Kyle Weng

Ge/Ay 117

Winter 2022

Methods

Note for Prof. Knutson and TA's: Sorry for turning this in so late; I spent too much time re-running my chains and trying to get them to converge properly. I probably should've accepted earlier that this was basically impossible because of the nature of my task/data in the first place.

I. Introduction

Cryptocurrency has existed since 2009—the year Bitcoin was launched. However, it largely remained out of the public eye until 2017, when multiple major currencies rocketed to new price highs. That boom was short-lived, and public interest waned after a few months. That said, Bitcoin, Ethereum, and other cryptocurrencies experienced a renewed wave of interest in 2020 and 2021, riding a rising tide that also benefitted the US stock market. This spotlight has brought with it traders and institutions alike looking to make a profit. Sophisticated agents frequently use sentiment analysis and other relatively new tools to help inform their trades. Google's publicly available search data is one such potentially useful resource. Here, we examine trends in Google searches for Ethereum and attempt to construct an informative model linking search trends and price signals.

II. Obtaining the Data

The Google search trend data was downloaded from trends.google.com. The Ethereum closing price data was indirectly obtained from Yahoo Finance. Instead of manually downloading five years of price data, we used the yfinance python package to download and store it directly as a Pandas dataframe in our code. Originally, we used the pandas-datareader package, but found that it had a bug that prevented us from accessing the full extent of Yahoo Finance's Ethereum price data.

III. Data Preprocessing

The first issue we ran into was that the two datasets being examined did not quite line up. The closing price data is recorded on a daily basis, while the Google trends search interest data is recorded on a weekly basis. We had to choose between either averaging the daily prices (on a weekly basis) to match the search interest or imputing search interest values to match the daily prices. We opted for the latter choice, as we didn't want to lose information from averaging every seven closing prices into one measurement.

How did we choose what values to impute into the search data? At first glance, it might seem reasonable to interpolate—perhaps linearly—between search interest data points.

However, we decided against this, as it would only be useful for retrospective analyses. It's worth noting that if someone were to actually use this model (or one similar to it) to predict the day's Ethereum closing price from past closing prices and search interest data, they would not have any way of (easily) determining whether search interest would rise, fall, or remain constant in a particular week. Perhaps that data would be more available if we had access to Google's private servers, but we do not. Thus, if the search interest for a particular day was S, we assumed that the search interest for each of the next six days was also S. Afterwards, the Google trends data would provide a fresh measurement, and we would repeat the process.

IV. Measurement Error

Another area of concern was the measurement error. This is especially critical for MCMC methods, as these are necessary for calculating chi-squared values. This data didn't have any actual measurement error, since prices, unlike measurements of temperature, mass, etc., have no associated uncertainty[1]. For the time being, though, we calculated error values by taking the standard deviation of the closing prices in a "sliding window" around a given day. For day n, we would calculate error by taking the standard deviation of the closing prices for days $n-1$, n, and $n+1$. For the first day, we used days $n-1$ and n; for the last day, we used days n and $n+1$.

It's worth noting, however, that we only used these error values (denoted by $\sigma_{calculated}$) for a few initial fits. For every model we tested, we ran at least one version that fitted the error as another parameter (denoted by $\sigma_{parameterized}$). For the linear model, we found that the version that parameterized error produced a better fit than the version that did not. For the models with higher dimensionality, it is not immediately clear if parameterizing error produces a better fit.

We also tested one model which incorporated an error parameter alongside the error values we calculated earlier (from the sliding window), producing $\sigma^2_{combined} = \sigma^2_{calculated} + \sigma^2_{parameterized}$. The intuition behind this model was that we could perhaps combine our quantified notion of "uncertainty" in Ethereum pricing (by using closing prices to approximate volatility) with a parameterized error to allow for better fits.

[1] However, prices for stocks and cryptocurrency (especially the latter) can feel quite uncertain at times, so perhaps there is a way we can incorporate this into our analysis in the future.

