Maximum likelihood for logistic regression

We first explain maximum likelihood in a simple but intuitive setting: flipping a biased coin. **Coin flipping does not directly have anything to do with machine learning**, but it *is* the simplest way to explain the concept of maximum likelihood.

Maximum likelihood for logistic regression can be understood by generalizing the coin flipping example, so that the "bias" is now influenced by input features $\boldsymbol{x}$, rather than being a constant. **That is why it's helpful to understand the coin flipping example, first.**

Maximum likelihood for flipping a biased coin

Let $y \in \{0, 1\}$ denote the observed outcome of a coin flip, where 0 means tails and 1 means heads.

Let $q \in [0, 1]$ denote the bias of the coin. For example, a coin with $q = 0.5$ is fair, and a coin with $q = 1$ always comes up heads.

A formula for the probability of a single coin flip outcome. The probability of a coin flip coming up heads ($y = 1$) or tails ($y = 0$), conditioned on the coin bias parameter q, is expressed as:

$$\begin{aligned} p(y = 1 \mid q) &= q \\ p(y = 0 \mid q) &= 1 - q \end{aligned}$$

The two cases above can be expressed in a single formula as:

$$p(y \mid q) = q^y (1 - q)^{1-y}$$

since if $y = 1$ it simplifies to q, and if $y = 0$ it simplifies to $1 - q$.

A formula for the probability of a sequence of coin flip outcomes. The joint probability of observing a specific sequence $y_1, \ldots, y_N$ of outcomes from N independent coin flips can then be expressed as:

$$p(y_1, \ldots, y_N \mid q) = \prod_{i=1}^{N} p(y_i \mid q) = \prod_{i=1}^{N} q^{y_i} (1 - q)^{1-y_i} \tag{1}$$

Formula (1) should be understood as "the probability of observing the sequence of outcomes $y_1, \ldots, y_N$ if the coin's bias were q."

When formula (1) is treated as a function of q, as when "training" q, it is a *likelihood function*.

Finding the maximum likelihood estimate of q, given a sequence of observed coin flip outcomes. If we know that a particular sequence of coin flip outcomes $y_1, \ldots, y_N$ contains n heads, then its likelihood (1) simplifies to:

$$p(y_1, \ldots, y_N \mid q) = q^n (1-q)^{N-n} \tag{2}$$

We all intuitively know that, if n tosses came up heads, the best estimate of the coin's bias is $q = \frac{n}{N}$. But why? In what sense is this considered the 'best'?

It is the best answer in the sense that $q = \frac{n}{N}$ happens to be the *maximum likelihood estimate* (MLE) of the unknown parameter q. The "maximum likelihood estimate of q" is the value of q that maximizes the likelihood (2).

Formalizing the maximum likelihood estimation problem for q. To find the maximum likelihood estimate of (2) programmatically, we must solve the following optimization problem:

$$\underset{q \in [0,1]}{\arg\max}\ q^n (1-q)^{N-n} \tag{3}$$

Differentiating (3) by q and setting the derivative to zero gives equation:

$$nq^{n-1}(1-q)^{N-n} - (N-n)q^n(1-q)^{N-n-1} = 0 \tag{4}$$

It is easy to show that $q = \frac{n}{N}$ satisfies equation (4). It can also be shown that (3) is concave in q, and so $\frac{n}{N}$ **is a maximum likelihood estimate for q.**

Conclusion. We have shown how the maximum likelihood principle provides an intuitively-correct estimate of a coin's bias, given a sequence of coin flip outcomes.

Generalizing the coin flipping example

So what is the connection to logistic regression?

In the coin flipping example, the training set for parameter q is a set of coin-flip outcomes $\mathcal{D} = \{y_i\}_{i=1}^N$, with no accompanying input features. That is because the coin's bias is assumed to be a constant, unaffected by the environment (temperature, wind speed, *etc*.).

So, if you were forced to predict the outcome of a coin flip, and you knew the coin's bias was $q > 0.5$, then **the most accurate prediction is always *heads*, no matter what!**

What if the coin were more complex? What if the coin's outcomes *might* be influenced by environmental factors? Well, you could measure the environmental factors accompanying a coin flip with outcome y, and put those measurements in a vector $\boldsymbol{x}$.

For example, you might have $\boldsymbol{x} = (x_1, x_2)$ where x_1 encodes air temperature and x_2 encodes the wind speed. The resulting dataset now comprises pairs $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ where $\boldsymbol{x}_i$ are the environmental conditions under which outcome y_i was produced.

In this new, more general setup, **whether you should predict heads or tails now depends on $\boldsymbol{x}$.** In other words, your estimate of the coin's bias should now be some sort of function $q(\boldsymbol{x}) \in [0, 1]$, rather than a constant $q \in [0, 1]$.

If you want to be able to fit this $q(\boldsymbol{x})$ function to data, it also needs some sort of parameters $\boldsymbol{w}$ that can be tuned. Your estimate of the coin's bias should therefore be some function $q(\boldsymbol{x}, \boldsymbol{w}) \in [0, 1]$. If $q(\boldsymbol{x}, \boldsymbol{w}) = 0.9$, it means that by predicting "heads" under condition $\boldsymbol{x}$ you will correctly predict the outcome 90% of the time, at least with respect to the training set.

