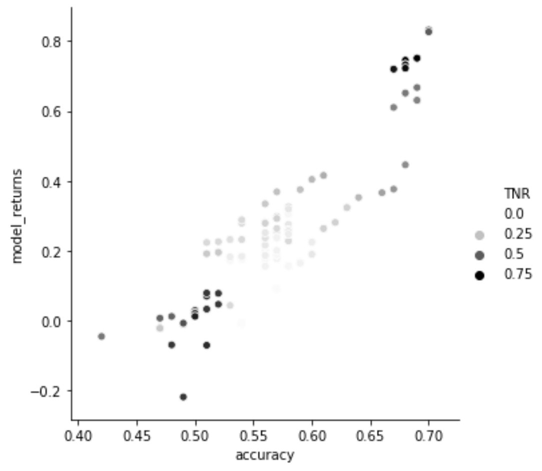
Including exogenous variables improved the accuracy score substantially from the base model. On the train set, accuracy increased from 47% for the base model to as high as 70% for some multivariate models. Features included SMAs, RSIs, and lag 1s of cointegrated pairs. I ran all combinations of 3, 2, and 1 feature ARMAX models resulting in ~230 models (see Figure 2). High precision correlated with high accuracy, high recall correlated with moderate accuracy, and high true negative rate resembled a barbell, with high TNR resulting in high and low accuracy. For the voting classifier I optimized for high accuracy scores with either high precision or high TNR.

Figure 2: ARMAX Model Distribution



[2] prediction_quality(): takes the model one-step-ahead predictions and simulates a trading strategy. Generates outcomes for confusion matrix as well as strategy returns.

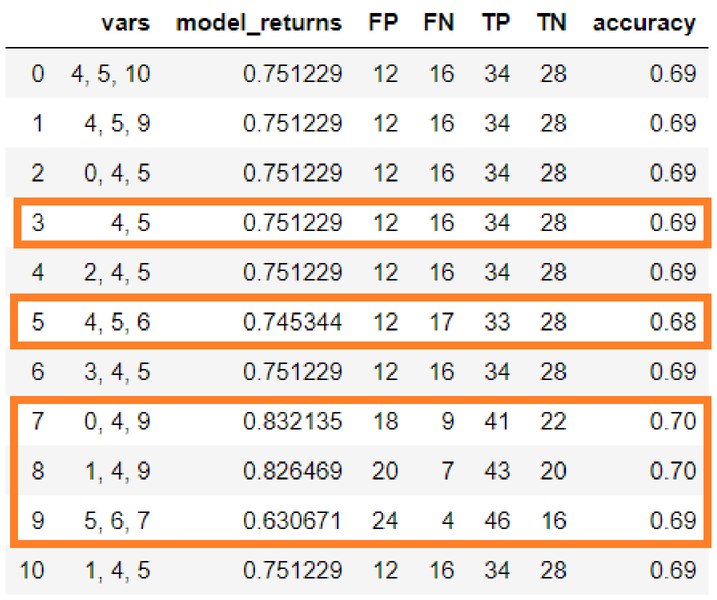
conf_matrix(): takes output from prediction_quality and returns confusion matrix and accuracy score.



Step 7: Build Voting Classifier

Models showing redundancy all include vars 4 and 5. To avoid skewing the weights, I dropped all but the bivariate 4,5 model. Resultantly, the voting classifier is comprised of five models. With varied information from each model, the idea is that the voting classifier will generalize to unseen data better than any one individual model.

