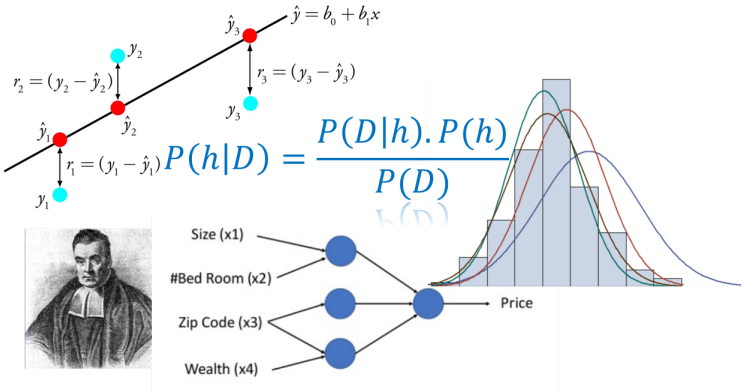
Editorial Associate "Towards Data Science" | Sr. Principal Engineer | Ph.D. in EE (U. of Illinois)| AI/ML certification, Stanford, MIT | Open-source contributor

Sep 13 · 8 min read

Where did the least-square come from?

What would you say in a machine learning interview, if asked about the mathematical basis of the least-square loss function?



Question: Why do you square the error in a regression machine learning task?

Ans: "Why, of course, it turns out all the errors (residuals) into positive quantities!"

Question: "OK, why not use a simpler absolute value function $|x|$ to make all the errors positive?"

Ans: "Aha, you are trying to trick me. Absolute value function is not differentiable everywhere!"

Question: "That should not matter much for numerical algorithms. LASSO regression uses a term with absolute value and it can be handled. Also, why not 4th-power of x or $\log(1+x^2)$? **What's so special about squaring the error?**

Ans: Hmm…

A Bayesian argument

Remember that, for all tricky questions in machine learning, you can whip up a serious-sounding answer if you mix the word "***Bayesian***" in your argument.

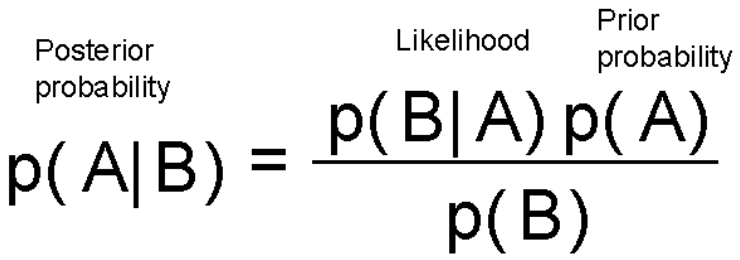
OK, I was kidding there.

But yes, we should definitely have the argument ready about where popular loss functions like least-square and cross-entropy come from—at least when we try to find the most likely hypothesis for a supervised learning problem using Bayesian argument.

Read on…

The Basics: Bayes Theorem and the 'Most Probable Hypothesis'

Bayes' theorem is probably the most influential identity of probability theory for modern machine learning and artificial intelligence systems. For a super intuitive introduction to the topic, please see this great tutorial by Brandon Rohrer. I will just concentrate on the equation.



This essentially tells that you update your belief (*prior probability*) after seeing the data/evidence (*likelihood*) and assign the updated degree of belief to the term *posterior probability*. You can start with a belief, but each data point will either strengthen or weaken that belief and you update your hypothesis all the time.

Let us now recast the Bayes' theorem in different symbols—symbols pertaining to data science. Let us denote, data by ***D*** and hypothesis by ***h***. This means we apply Bayes' formula to try to determine ***what***

hypothesis the data came from, given the data. We rewrite the theorem as,

$$P(h|D) = \frac{P(D|h).P(h)}{P(D)}$$

Now, in general, we have a large (often infinite) hypothesis space i.e. many hypotheses to choose from. The essence of Bayesian inference is that we want to examine the data to maximize the probability of one hypothesis which is most likely to give rise to the observed data. We basically want to determine ***argmax*** of the $P(h|D)$ i.e. we want to know for which h, observed D is most probable.

A shortcut trick: Maximum Likelihood

The equation above looks simple but it is notoriously tricky to compute in practice—because of extremely large hypothesis space and complexity in evaluating integrals over complicated probability distribution functions.

