ML notes: Why the log-likelihood?

Morgan Follow
Jun 13, 2017 · 9 min read

> *Disclaimer:*

> *This blog post is in my "ML notes" category. Its goal is to help me make sure I understand the tools and theories used in ML. I believe that pedagogically explaining what I learn, removing step by step any unknowns, is the best way to achieve this goal.*

> *What do you need? A little bit of mathematical background in calculus, algebra, probability and machine learning (mainly definitions).*

> *If you find any mistakes, please get in touch. I would be very sad to keep wrong beliefs in my head and discover them too late, Thanks!*

. . .

Some context

Machine learning is about modelling: you experience something and you wonder afterwards if you could have predicted it, or even better if you can build something that could have predicted it for you. Secretly, you are hoping that your model will predict future experiences, people call that "generalisation".

Let's normalise this hand-wavy description:

- A set of experiences can be seen as some data forming a dataset "D". It can be pictures, sounds, spreadsheets, etc, or even a mix.
- To handle those experiences, you define a model. You have a lot of choices here but in the end, you get an algorithm which covers initialisation (past), learning (present) and predictions (future).

The modelling part

Choosing a family of models is already choosing something which means that either you are choosing randomly or you have a prior knowledge of the task at hand and you are making an educated guess. Anyway, we already are in the realm of probability before we even know it.

On top of that, after choosing this family of model, you end up choosing many hyper-parameters (again either randomly or thanks to an educated guess). Even the function to initialise the parameters of your model is in itself a hyper-parameter.

In our case, we will consider all those choices (HPs and model architecture) fixed: we have no way to change them. Deal with it!

> *That said, some models have parameters, some don't. For the rest of this blog post, I will focus on models with parameters for ease of notations, those parameters will be noted using the Greek letter theta.*

The origin of everything

Let's sum up:

- We have a **dataset** and a **model** with parameters. 👌🏼
- We would like to know if our model can learn from this dataset to predict new data. But how do we do that? 😨

At that point, some smart people started to get interested into something called the likelihood: basically you have some knowledge about the world, you experience something and you wonder how likely your knowledge and this new experience fit together.

If it doesn't fit, you are surprised meaning it was unlikely! On the other hand, if it does fit a lot, you're not surprised, it was predictable.

Hopefully, someone has already thought about how to write down a "likelihood" of something given the knowledge of another thing. Remember that, at the end of the day, we are interested in finding the most likely parameters given our dataset: the dataset is considered fixed too.

Think about it, we are interested in **the likelihood of the model and its parameters given the dataset** which by definition is equal to **the probability of the dataset given the model and its parameters:**

$$L(\theta, m|\mathcal{D}) = p(\mathcal{D}|\theta, m)$$

Likelihood of a model and its parameters given a dataset

Well, why is this formula in seeming contradiction with the Bayes formula? Because (see on p. 6 below), the probability of a dataset given a model is a constant in respect to thetas so we can ignore it in the optimisation process.

Now given the fact that the dataset is made of "n" datum, we can rewrite this as the **joint probability of all the individual datum:**

$$p(\mathcal{D}|\theta, m) = p_{D_1,\ldots,D_n}(d_1, \ldots, d_n|\theta, m)$$

The joint probability of the data

This is clearer but not very helpful because all the datum are entangled in one big joint probability distribution. To fix this we must make some assumptions about our dataset.

Assumption 1: Independence

The first one is to consider all of our data independent of each other. Thanks to the product rule, this could be rewritten this way:

$$p_{D_1,\ldots,D_n}(d_1, \ldots, d_n|\theta, m) \overset{\text{independence}}{=} \prod_i p_{D_i}(d_i|\theta, m)$$

here the subscripts D1,..., Dn mean that data {d1,..., dn} are DRAWN from INDEPENDENT probability distributions {D1,..., Dn}

P_D. means the probability of d defined by D

Effect of independence on a joint distribution

Notice that no one spoke about the probability distribution of each datum so far. Before jumping to any conclusion you must keep track of each distribution of probability (The big "D_i"). But this means that, in the most general case, we have only one datum per probability distribution, you can't figure out anything useful from only one sample, so we will use another assumption to get rid of it.

Assumption 2: Identical distribution

The second assumption is to consider our data identically distributed: It means that every datum is coming from the same probability distribution. This allows us to remove the subscript "i" from the distribution:

$$\prod_i p_{D_i}(d_i|\theta, m) \overset{\text{identically\_distributed}}{=} \prod_i p_D(d_i|\theta, m) = \prod_i p(d_i|\theta, m)$$

Effect of identically distributed variables

Now we have "n" independent datum coming from the same distribution. We might have a good chance to be able to infer something about this distribution.

This is why people are nearly always relying on what is called the i.i.d assumption. Before reaching this product we don't really know how to deal with the previous notations. To sum up, we have:

$$p(\mathcal{D}|\theta, m) \overset{\text{i.i.d.}}{=} \prod_i p(d_i|\theta, m)$$

Effect of i.i.d. assumptions

The logarithm

Now we start to have seriously simplified our problem, at least the product notation is concise and clear. But remember that given a dataset, what we want is to maximise the likelihood of the parameters.

Maximisation is an optimisation problem, most of the time you have to iterate to get close to it. Maximising the likelihood would mean maximising the above product which is still entangling all the datum. That's not great for optimisation because it means that you have to load all the datum into memory at once to create the product and only then, gets the partial derivatives of all thetas and apply your optimisation step. That would require a huge memory capacity.

On the opposite, **a sum is a lot easier to optimise as the derivative of a sum is the sum of derivatives.** If we had a sum instead of a product, we could load one datum at a time compute its partial derivatives, accumulating those gradients and apply the optimisation step at the end. Solving the problem of memory space!

