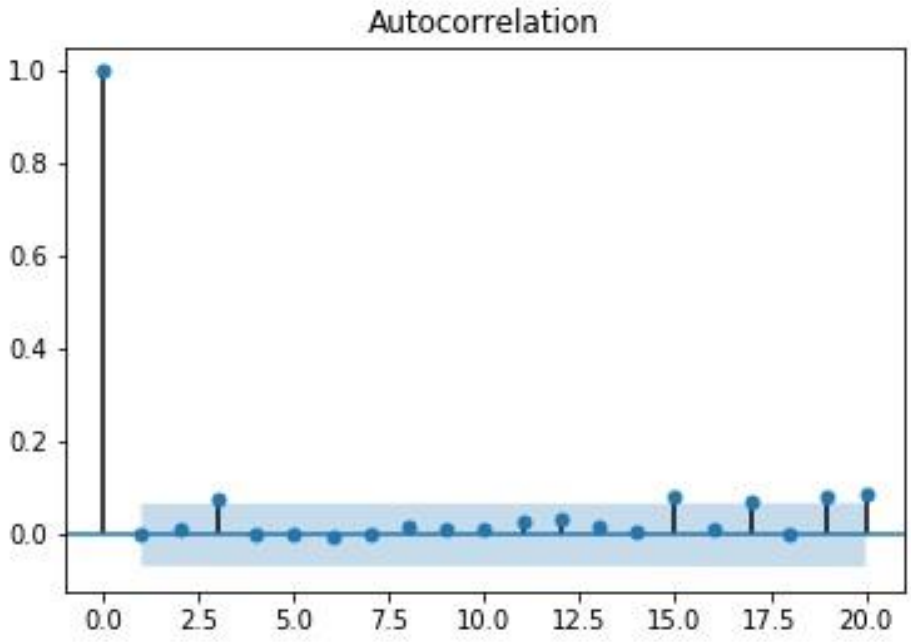
Machine Learning for Cryptocurrencies

Peter Goodridge

Time series models are different from other modelling techniques in that they seek their prediction using little to no domain information. The chief input to their formulae is the past behavior of the target variable. In the context of asset pricing, this means using a series of past returns to predict future returns. This analysis will explore the efficacy of autoregressive techniques and Random Forest in forecasting cryptocurrency prices.

Autocorrelation models have, for many years, been the standard for time series prediction. They rely on a correlation between the current value and lagged values. In its simplest form, the autoregressive(1) or AR(1), the response variable is predicted by the equation $y_t = \mu + \phi_t y_{t-1}$ where phi is a coefficient that represents the correlation between y and its lagged value. The ARMA model adds an additional term that signifies the difference between the previous value and the mean. The ARMA(1,1) model is represented by the equation $y_t = u + \phi_t y_{t-1} + \theta e_{t-1}$. A further generalization, ARIMA, adds a term for the difference between the current and most recent value.[i]

There is no guarantee of any correlation between lagged returns and the most recent return. In this plot of autocorrelation for Ethereum, we only see one significant lagged variable that could be usable in practice, the lag 3 value.

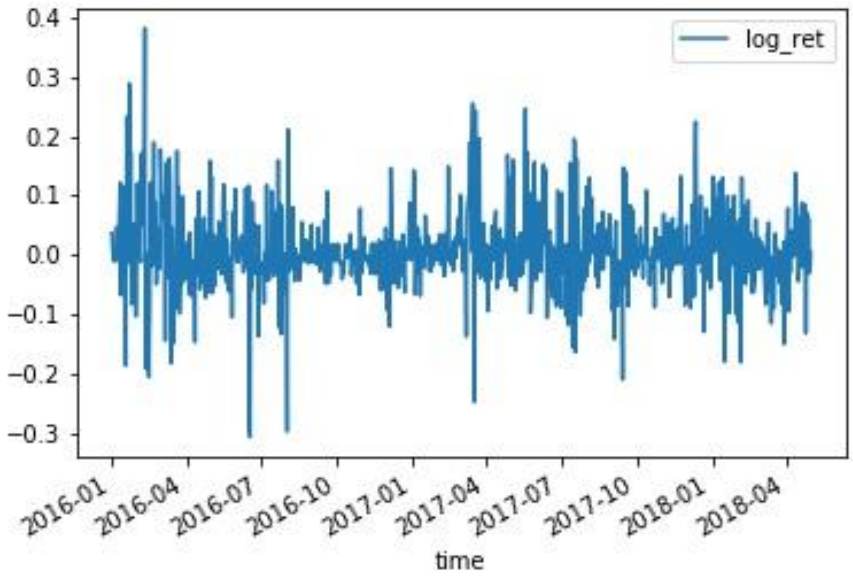


Even this value is could be a type 1 error, as the autocorrelation should quadratically degrade for an AR model. For the MA model, autocorrelation has a more abrupt degradation. To determine the final parameters for the model, the p values of the parameters, likelihood function scores, and error on testing data should be considered. Even with the optimal parameters, (0,1,1), the ARIMA model had significant error. The ratio of $(\text{actual returns} - \text{predicted returns})^2$ to $(actual\ returns - \mu)^2$ was just over 1 for the testing period of the last 100 days. The ARIMA model explained very little of the variance in actual returns.

