Jonathan Balaban Follow

Feb 19, 2018 · 9 min read · ✦ · Listen

A Gentle Introduction to Maximum Likelihood Estimation

The first time I heard someone use the term **maximum likelihood estimation**, I went to Google and found out what it meant. Then I went to Wikipedia to find out what it really meant. I got this:

> *In statistics, **maximum likelihood estimation** (**MLE**) is a method of estimating the parameters of a statistical model given observations, by finding the parameter values that maximize the likelihood of making the observations given the parameters. MLE can be seen as a special case of the maximum a posteriori estimation (MAP) that assumes a uniform prior distribution of the parameters, or as a variant of the MAP that ignores the prior and which therefore is unregularized.*

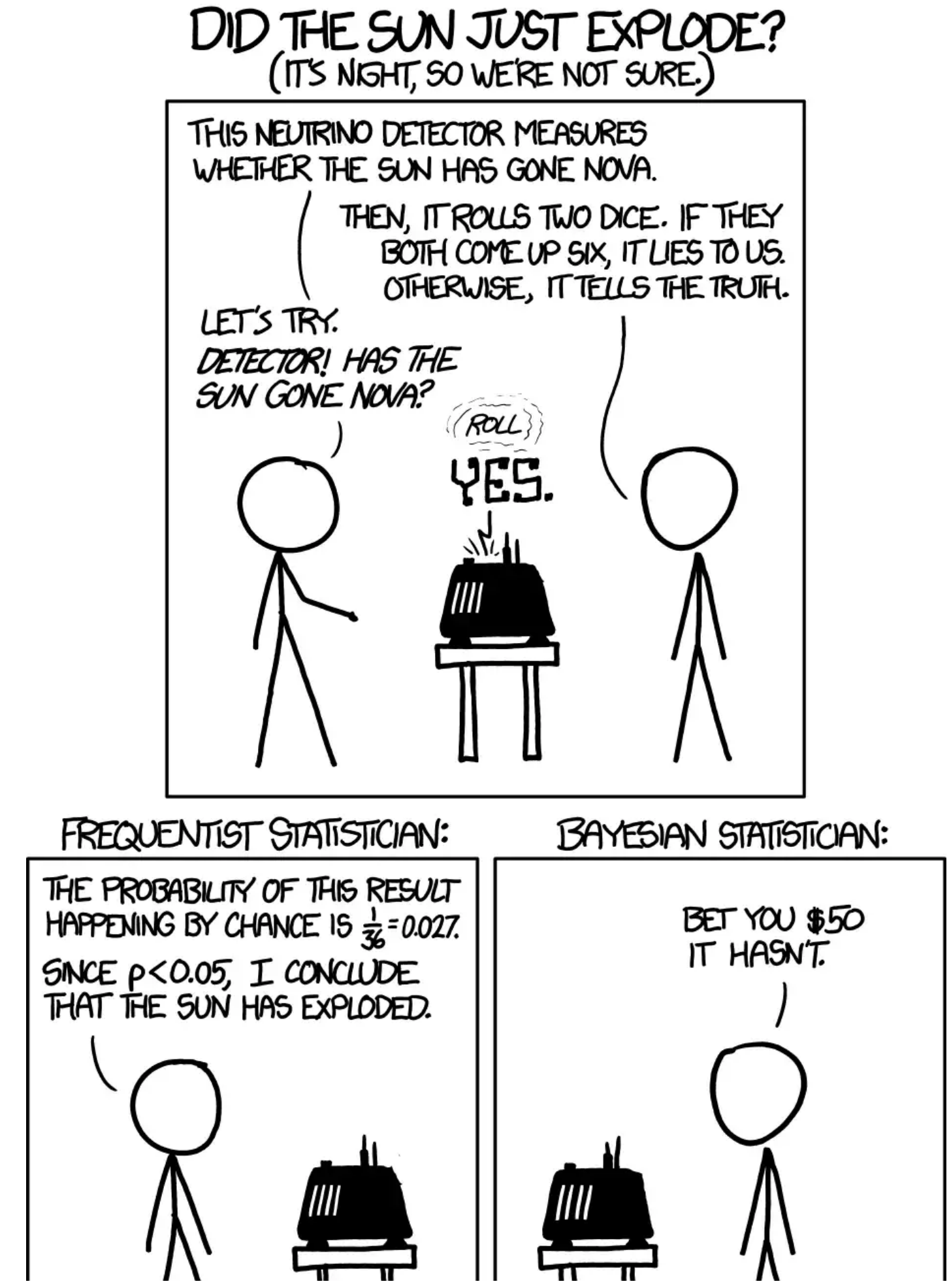
Okay??