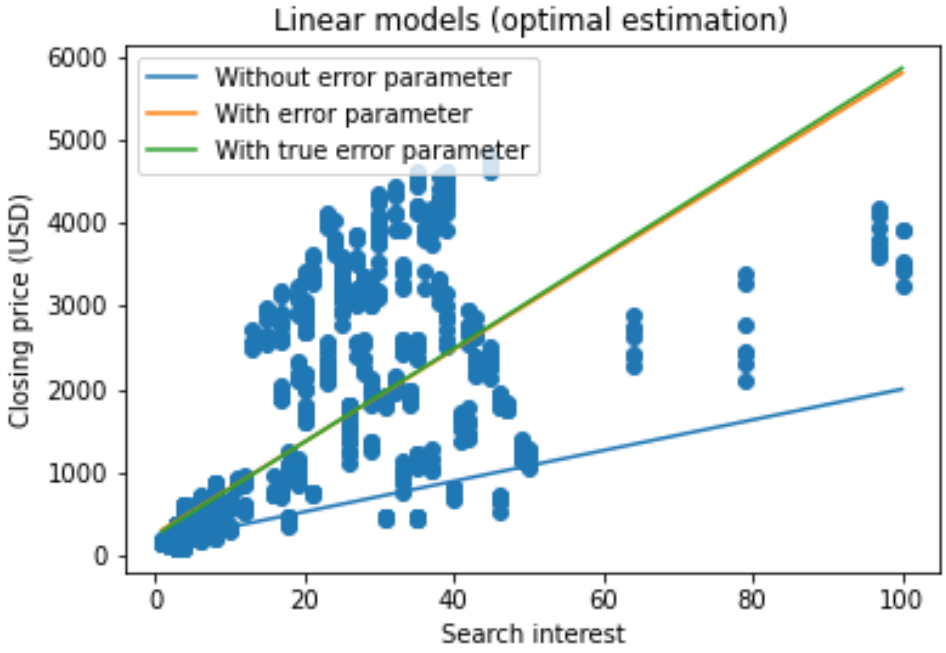

Figure 1. Initial MCMC parameters for our linear models (from optimal estimation). Note that our "combined error" model fits about as well as our parameterized error model.

However, we found that for the linear model, this version ($\sigma_{combined}$) produced virtually identical results to the version that used parameterized error alone ($\sigma_{parameterized}$); this is shown in Fig. 1. Thus, we decided not to test higher dimensional models with "combined error".

V. Model Choice

We used polynomial and sigmoid models. Specifically, we fit the data to the functions:

$$f(x) = a + bx$$

$$g(x) = a + bx + cx^2$$

$$h(x) = a + bx + cx^2 + dx^3$$

$$i(x) = \frac{a}{1 + e^{-c(x-b)}} + d$$

(In these functions, x is a search interest value, and $f(x), g(x), h(x), i(x)$ are corresponding closing prices.) There was no particular intuition behind choosing the linear, quadratic, and cubic models, as we didn't expect any simple function to be able to fit the data well. On the other hand, we added the sigmoid model because we noticed that up to a certain point, there was a positive correlation between the closing price and the search interest (namely from a search interest of 0 to 20); after that, though, there was little to no correlation between them.

A full list of models, best-fit model parameters, and parameter confidence intervals can be found in Section XI.

VI. Log-likelihood Functions

The log-likelihood function used was:

$$L(\{x_k, y_k, \sigma_k\}) = \sum_k ln(\frac{1}{\sigma_k\sqrt{2\pi}}) + \sum_k -\frac{1}{\sigma_k^2 2\pi}(y_k - M(x_k))^2$$

$M(x)$ is any model; for the purposes of this paper, it is always one of the four models specified above. In practice, we also specified the particular parameters of the model (a, b, c, etc.) and the model itself as inputs to our function. We also wrote a slightly modified version for use with versions of models where error was parameterized.