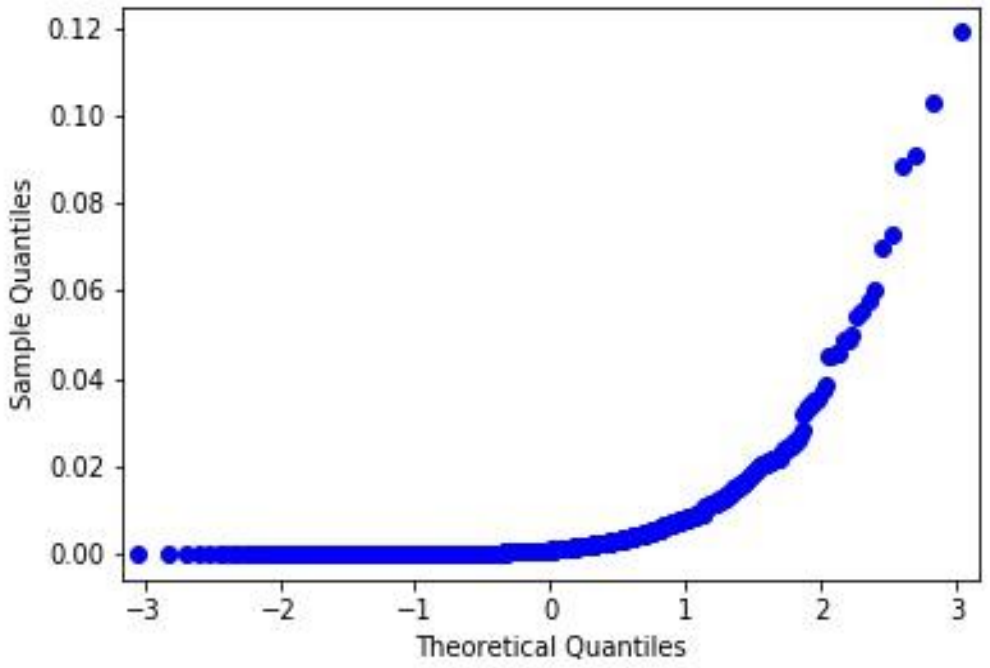
One of the shortcomings of the ARIMA model is its assumption of constant variance. If the target variable's variance increases during a time of uncertainty, one can expect returns to deviate further from the mean. This can cause additional error in the ARIMA model. The GARCH model accounts for these periods of additional variance. GARCH specifies that the

response variable is proportional to the variance at a given time. The variance term is conditioned on lagged values of the variance and lagged values of deviations from the mean variance. This makes the variance portion of the GARCH model much like an ARIMA model, but instead on the variance.[ii]

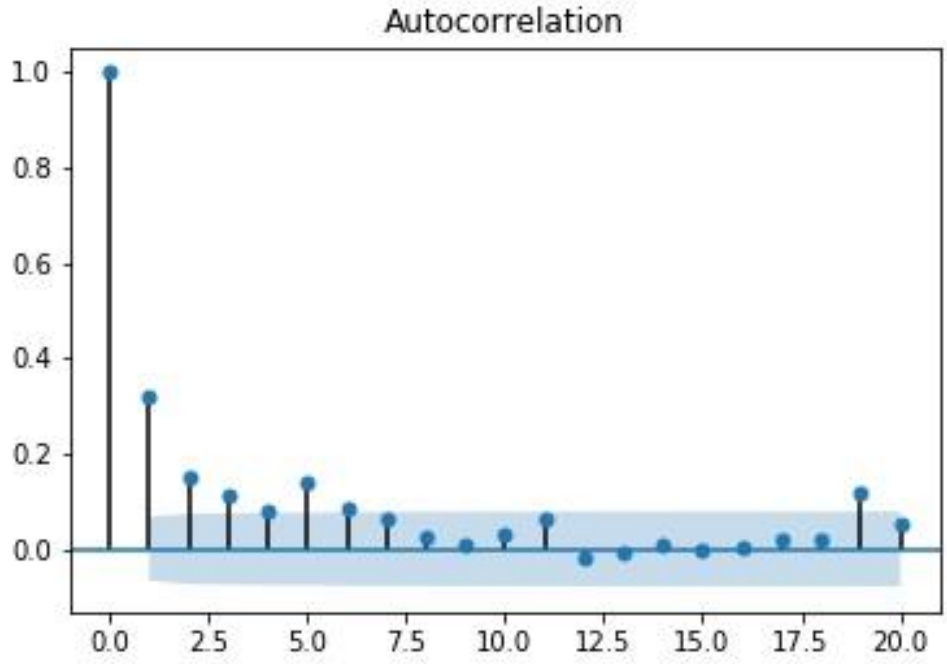
The data for both Bitcoin and Ethereum appears to follow this conditionally heteroskedastic pattern.