Figure 3: Voting Classifier Model Selection



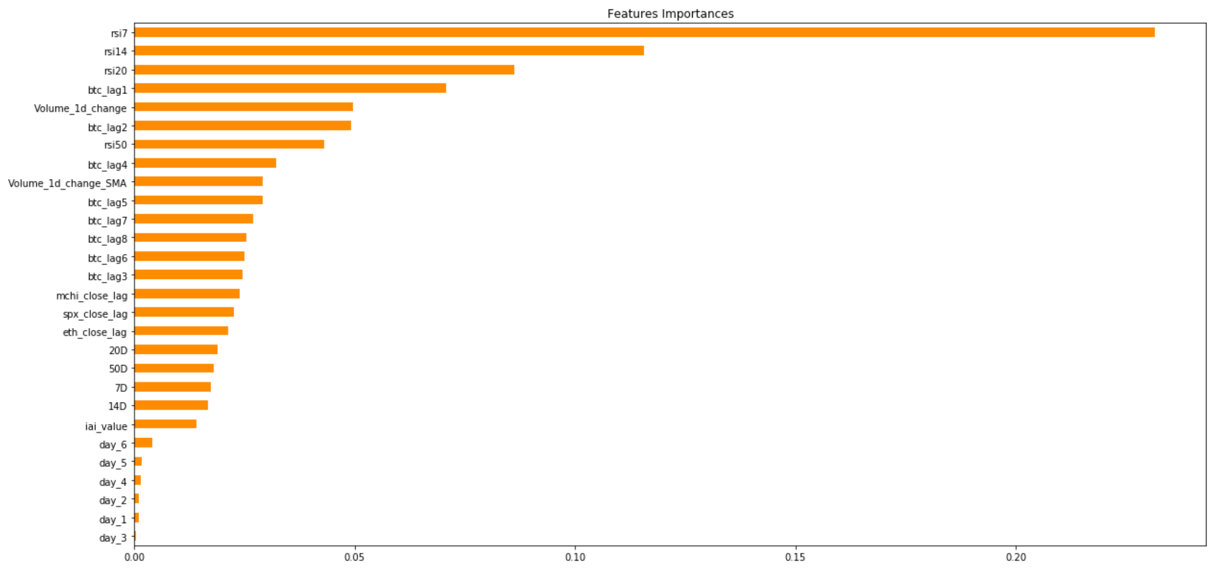
	vars	model_returns	FP	FN	TP	TN	accuracy
0	4, 5, 10	0.751229	12	16	34	28	0.69
1	4, 5, 9	0.751229	12	16	34	28	0.69
2	0, 4, 5	0.751229	12	16	34	28	0.69
3	4, 5	0.751229	12	16	34	28	0.69
4	2, 4, 5	0.751229	12	16	34	28	0.69
5	4, 5, 6	0.745344	12	17	33	28	0.68
6	3, 4, 5	0.751229	12	16	34	28	0.69
7	0, 4, 9	0.832135	18	9	41	22	0.70
8	1, 4, 9	0.826469	20	7	43	20	0.70
9	5, 6, 7	0.630671	24	4	46	16	0.69
10	1, 4, 5	0.751229	12	16	34	28	0.69

Note: result of 90 day prediction on train set

NON-LINEAR MODELS

The process is similar to linear models – fit and predict on train, then predict on test data. I started with an untuned Random Forest Classifier and tuned the hyperparameters. Figure 4 shows the relative information gain from the features.

Figure 4: Random Forest Feature Importance



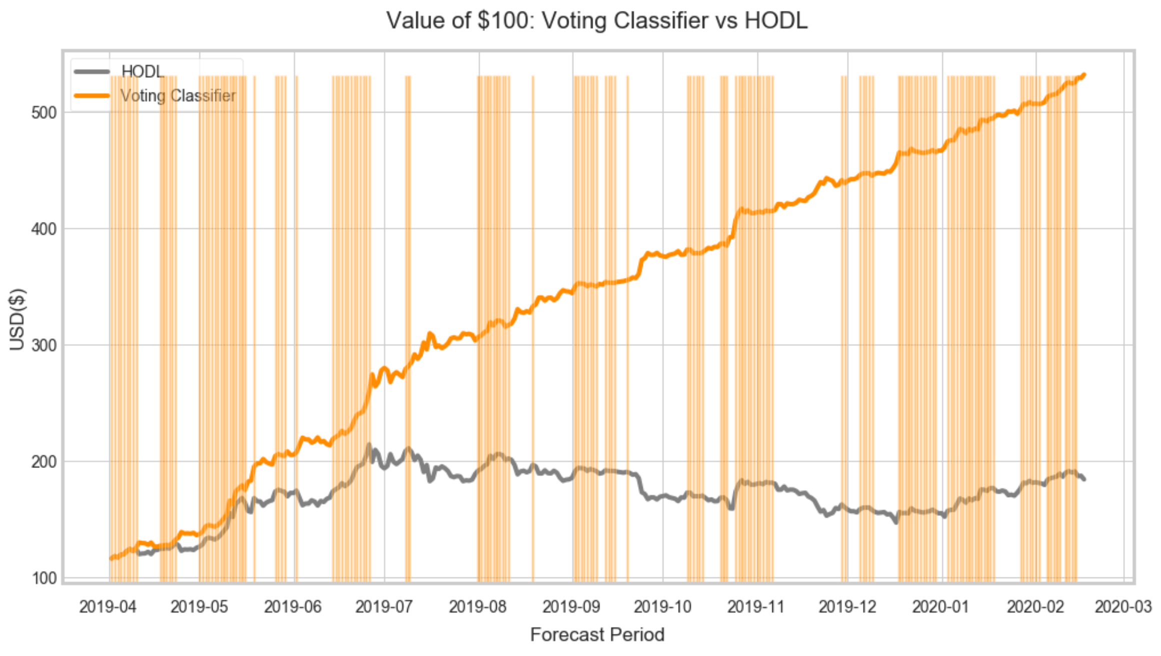
Note: asset_close_lag refers to the lag 1 cointegration of the particular asset. IAI is investopedia anxiety index which is a sentiment analysis. Day is categorical weekday variables.