VII. Optimal Estimation

For each of the models, the initial parameter estimates were obtained via optimal estimation. We used scipy.optimize.minimize to numerically find the minimum value of $-L(\{x_k, y_k, \sigma_k\})$ for each model we tested. The corresponding parameters can be found in section XI. For polynomial models with $\sigma_{parameterized}$, we were able to check our maximum likelihood estimate parameters with results from numpy's polyfit function. For the sigmoid model, we wrote a custom sigmoid function and used scipy's curve_fit function. Then, we used the initial estimates as starting points for the MCMC processes. Refer to Figs. 1 (Section IV) and 3, 4, 5 (Section XII) to see optimal estimation results for each of our models.

VIII. MCMC

We used the basic MCMC implementation developed earlier during the class to find posterior distributions for each parameter in each of our models. The main reason for using this implementation was that we were already familiar with its inner workings, since we wrote it ourselves. We tuned our step sizes and number of iterations on a per-model basis to find 68% confidence intervals for our parameters while still maintaining acceptance ratios between 20 and 50% for each chain.[2] We checked burn-in periods and convergence by visually inspecting trace plots for each parameter and chi-squared. Then, we trimmed the burn-in points before calculating the 68% confidence intervals for each parameter[3].

IX. Convergence Issues

Unfortunately, many of our chains did not converge—namely, those that parameterized error. While their χ^2 plots appeared to be *somewhat* stable, their parameter trace plots showed constant fluctuation. This may have been because the initial parameters supplied to the chains

[2] This was impossible for some of our chains, though, because they never converged—see section IX.
[3] These are available in section XI.

were already optimal, resulting in the chain being unable to find any "better" parameters to converge on.

More importantly, however, all of these chains displayed continually increasing error parameters (seen in Figs. _, _, _). Because error was a part of the χ^2 function used in our MCMC implementation[4] but *not* a part of our actual models, the chain could safely increase the error parameter without penalizing the model's actual quality of fit. We can briefly refer to the χ^2 equation to clarify:

$$\chi^2 = \frac{(M(x) - y)^2}{\sigma^2}$$

Because none of our models $M(x)$ contain σ, the chain can freely increase σ to artificially decrease χ^2 without actually penalizing (or otherwise changing) its model's performance. This behavior can be seen below in Fig. 2.

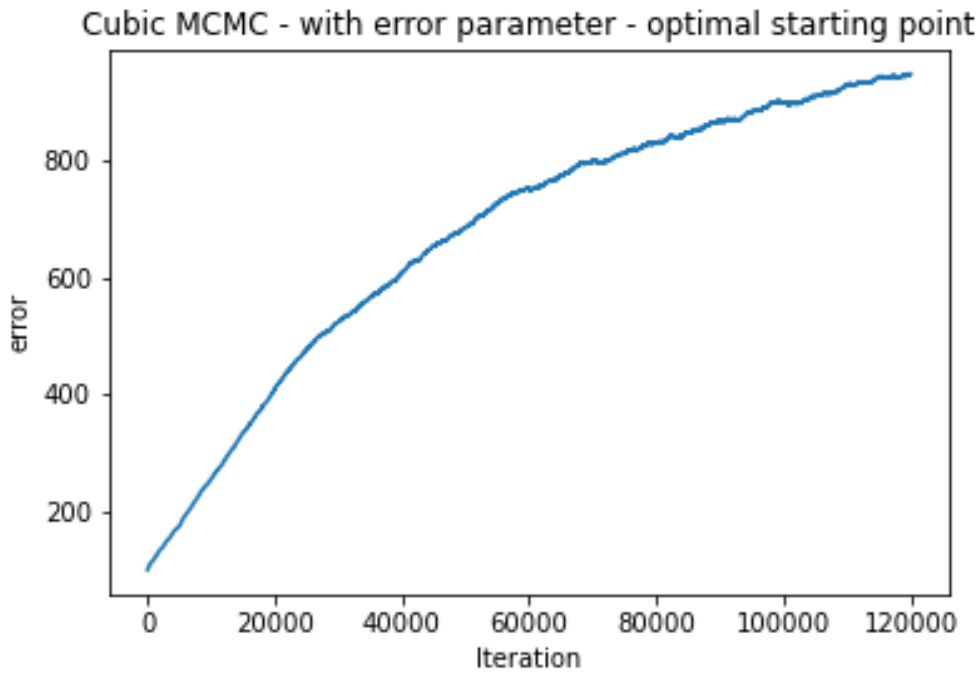

Figure 2. Although this figure is specifically from our cubic MCMC model, this graph is virtually identical for all of our MCMC models with parameterized error.