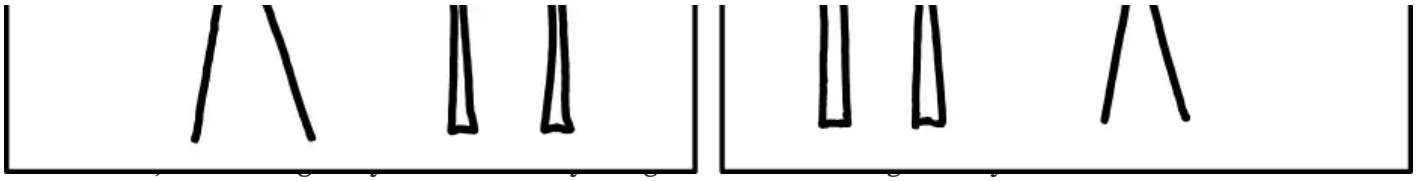
To spare you the wrestling required to understand and incorporate MLE into your data science workflow, ethos, and projects, I've compiled this guide. Below, we will:

- Set a probabilistic context to MLE
- Delve into the math required
- Look at how MLE works in Python
- Explore best practices in data science with MLE

But first, some xkcd:

Frequentists vs. Bayesians


DID THE SUN JUST EXPLODE?
(IT'S NIGHT, SO WE'RE NOT SURE.)
THIS NEUTRINO DETECTOR MEASURES WHETHER THE SUN HAS GONE NOVA.
THEN, IT ROLLS TWO DICE. IF THEY BOTH COME UP SIX, IT LIES TO US. OTHERWISE, IT TELLS THE TRUTH.
LET'S TRY.
DETECTOR! HAS THE SUN GONE NOVA?
ROLL
YES.
FREQUENTIST STATISTICIAN:
THE PROBABILITY OF THIS RESULT HAPPENING BY CHANCE IS $\frac{1}{36} = 0.027$.
SINCE $p < 0.05$, I CONCLUDE THAT THE SUN HAS EXPLODED.
BAYESIAN STATISTICIAN:
BET YOU $50 IT HASN'T.



MLE in a nutshell helps us answer this question:

> Which are the best parameters/coefficients for my model?

And interestingly, you can use either school of though to explain why MLE works! Because, while MLE gives a spot estimate — which is common to frequentist outputs — it can be considered a special case of maximum a posteriori (MAP) estimation, where we use a naïve prior and never bother to update it.

Setting Up Our Problem

To approach MLE today, let's come from the Bayesian angle, and use Bayes Theorem to frame our question as such:

```
P(β|y) = P(y|β) x P(β) / P(y)
```

Or, in English:

```
posterior = likelihood x prior / evidence
```

We can effectively ignore the `prior` and the `evidence` because — given the Wiki definition of a uniform prior distribution — all coefficient values are equally likely. And probability of all data values (assume continuous) are equally likely, and basically zero.

So, in actual English: the probability of some specific coefficients given I'm seeing some results, relates to framing the question the exact opposite way. Which helps, cause that question is WAY easier to solve.

Probability and Likelihood

We'll now introduce the concept of likelihood, or `L` in our code henceforth. To grasp the distinction, I'll tag in excerpts from Randy Gallistel's excellent post:

> *The distinction between probability and likelihood is fundamentally important: Probability attaches to possible results; likelihood attaches to hypotheses.*
>
> *Possible results are mutually exclusive and exhaustive. Suppose we ask a subject to predict the outcome of each of 10 tosses of a coin. There are only 11 possible results (0 to 10 correct predictions). The actual result will always be one and o one of the possible results. Thus, the probabilities that attach to the possible results must sum to 1.*