A more concrete diagnostic for heteroskedasticity is the normality the square of the residuals from the ARIMA model. The QQ plot can be used as a first line measure, with statistical tests for normality determining borderline cases. We see that the dataset does appear to be candidate for the GARCH model.

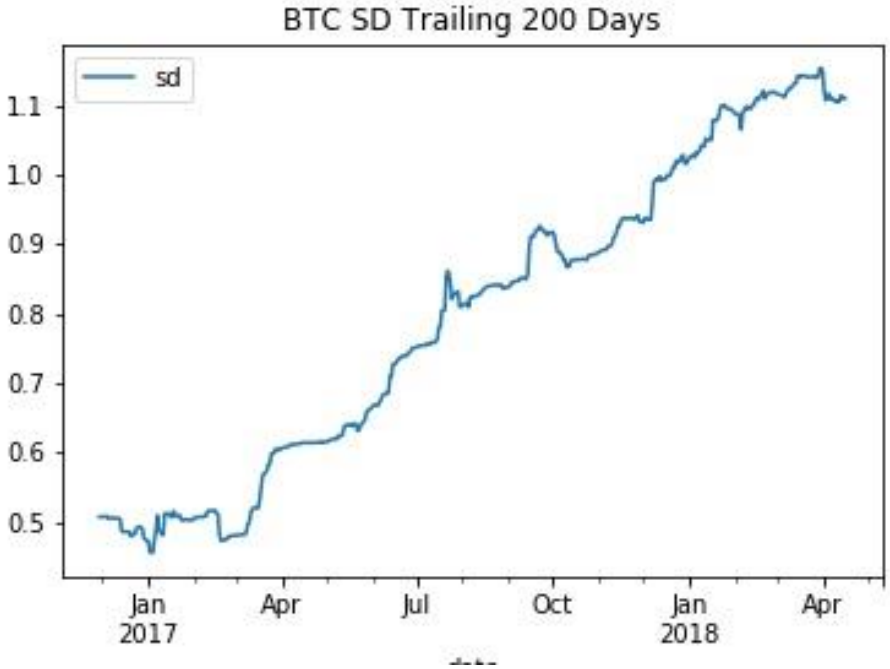


There is clear autocorrelation between the squares of the residuals.



The ratio $(\text{actual returns} - \text{predicted returns})^2$ to $(actual\ returns - \mu)^2$ (squared error to variance) was under 1 for both Bitcoin and Ethereum, indicating that the model was able to explain some of the variance. The exact ratios were .978 for ETH and .995 for BTC. The GARCH improved the prediction results in comparison to ARIMA but was still unable to make predictions accurate enough to use for trading.

GARCH assumes the unconditional variance to be constant. As one can see, the rolling variance of bitcoin returns appears to be increasing.



Both the ARIMA and GARCH models must also seek linear relations, using linear algebra or a numerical estimation thereof to derive their parameters. Even the terms in the model matrix are linear in ARIMA, correlation and deviation from the mean, and quadratic for GARCH, variance and squared residual[iii]. This keeps overfitting in check, but also increases bias[iv]. Finally, they

are unable to use predictive variables other than the time series in question. We will next explore a method that does not have these limitations.

Random Forest is a machine learning algorithm that uses a collection of decision trees to determine the predicted class or numeric value of an observation. As such, it is well suited towards both classification and regression. When used for regression, it is considered a nonparametric model because it does not use coefficients for the predictor variables to estimate the response variable. Additionally, it makes no assumption about the distribution of the response variable, the distribution of the predictor variables, or the order of their relationship. Each of these trees are constructed with bootstrapped samples. Rather than choosing the "best" variable to split branches on, the variables are also chosen at random from a subset of variables chosen for each tree. A class or value is determined by running the observation down each tree and taking the average. Each tree is unique, so the model is more robust to overfitting.[v]

Analogous to the autoregressive models, lagged values of log returns were chosen as the predictors. However, this algorithm, may be able to identify nonlinear trends, such as two days with gains above a threshold, followed by a weak down day will usually be proceeded by a day with strong gains, or any arbitrary permutation of that type. This contrasts with the autoregressive model, that was only able to relate the current value to a linear combination of past values and their deviations from the mean.