In an effort to remedy this, we tried to hold the initial error parameter (from the maximum likelihood estimate/optimal estimation) constant, preventing the MCMC from modifying it. However, the resulting parameter chains consistently diverged from the maximum likelihood estimate (where the minimized function parameterized error) and converged

[4] Equivalently, we could have used a log-likelihood function; we stuck with our particular implementation because it was already written.

towards a different maximum likelihood estimate (where the minimized function did *not* parameterize error). This was particularly uninformative, since our resulting chains had parameter values that were nowhere near our original, optimal inputs. Thus, we opted not to hold our error parameter constant.

X. Model Selection

Although it was apparent that a simple model would be insufficient to adequately fit the data, we continued anyways because we were curious about if such a model could be at least a *little* informative when used on such noisy data. Given that we don't work at a high-frequency trading company or financial institution (and that this is not an area of publicly accessible research), we also expected to not be able to "crack the code" of predicting Ethereum prices.

For the sake of experimentation, we tried four different models: three polynomial and one logistic. Visually, while some of them may seem adequate for limited ranges of data, none of them fit well. For the sake of practice, we calculated each model's BIC with the χ^2 values obtained from the earlier Markov chains.[5] The question of which model is the "best choice" (ie., which model has the lowest BIC) relies heavily on whether error was parameterized. Table 1 (Section XI) shows that models with parameterized errors were able to achieve *much* lower BIC values than models without. This intuitively checks out, as parameterized error models were able to increase error to decrease χ^2 and thus decrease BIC (see section IX). Therefore, when considering models that didn't parameterize error, we find that the best model (of those we tested) is the sigmoid model. When considering models that do parameterize error, we find that the best model is the quadratic model.

XI. Results

For models that did not converge, take corresponding calculated values with a grain of salt.

Model	Form	Converged?	Parameterized error?	Points trimmed	Points remaining	Acceptance ratio	BIC
1	Linear	Yes	No	4000	6000	0.32	4651195.73
2	Linear	No	Yes	10000	20000	0.85	643.63
3	Linear	No	Yes[6]	80000	20000	0.94	294.32
4	Quadratic	Yes	No	20000	30000	0.21	4421426.10
5	Quadratic	No	Yes	20000	30000	0.70	226.47
6	Cubic	Yes	No	17500	2500	0.25	4331992.18
7	Cubic	No	Yes	60000	60000	0.61	827.71
8	Sigmoid	Yes	No	6000	9000	0.26	3594872.76
9	Sigmoid	No	Yes	6000	4000	0.71	992.80

[5] Specifically, we used the approximation $BIC \approx kln(n) + \chi^2$, where k is the number of model parameters and n is the number of data points.

[6] This is the one model we tested that utilized combined error—see section IV.

Table 1. Convergence results for each model MCMC.

In the following table, the median value for a model parameter is given, followed by the 68% confidence interval for that parameter in parentheses. "--" indicates that the parameter did not exist in the given model. For example, all linear models have no "c" term because they are defined by $f(x) = a + bx$.

Model	a	b	c	d	error
1	156.05 (156.02,156.14)	18.45 (18.40,18.47)	--	--	--
2	290.46 (262.85,302.14)	55.89 (50.73,58.97)	--	--	1021.63 (722.60,1177.56)
3	326.39 (296.32,346.74)	58.76 (48.43,60.86)	--	--	1904.30 (1864.13,1917.54)
4	167.06 (166.95,167.17)	9.74 (9.66,9.78)	0.414 (0.413,0.415)	--	--
5	56.97 (-100.99,74.13)	102.41 (91.54,116.98)	-0.76 (-0.97,-0.70)	--	1949.04 (1485.85,2039.61)
6	171.65 (171.44,171.72)	5.64 (5.53,5.90)	0.86 (0.85,0.87)	-0.0067 (-0.0069,-0.0067)	--
7	-337.10 (-418.04,-263.39)	186.41 (173.47,202.75)	-3.75 (-4.29,-3.45)	0.0234 (0.0200,0.0271)	944.92 (748.87,949.40)
8	623.26 (572.74,933.64)	7.37 (6.93,12.51)	0.89 (0.29,1.04)	176.98 (145.94,178.26)	--
9	2490.74 (2368.87,2554.65)	14.11 (13.35,15.13)	0.29 (0.25,0.31)	162.14 (133.49,168.11)	767.39 (574.92,845.35)