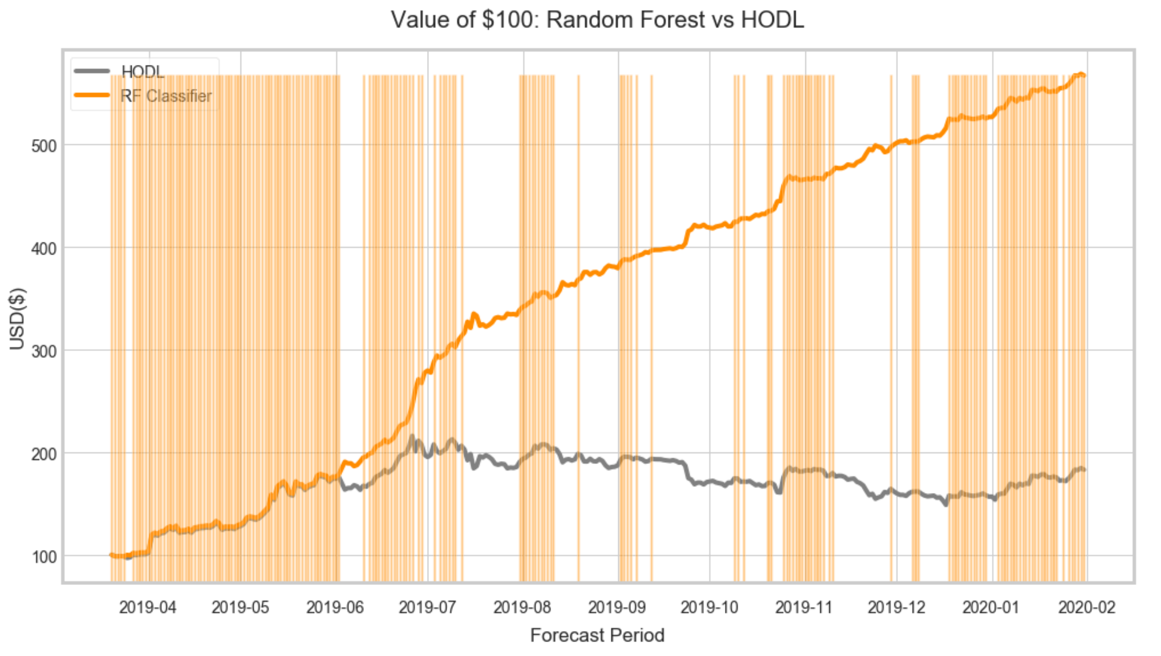
RESULTS ANALYSIS

SIGNAL VS BENCHMARK – TRACKING BUY AND SELL DECISIONS

The orange vertical lines in Figure 5 track the buy signals and the whitespace tracks the sell signals of the respective models. It's clear that the buy signals are most prevalent during bitcoin uptrends and sell signals most prevalent during bitcoin downtrends.