However, in the quest of our search for the '*most probable hypothesis given data,*' we can simplify it further.

- We can drop the term in the denominator it does not have any term containing h i.e. hypothesis. We can imagine it as a normalizer to make total probability sum up to 1.
- **Uniform prior assumption**—this essentially relaxes any assumption on the nature of $P(h)$ by making it uniform i.e. all hypotheses are probable. Then it is a constant number $1/|Vsd|$ where $|Vsd|$ is the size of the **version space** i.e. a set of all hypothesis consistent with the training data. Then it does not actually figure in the determination of the maximally probable hypothesis.

After these two simplifying assumptions, the **maximum likelihood (ML)** hypothesis can be given by,

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}}\ P(D|h)$$

> *This simply means the* most likely *hypothesis is the one for which the* ***conditional probability*** ***of the observed data*** *(given the hypothesis)*

> *reaches maximum.*

Next piece in the puzzle: Noise in the data

We generally start using least-square error while learning about simple linear regression back in Stats 101 but this simple-looking loss function resides firmly inside pretty much every supervised machine learning algorithm viz. linear models, splines, decision trees, or deep learning networks.

So, what's special about it? Is it related to the Bayesian inference in any way?

> *It turns out that, the key connection between the least-square error and Bayesian inference is through the* ***assumed nature of the error or residuals****.*

Measured/observed data is never error-free and there is always **random noise** associated with data, which can be thought of the signal of interest. Task of a machine learning algorithm is to estimate/approximate the function which could have generated the data by separating the signal from the noise.

But what can we say about the nature of this noise? It turns out that noise can be modeled as a random variable. Therefore, we can associate a probability distribution of our choice to this random variable.

> ***One of the key assumptions of least-square optimization is that probability distribution over residuals is our trusted old friend—Gaussian Normal.***

This means that every data point (***d***) in a supervised learning training data set can be written as the sum of the unknown function ***f***(***x***) (which the learning algorithm is trying to approximate) and an error term which is drawn from a Normal distribution of zero mean (***μ***) and unknown variance ***σ²***. This is what I mean,

$$d_i = f(x_i) + e(i) = f(x_i) + \boldsymbol{N}(\mu = 0, \sigma^2)$$

And from this assertion, we can easily derive that the maximum likely hypothesis is the one which minimizes the least-square error.

Math warning: There is no way around some bit of math to formally derive the least-square optimization criteria from the ML hypothesis. And there is no good way to type in math in Medium. So, I have to paste an image to show the derivation. **Feel free to skip this section, I will summarize the key conclusion in the next section.**

Derivation of least-square from Maximum Likelihood hypothesis

We start with the **maximum likelihood hypothesis**:

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}}\, p(D|h)$$

We assume a fixed set of training instances $\langle x_1, x_2, \ldots, x_n \rangle$.
Therefore, we consider the data D to be the corresponding sequence of target values $D = \langle d_1, d_2, \ldots, d_n \rangle$
Here, $d_i = f(x_i) + e_i$ where e_i is the ***Normally distributed*** error.
Assuming the training examples are mutually independent given h, we can write $P(D|h)$ as the product of the various $p(d_i|h)$

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \prod_{i=1}^{m} p(d_i|h)$$

Given that the noise e_i obeys a Normal distribution with zero mean(μ) and unknown variance σ^2, each d_i must also obey a Normal distribution with variance σ^2 centered around the true target value $f(x_i)$ rather than zero. Therefore $p(d_i|h)$ can be written as a Normal distribution with variance σ^2 and mean $\mu = f(x_i)$. Because we are writing the expression for the probability of d_i given that h is the correct description of the target function f, we will also substitute $\mu = f(x_i) = h(x_i)$, yielding,

$$\begin{aligned} h_{ML} &= \underset{h \in H}{\operatorname{argmax}} \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(d_i - \mu)^2} \\ &= \underset{h \in H}{\operatorname{argmax}} \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(d_i - h(x_i))^2} \end{aligned}$$

We now apply **log-likelihood** transformation. This is justified because $ln(p)$ is a monotonic function of p. Therefore maximizing $ln(p)$ also maximizes p. So, the product becomes summation.