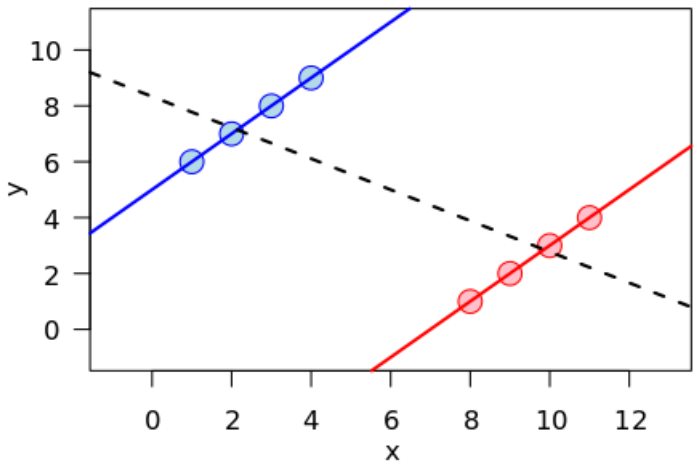
> *Hypotheses, unlike results, are neither mutually exclusive nor exhaustive. Suppose that the first subject we test predicts 7 of the 10 outcomes correctly. I might hypothesize that the subject just guessed, and you might hypothesize that the subject* may *be somewhat clairvoyant, by which you mean that the subject may be expected to correctly predict the results at slightly greater than chance rates over the long run. These are different hypotheses, but they are not mutually exclusive, because you hedged when you said "*may *be." You thereby allowed your hypothesis to include mine. In technical terminology, my hypothesis is nested within yours. Someone else might hypothesize that the subject is strongly clairvoyant and that the observed result underestimates the probability that her next prediction will be correct. Another person could hypothesize something else altogether. There is no limit to the hypotheses one might entertain.*
>
> *The set of hypotheses to which we attach likelihoods is limited by our capacity to dream them up. In practice, we can rarely be confident that we have imagined all the possible hypotheses. Our concern is to estimate the extent to which the experimental results affect the relative likelihood of the hypotheses we and others currently entertain. Because we generally do not entertain the full set of alternative hypotheses and because some are nested within others, the likelihoods that we attach to our hypotheses do not have any meaning in and of themselves; only the relative likelihoods — that is, the ratios of two likelihoods — have meaning.*

Amazing! Thank you, Randy!

MLE is Frequentist, but can be motivated from a Bayesian perspective:

- Frequentists can claim MLE because it's a **point-wise estimate** (not a distribution) and it assumes **no prior distribution** (technically, uninformed or uniform).
- Also, MLE's do not give the 95% probability region for the true parameter value.
- However, MLE is a special form of MAP, and uses the concept of likelihood, which is central to the Bayesian philosophy.

BEWARE the assumption of naïve or uniform priors!! You may mis-attribute the data toward a model that is highly unlikely. You may fall victim to Simpson's Paradox, as below. You could easily be tricked by a small sample size.



Simpson's Paradox

All of the above are problems that Frequentists and data scientists must deal with or be aware of, so there's nothing inherently worse about MLE.

Back to Our Problem

So if $p(y|\beta)$ is equivalent to $L(\beta|y)$, then $p(y_1,y_2,\ldots,y_n|\beta)$ is equivalent to $L(\beta|y_1,y_2,\ldots,y_n)$. Also, remember that we can multiply independent probabilities, like so:

$$p(A,B) = p(A)p(B)$$

We are getting close! Here's our current setup:

$$L(\beta|y1,y2,\ldots,yn) = p(y1|\beta)p(y2|\beta),\ldots,p(yn|\beta) = \prod p(yi|\beta)$$

That part at the right looks like something we can maximize:

$$\max_{\beta} \left\{ \prod_{i} p(y_i|\beta) \right\}$$

Initial Cost Function

But we can do even better! How about using the natural log to turn our product function into a sum function? Logs are monotonic transformations, so we'll simplify our computation but maintain our optimal result.

$$\max_{\beta} \left\{ \ln \left\{ \prod_{i} p(y_i|\beta) \right\} \right\}$$

Halfway there!

Our final cost function looks like this:

$$\max_{\beta} \left\{ \sum_{i} \ln \{ p(y_i|\beta) \} \right\}$$

Ready to roll!

To keep things simple from here, let's assume we have a regression problem, so our outcome is continuous. MLE works great for classification problems with discrete outcomes, but we have to use different distribution functions, depending on how many classes we have, etc.