Figure 5: Trading Signals and Cumulative Return

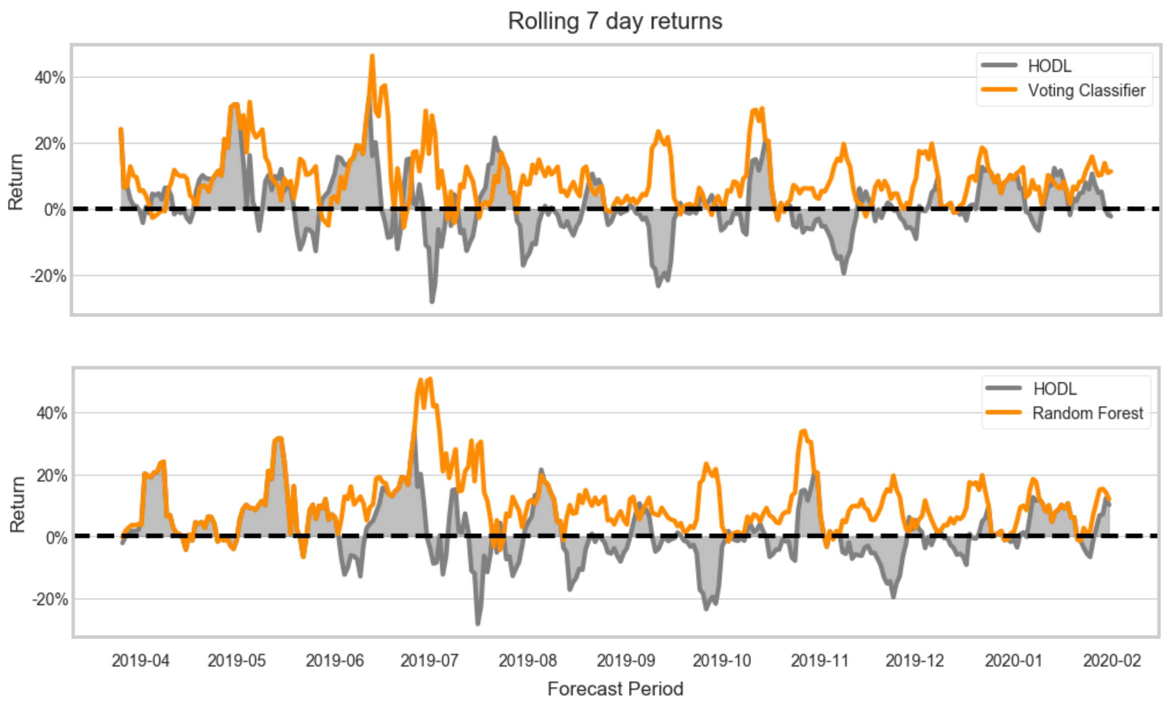




ROLLING RETURNS

Looking at rolling 7-day returns shows why the models outperform – they limit exposure to drawdowns.

Figure 6: Rolling Returns

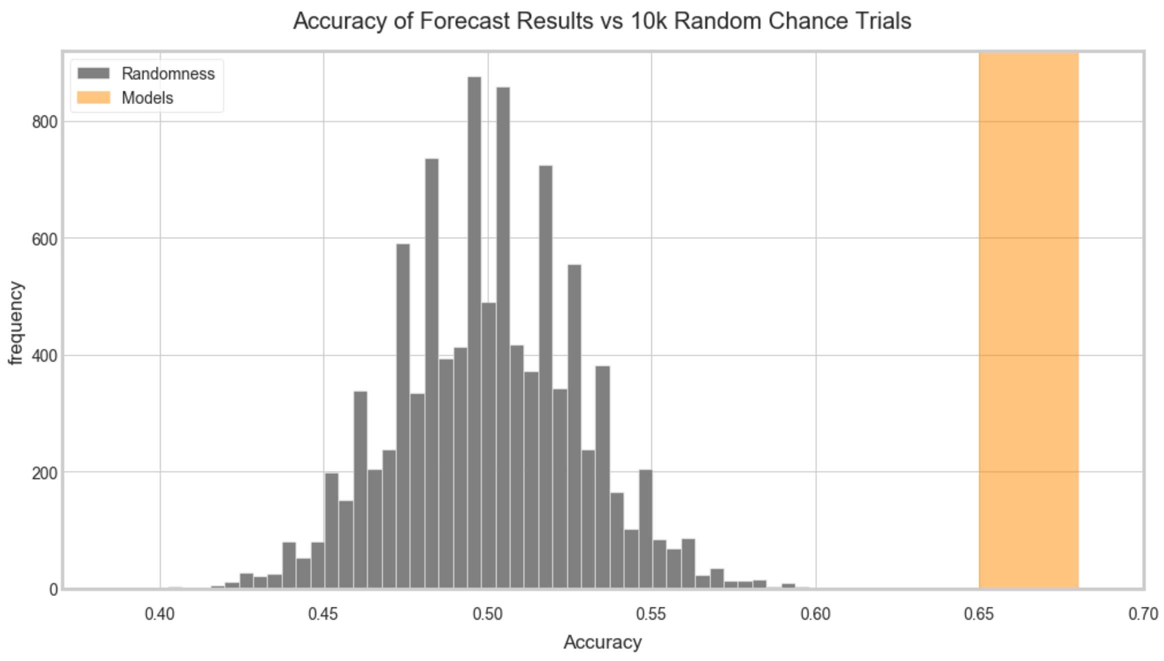


Note: forecast timeline varies slightly between models due to data loss from transformation of certain x variables

RANDOMNESS SIMULATION

The figure below shows the result of 10,000 simulations of random guess trials over the forecast period. I can reject the null hypothesis that model accuracy is not significantly higher than random chance.

Figure 7: Binomial p(0.5) Forecasts vs Trading Models



UNPACKING ACCURACY SCORES

Voting Classifier	Predicted Negative	Predicted Positive
Actual Negative	33%	15%
Actual Positive	20%	32%

The ARIMAX-based ensemble performed better at minimizing False Positives. This is ideal if the goal is to minimize loss.

65%
Total Accuracy

RF Classifier	Predicted Negative	Predicted Positive
Actual Negative	30%	19%
Actual Positive	13%	38%

The tree-based ensemble performed better at minimizing False Negatives. This is ideal if the goal is to maximize appreciation.

68%
Total Accuracy

FURTHER WORK

Given more time I would explore the following:

More features: test cointegration between bitcoin price and network value using Metcalfe's law with a few exponent variants

More combinations: ARIMAX combinations of >3 x variables

Additional Models: gradient boosting, KNN, LSTM, VAR… stack combinations of these with my current ARMAX and RF models.