Table 2. Parameter best-fit values and 68% confidence intervals for each model.

Additionally, we generated predictions for each model fit by taking 100 random samples from its corresponding chain and plotting it alongside the data. These figures can be seen on the poster.

XII. Miscellaneous figures – optimal estimation results

These are plots of initial parameter fits from optimal estimation. The linear models are found earlier in the paper (Fig. 1). The other models follow:

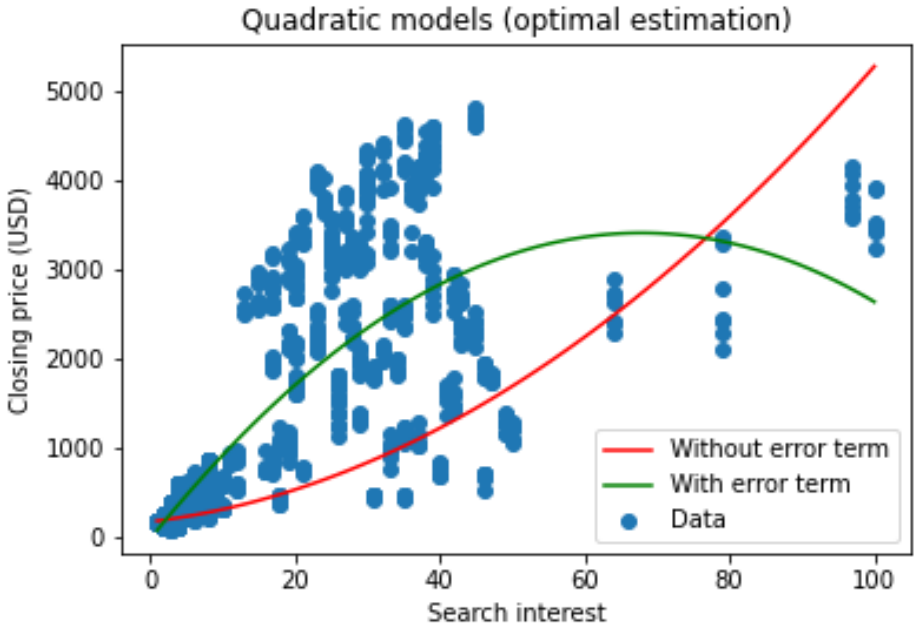

Figure 3. Unlike the linear case, we weren't sure if adding the error term here actually results in a better fit. For the sake of consistency, we tested both models anyway.