Now, remembering that a central assumption of models like Ordinary Least Squares (OLS) is that the residuals are normally distributed around mean zero, our fitted OLS model literally becomes the embodiment of a maximum expectation of `y`. And our probability distribution is... Normal!

$$y_i \sim N(x_i\beta, \sigma^2)$$

y is normally distributed around our ŷ

Because computers are *much better* than us at computing the probabilities, we'll turn to Python from here!

MLE in Python

Implementing MLE in your data science modeling pipeline can be quite simple, with a variety of approaches. Below is one approach you can steal to get started.

Setup

MLE is easy if you import the right packages:

```
# import libraries
import numpy as np, pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
from scipy.optimize import minimize
import scipy.stats as stats

import pymc3 as pm3
import numdifftools as ndt
import statsmodels.api as sm
from statsmodels.base.model import GenericLikelihoodModel
%matplotlib inline
```

From there, we will generate data that follows a normally distributed errors around a ground truth function:

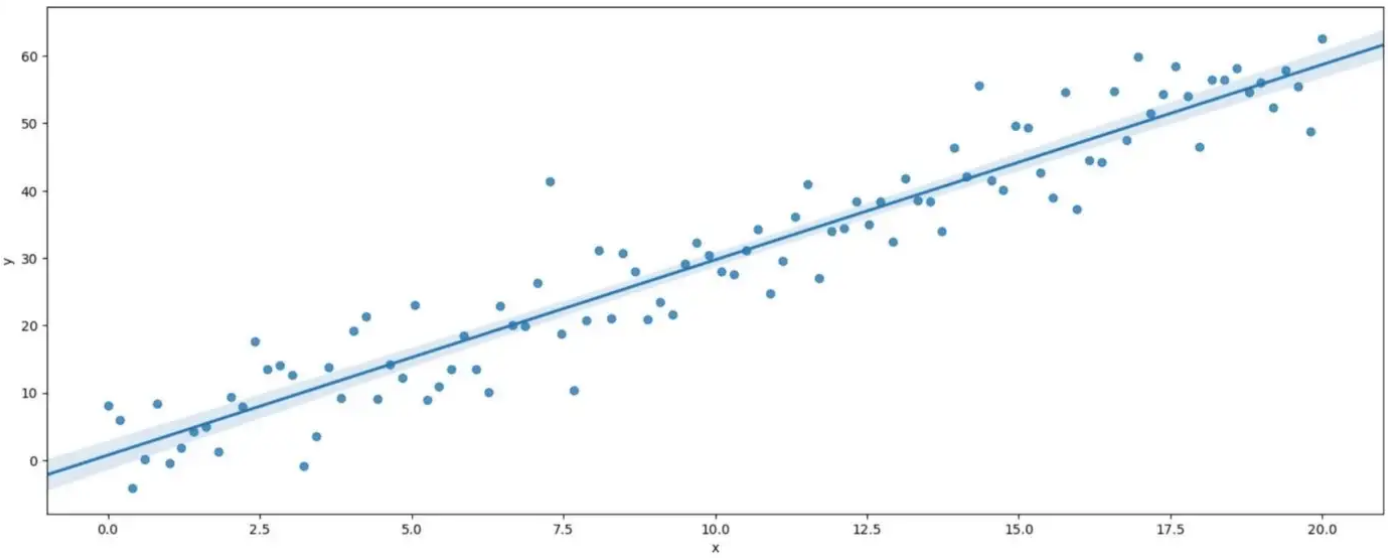
```
# generate data
N = 100
x = np.linspace(0,20,N)
ϵ = np.random.normal(loc = 0.0, scale = 5.0, size = N)
y = 3*x + ϵ

df = pd.DataFrame({'y':y, 'x':x})
df['constant'] = 1
```

Finally, let's visualize using Seaborn's regplot:

```
# plot
sns.regplot(df.x, df.y);
```

I get the below, and you should see something similar. But, keep in mind there's randomness here and we didn't use a seed:



Scatter plot with OLS line and confidence intervals

Modeling OLS with Statsmodels

Since we created regression-like, continuous data, we will use `sm.OLS` to calculate the best coefficients and Log-likelihood (LL) as a benchmark.

```
# split features and target
X = df[['constant', 'x']]
```

```
# fit model and summarize
sm.OLS(y,X).fit().summary()
```

I get this, and will record the fitted model's coefficients:

	coef	std err	t	P>|t|	[0.025	0.975]
constant	0.7362	1.089	0.676	0.500	-1.424	2.896
x	2.8986	0.094	30.825	0.000	2.712	3.085

Omnibus:	5.436	Durbin-Watson:	1.992
Prob(Omnibus):	0.066	Jarque-Bera (JB):	4.966
Skew:	0.417	Prob(JB):	0.0835
Kurtosis:	3.705	Cond. No.	23.1

Notice `constant` is close to zero, and `beta` for feature `x` is close to 3, per the ground truth generator we used.