And that's actually why we use the logarithm: to get rid of this nasty product signs. (Some good guys on Reddit pointed out those awesome answers to me. Look it up, there is even more insight!)

We can write down this (thanks to the fact that the logarithm is a monotonically increasing function):

$$\underset{\theta}{\operatorname{argmax}} \prod_i p(d_i|\theta, m) = \underset{\theta}{\operatorname{argmax}} \sum_i \log(p(d_i|\theta, m))$$

Log-likelihood as a way to change a product into a sum

Effectively un-correlating each datum! (Notice how I added the maximum in there! If I didn't the equality would not hold)

So here we are, **maximising the log-likelihood of the parameters given a dataset** (which is strictly equivalent to minimising the negative log-likelihood, of course).

The choice of minimum or maximum depends only on how you want to frame your problem, how you want to change the probability of each datum into a cost function for the optimisation part.

Remark 1: Mini-batch

We have solved the memory space problem, but we still have to compute all the dataset before making any updates. All the people in ML know we can actually use mini-batch instead of the whole batch of data, why is that?

Notice that we can change our last equation to get an expectation:

U_D means and a distribution of data uniformly sampled from D (another distribution

$$\begin{aligned}\underset{\theta}{\operatorname{argmax}} \sum_i \log(p(d_i|\theta, m)) &= \underset{\theta}{\operatorname{argmax}} \frac{1}{n} \sum_i \log(p(d_i|\theta, m)) \quad (1)\\ &= \underset{\theta}{\operatorname{argmax}}\, E_{d \sim U_D}[log(p(d|\theta, m))] \quad (2)\end{aligned}$$

Indeed, why sample the entire dataset to maxmize this sum?
We can just sample n samples and do optimization on them. So sum and expectation can do (but not be) the same thing.

Recovering mini-batch optimisation

(2) is the expectation when you sample your data uniformly from the dataset (probability of each datum is 1 over n).

Having an expectation is great because now, we can use **stochastic approximation** and recover the stochastic gradient descent. 💪🏼

Remark 2: MAP and regularisation

At that point, one could point out that maximising the likelihood given a fixed dataset can actually be a bad idea to predict future experience, your model might end up being over-specialised on this precise dataset, this is called "overfitting".

And that's definitely true, but when you look at the equations so far, one can wonder what is missing? Where did we make an untold assumption?

Well, it's actually when we wrote down the first equation! Because before training any models with parameters, you have to choose some initial ones and here again this choice is either random or an educated guess, meaning that we can frame it in terms of probabilities thanks to "the Bayes".

"The Bayes" gave us another equation introducing a relation between what people call "posterior" and "prior" distribution and the likelihood (disguised as a probability):

- The prior is when you haven't seen any data about your problem and yet you have to choose some initial parameters for your model (you must start somewhere), how do you do that? Well, you can at least sample your parameters from a distribution reflecting your educated guess or sample from the uniform distribution reflecting a random choice.

VS.

- The posterior is the new probability distribution of those parameters now that you've seen the data: you've experienced new things, you can adjust your knowledge.

Let's look at the Bayes formula:

Don't worry about m here. Both sides are CONDITIONED on the model (m). You can do that with probability equations.

posterior is not likelihood !!!

$$p(\theta|\mathcal{D},m) = \frac{\overset{\text{likelihood}}{p(\mathcal{D}|\theta,m)}\,\overset{\text{prior}}{p(\theta|m)}}{\underset{\text{evidence}}{p(\mathcal{D}|m)}}$$

The Bayes formula

On the left, there is the posterior (be careful! this is not the likelihood), on the top right there is the likelihood and the prior.

Notice, on the bottom right, that the probability of a dataset given a model is a constant in respect to thetas so we can ignore it in the optimisation process. If we redo the set of calculus we've done before, it gives us what one calls the MAP (the Maximum A Posteriori) :

$$\underset{\theta}{\operatorname{argmax}}\ p(\theta|\mathcal{D}, m) = \underset{\theta}{\operatorname{argmax}} \sum_i \log(p(d_i|\theta, m)) + \log(p(\theta|m))$$

Making the regularisation terms appear in our optimisation formula

A new term appears in our optimisation formula! This little term is what people call the regularisation term, it takes into account your "prior" knowledge of the problem.

For example, if you initialised your thetas from a Gaussian distribution (means of 0, variance of 1) you would end up with an L2 regularisation on theta. Try it and be amazed!

> *Notice that if we sample our thetas from the uniform distribution, the added term becomes a constant so we can get back to maximising the likelihood.*

Remark 3

Notice how engineering problems pushed us to find better notations or better optimisation procedures, surprisingly in machine learning, the basic probability theories are often not that complicated to grasp but the engineering feat to make them actually work are insane.

Just look at this new paper introducing a Self Normalising Neural Networks, 80 pages of mathematical calculus and demonstration just to get rid of an engineering problem called vanishing/exploding gradient in feedforward networks. That's insane!

Going further

I won't dive into this but one could wonder what happens if we stop fixing the hyperparameters of "m". In this case, one could rewrite our first equation as:

$$P(\mathcal{D}|\theta, m) = P(\mathcal{D}|\theta, \eta_1, \ldots, \eta_K, f_m)$$

The general formula including hyperparameters

With f_m (family of model) is now the only new fixed hyperparameter. All the “eta” representing the hyper-parameters of the model. Now, one can try to find an optimisation algorithm taking into account those “eta”. The main problem is usually that all hyperparameters are not in the same mathematical space (integers vs reals numbers, etc…) and so you must mix optimisation techniques.

· · ·

Anyway if you read this far, thank you!

I hope this was enlightening and interesting as this is my first ML note, don’t hesitate to give feedback, cheers! 🍺

· · ·

ML notes series

- Why the log likelihood? (this one :))
- Why the mean squared error?