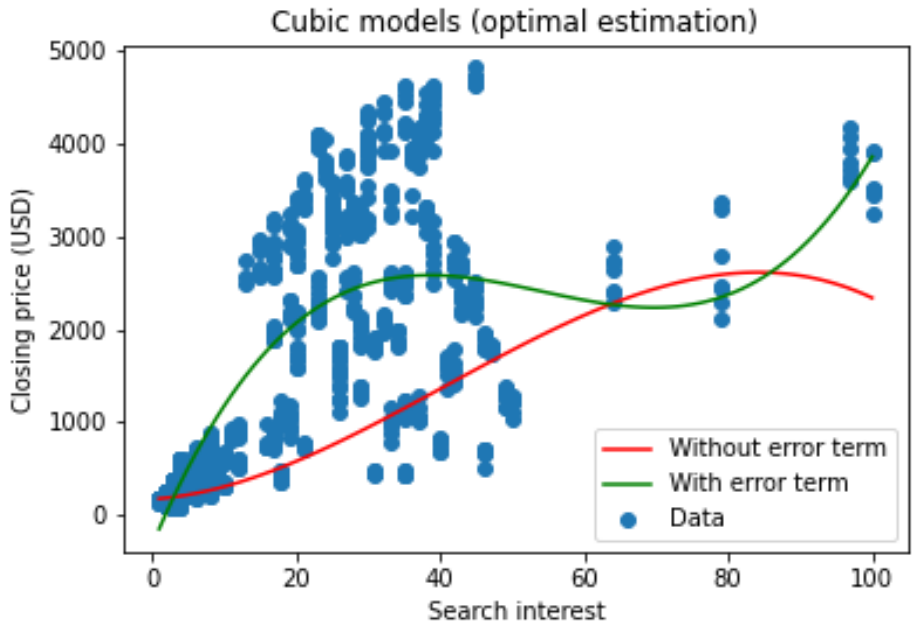

Figure 4. The model without the error term almost appears to be quadratic, although the x-axis is too short to display its full extent. In this case, adding the error term was probably a good idea, as it allowed the cubic model to actually *function* as a cubic model.

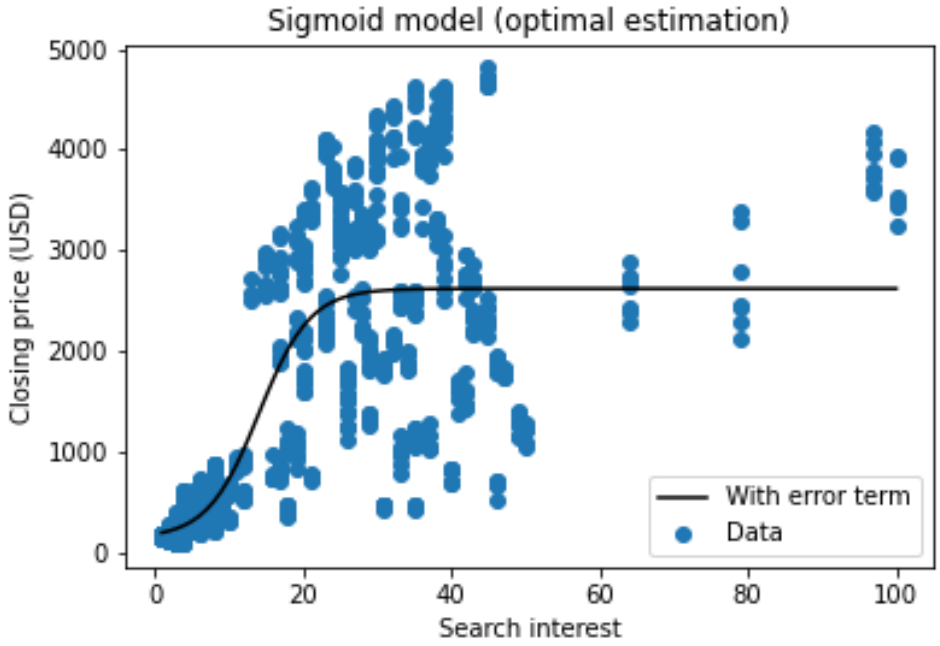

Figure 5. With the error term, the sigmoid model takes advantage of the fact that there is a fairly strong correlation between the closing price and search interest from 0 to approximately 20. The sigmoid model without the error term was not included because our minimizer produced a result that had no visual fit whatsoever with the data.

XIII. Conclusions

Our attempt to treat error as a parameter was largely unsuccessful because the models that did so had parameter chains that didn't converge. We did try various approaches to fix this (holding error constant, incorporating the error term into the model function itself), but none of them were successful. Possible alternative methods, then, include modifying the log-likelihood function or adding the error term into the model (preferably in a more sophisticated way than just literally adding it—we already tried that). To be fair, these problems might just be due to the fact that none of our models fit well with the data. Perhaps the problem is just that our models are inherently too simple.

While there were virtually no useful model conclusions drawn from this project, it was still fascinating to see how different models stood up to a noisy (and fairly uninformative) dataset.