Maximizing LL to solve for Optimal Coefficients

From here, we'll use a combination of packages and custom functions to see if we can calculate the same OLS results above using MLE methods.

Because `scipy.optimize` has only a `minimize` method, we'll minimize the negative of the log-likelihood. This is even what they recommend! Math trickery is often faster and easier than re-inventing the wheel!

We can build a simple function that does everything in one pass for regression outputs:

```
# define likelihood function
def MLERegression(params):
 intercept, beta, sd = params[0], params[1], params[2] # inputs are guesses at our
parameters
 yhat = intercept + beta*x # predictions

# next, we flip the Bayesian question
# compute PDF of observed values normally distributed around mean (yhat)
# with a standard deviation of sd
 negLL = -np.sum( stats.norm.logpdf(y, loc=yhat, scale=sd) )
```

```
    # return negative LL
     return(negLL)
```

Now that we have a cost function, let's initialize and minimize it:

```
# let's start with some random coefficient guesses and optimize
guess = np.array([5,5,2])

results = minimize(MLERegression, guess, method = 'Nelder-Mead',
 options={'disp': True})

------------------------------------------------------------------
Optimization terminated successfully.
         Current function value: 311.060386
         Iterations: 111
         Function evaluations: 195
```

Let's check the results:

```
results # this gives us verbosity around our minimization
# notice our final key and associated values…

------------------------------------------------------------------
final_simplex: (array([[0.45115297, 3.03667376, 4.86925122],
       [0.45123459, 3.03666955, 4.86924261],
       [0.45116379, 3.03667852, 4.86921688],
       [0.45119056, 3.03666796, 4.8692127 ]]), array([300.18758478, 300.18758478,
300.18758478, 300.18758479]))
           fun: 300.18758477994425
       message: 'Optimization terminated successfully.'
          nfev: 148
           nit: 80
        status: 0
       success: True
             x: array([0.45115297, 3.03667376, 4.86925122])
```

And we can clean up further:

```
# drop results into df and round to match statsmodels
resultsdf = pd.DataFrame({'coef':results['x']})
resultsdf.index=['constant','x','sigma']
np.round(resultsdf.head(2), 4)

# do our numbers match the OLS model?
------------------------------------------------------------------
```

You'll note that OLS and MLE match up nicely! Your results will differ, again, as we're not using random seeds.

Best Practices for MLE

Before we go any further, this might be a good moment to reinforce our trust in MLE. As our regression baseline, we know that Ordinary Least Squares — by definition — is the best linear unbiased estimator for

continuous outcomes that have normally distributed residuals and meet the other assumptions of linear regression. Is using MLE to find our coefficients as robust?

Yes!

- MLE is consistent with OLS.
- With infinite data, it will estimate the optimal *β, and approximate it well* for small but robust datasets.
- MLE is efficient; no consistent estimator has lower asymptotic mean squared error than MLE.

So it looks like it fully replicates what OLS does. Then… why use MLE instead of OLS?

Because!

- MLE is generalizable for regression and classification!
- MLE is efficient; no consistent estimator has lower asymptotic error than MLE if you're using the right distribution.

We can think of MLE as a modular way of fitting models by optimizing a probabilistic cost function!

Four major steps in applying MLE:

1. Define the likelihood, ensuring you're using the correct distribution for your regression or classification problem.
2. Take the natural log and reduce the product function to a sum function.
3. Maximize — or minimize the negative of — the objective function.
4. Verify that uniform priors are a safe assumption! Otherwise, you could attribute the data to a generating function or model of the world that fails the Law of Parsimony.

There's much more in the MLE space, including categorical distributions, using Bayesian statistics packages like `PyMC3` , etc. But we'll stop here for today.

How are you using MLE in your data science workflow? Comment below, or connect with me on LinkedIn or Twitter!

Special thanks to Chad Scherrer for his excellent peer review.

Machine Learning Data Science Statistics Mathematics Data