The Random Forest's ability to use additional features in prediction was also exploited. Trends involving momentum seemed like a strong possibility, so the streak of up/down days was added as a feature. It is common thinking in securities trading that gains or losses on high volume indicate investor exuberance towards that trend, so more days like that can be expected in the future. As such a measure of volume was also added.

Since 1/1/2017, Bitcoin and Ethereum have been somewhat cointegrated. The p value of the Dickey-Fuller test for stationarity in the difference between their values is .06. If the difference exceeds a certain level, one should be expected to catchup to the other. This proved to be a feature used by the model.

No feature related to trade volume proved to be useful. The variable importance for the remaining features used in Random Forest were all similar, approximately .15 for each.

The ratio of squared error to varinace was 1.07 for bitcoin and .929 for Ethereum. These results were comparable to those from GARCH. However, given the variance in the estimate against the test set, combined with trading costs, still not suitable to employed in a trading a strategy.

The poor results on the test set contrasts with a high R squared on the training data, indicating some overfitting. As mentioned earlier, Random Forest should be robust to overfitting, but is not immune. R squared on the training data was nearly .5 for Ethereum, while R squared was (-.05) on the out-of-bag, or holdover samples used during training (It should be noted that R squared is the complement of the metric that I have been using for model evaluation.)This further reinforces the assessment that this model is not yet suitable for live trading.

Train-test splits for time series pose several challenges. With datasets that are time invariant, any train test split would prove useful, for instance training on highest 80% of observations by index and testing on the 20% of lowest. This is because the observations are independent. This is less reasonable for time series datasets, where observations near each other are correlated. For such a dataset, the best train-test splits should, optimally, preserve the order of the observations.[vi]

In terms of convenience, Random Forest with the sklearn implementation makes train test splits easier. Build in methods can be easily used to conduct the time series cross validation. The forecast method in the ARCH package must be used with all the data that was used to train the

model. New data cannot be passed to a fitted model. Predicting any further than the most recent observation is impractical due to the wide confidence intervals. The best alternative is using the leave one out method using the last n samples. This is convenient and still allows for some variation in the training set.

Because Random Forest uses bootstrap samples from the data that is passed to it, one is not able completely control what observations are used to build the ensemble of decision trees.[vii] For certain time series, this would prevent Random Forest from being viable. Because returns where used in this instance, instead of price, and because their process is mostly stationary, Random Forest remains viable. Random Forest even insures that all market conditions present in the dataset are used to both train and test the model, enhancing its ability to generalize.

Random Forest does not specifically account for localized or conditional changes in variance as GARCH does. Variance could be computed as a rolling window, but this would cut down on the number of observations in the training set by the size of the window. To implement conditional variance in the same manner as GARCH, the user of Random Forest would be required performance the calculations themselves. This is increases the possibility of an incorrectly specified model but could boost performance.

In terms of computational cost, GARCH is the superior of the two. Random Forest takes about 6 seconds in the implementation used for this analysis, while GARCH takes 2.4 seconds. This makes it easier to implement real time. Both models are still viable in real time implementation because they can be trained offline and saved for use in production.

GARCH is an easy to implement time series model that has been in use for many years. By default, it accounts for shocks in the market and autocorrelation. It is, however, not possible to customize with additional predictor variables, so its performance on these datasets was somewhat lacking. Random Forest is highly customizable and robust enough to overfitting that

it offers comparable performance in predicting cryptocurrency returns with even with unoptimized features. Neither model performed well enough to warrant employing in an algorithmic trading strategy, but Random Forest shows the most promise given the possibility for customization.

---

[i] Nau, Robert "Statistical forecasting: notes on regression and time series analysis", Duke University, 14 December, 2017

[ii] Fryzlewicz, Piotr "Lecture notes: Financial time series, ARCH and GARCH models", University of Bristol, 18 July, 2007.

[iii] Fryzlewicz, Piotr

[iv] Richert, Willi and Coelho, Luis Pedro "Buidling Machine Learning Systems with Python", Packt Publishing, July 2013 p 30

[v] Cutler, Adele and Breiman, Leo "Random Forests", UC Berkeley, unknown date

[vi] Brownlee, Jason "How To Backtest Machine Learning Models for Time Series Forecasting", Machine Learning Mastery, 19 December, 2016

[vii] Cutler, Adele and Breiman, Leo