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \sum_{i=1}^{m} \left[\ln \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2}(d_i - h(x_i))^2] \right]$$

The first term in this expression is a constant independent of h, and can therefore be discarded, yielding

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \sum_{i=1}^{m} -\frac{1}{2\sigma^2}(d_i - h(x_i))^2$$

Maximizing this negative quantity is equivalent to minimizing the corresponding positive quantity.

$$h_{ML} = \underset{h \in H}{\operatorname{argmin}} \sum_{i=1}^{m} \frac{1}{2\sigma^2}(d_i - h(x_i))^2$$

Finally, we can again discard constants that are independent of h.

$$h_{ML} = \underset{h \in H}{\operatorname{argmin}} \sum_{i=1}^{m} (d_i - h(x_i))^2$$

> ***Voila! The last term is nothing but simple least-square minimization.***

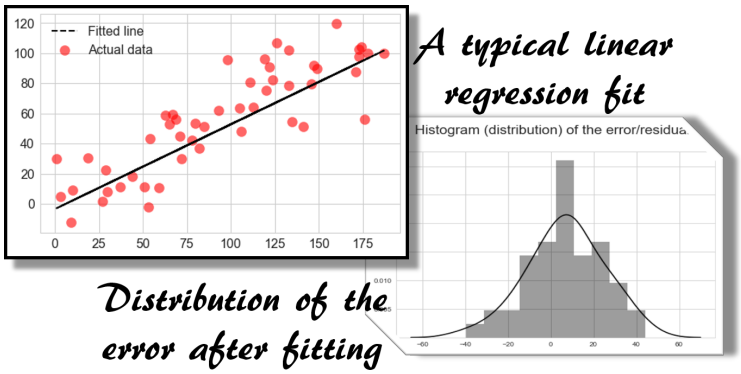
So, what did all this math show?

It showed that, starting from the assumption that error of a supervised training dataset is distributed over a Gaussian Normal, the **maximum likely hypothesis for that training data is the one which minimizes the least-square error loss function.**

There is no assumption about the type of the learning algorithm. **This applies equally to anything and everything starting from the simple linear regression to deep neural net.**

> *Such is the power and unifying nature of Bayesian inference.*

Here is the a typical scenario for a linear regression fit. Bayesian inference argument works on this model and lends credibility to the choice of square of the error as the optimum loss function.



Is the assumption of Normality sound enough?

You can question the validity of the assumption about the Normally distributed error terms. But in most cases, it works. This follows from Central Limit Theorem (CLT) in the sense that error or noise is never generated by a single underlying process but rises from the combined impact of multiple sub-processes. And when large number of random sub-processes combine, their averaged value follows Normal distribution (from CLT). Therefore, it is not a stretch to assume such distribution for most of the machine learning task we undertake.

Does similar argument hold for classification problems?

Least-square loss function is used in regression tasks. But what about the classification problems where we deal with classes and probabilities, and not with arbitrary real numbers?

Amazingly, it turns out that similar derivation can be done using ML hypothesis and simple choice of class definition to arrive at,

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \sum_{i=1}^{m} d_i . \ln\left(h(x_i)\right) + (1 - d_i) . \ln\left(1 - h(x_i)\right)$$

… which is nothing but **cross-entropy loss function.**

So, the same Bayesian inference yields the cross-entropy as the preferred choice of loss function for obtaining maximum likely hypothesis in classification problems.

Summary and Conclusions

We can summarize and extend our discussions and arguments in the article through the following points,

- **Maximum likelihood estimate (MLE)** is a powerful technique to arrive at the most probable hypothesis for a given set of data if we can make a uniform prior assumption i.e. at the start, all hypotheses are equally likely.
- If we can assume that each data point in a machine learning task is a sum of the true function and some random noise variable which is normally distributed, then we can derive the fact that the **maximally probable hypothesis is the one which minimizes the square loss function**.
- This conclusion holds true independent of the nature of the machine learning algorithm.
- However, another implicit assumption is the **mutual independence** of the data point which enables us to write the joint probability as a simple product of individual probabilities. This also **underscores the importance of removing collinearity among training samples before a machine learning model should be build.**

> *At the end of the day, you can say that the least-square minimization is really special as it is, in fact, intimately related to the most celebrated distribution function in this universe :-)*

. . .

If you have any questions or ideas to share, please contact the author at **tirthajyoti[AT]gmail.com**. Also, you can check author's **GitHub repositories** for other fun code snippets in Python, R, or MATLAB and machine learning resources. If you are, like me, passionate about machine learning/data science, please feel free to add me on LinkedIn or follow me on Twitter.