Given a set of observations $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$, the likelihood function is a generalization of (1):

$$\begin{aligned} p(y_1, \ldots, y_N \mid \boldsymbol{x}_1, \ldots \boldsymbol{x}_N, \boldsymbol{w}) &= \prod_{i=1}^N p(y_i \mid \boldsymbol{x}_i, \boldsymbol{w}) \\ &= \prod_{i=1}^N q(\boldsymbol{x_i}, \boldsymbol{w})^{y_i} (1 - q(\boldsymbol{x_i}, \boldsymbol{w}))^{1-y_i} \end{aligned} \tag{5}$$

Here we have not specified the form of $q(\boldsymbol{x}, \boldsymbol{w})$. Logistic regression assumes one specific form, and a neural network might assume another specific form.

Whatever the form, we can ask: "**what is the maximum likelihood estimate of $\boldsymbol{w}$, given the data?**"

(Note, however, that since q is not constant, (5) cannot be simplified the way we simplified (2).)

Maximum likelihood for logistic regression

Logistic regression is used in classification scenarios, where the correct class is assumed to depend on input features $\boldsymbol{x}$. In the binary case, logistic regression must estimate the probability of $\boldsymbol{x}$ being classified as either positive (1, heads) or negative (0, tails).

The form of q that logistic regression assumes. For logistic regression, the probability of an input $\boldsymbol{x}$ belonging to the positive class (1, heads) is defined to take the specific form:

$$q(\boldsymbol{x}, \boldsymbol{w}) = \sigma(\boldsymbol{w}^T \boldsymbol{x})$$

where $\boldsymbol{w}^T \boldsymbol{x}$ is a linear transformation of $\boldsymbol{x}$, which is then fed into the logistic sigmoid function $\sigma(a) = \frac{1}{1+e^{-a}}$. The probability of $\boldsymbol{x}$ belonging to the negative class (0, tails) is $1 - \sigma(\boldsymbol{w}^T \boldsymbol{x})$.

That's it! Logistic regression can by thought of as a particular scheme for defining an input-sensitive estimate of a coin's bias. Of course, most regression tasks are not about coins—that's not the point. The point is to predict which class is most probable under a given input $\boldsymbol{x}$, even when that prediction will not be correct 100% of the time, akin to the inherent uncertainty of flipping a coin, even a heaily biased one.

The likelihood function of logisitic regression. By simple substitution from (5) we can see that logistic regression's likelihood function is:

$$p(y_1, \ldots, y_N \mid \boldsymbol{x}_1, \ldots \boldsymbol{x}_N, \boldsymbol{w}) = \prod_{i=1}^{N} \sigma(\boldsymbol{w}^T \boldsymbol{x}_i)^{y_i} (1 - \sigma(\boldsymbol{w}^T \boldsymbol{x}_i))^{1-y_i}$$

Finding a maximum likelihood estimate of $\boldsymbol{w}$. Finding a maximum likelihood estimate for logistic regression entails solving the following optimization problem:

$$\arg\max_{\boldsymbol{w}} \prod_{i=1}^{N} \sigma(\boldsymbol{w}^T \boldsymbol{x}_i)^{y_i} (1 - \sigma(\boldsymbol{w}^T \boldsymbol{x}_i))^{1-y_i} \tag{6}$$

Although it can be shown that (6) is concave in $\boldsymbol{w}$, is has no closed form solution. So, we must resort to gradient-based algorithms to maximize it. For practical reasons, we typically solve an equivalent optimization problem instead: minimize the negative log-likelihood, *i.e.*,

$$\arg\min_{\boldsymbol{w}} \sum_{i=1}^{N} -y_i \ln\big(\sigma(\boldsymbol{w}^T \boldsymbol{x}_i)\big) - (1 - y_i) \ln\big(1 - \sigma(\boldsymbol{w}^T \boldsymbol{x}_i)\big) \tag{7}$$

Recall three useful identities, by the chain rule of differentiation on $\ln x$, $\sigma(x)$, and $\boldsymbol{w}^T \boldsymbol{x}$:

$$\begin{aligned} \nabla_{\boldsymbol{w}}[\ln x] &= \frac{1}{x} \nabla_{\boldsymbol{w}}[x] \\ \nabla_{\boldsymbol{w}}[\sigma(x)] &= \sigma(x)(1 - \sigma(x)) \nabla_{\boldsymbol{w}}[x] \\ \nabla_{\boldsymbol{w}}\left[\boldsymbol{w}^T \boldsymbol{x}\right] &= \boldsymbol{x} \end{aligned}$$

We can derive the gradient of (7) with respect to $\boldsymbol{w}$ by applying these identies, in order:

$$
\begin{aligned}
&\nabla_{\boldsymbol{w}}\left[\sum_{i=1}^{N} -y_i \ln\big(\sigma(\boldsymbol{w}^T\boldsymbol{x}_i)\big) - (1-y_i)\ln\big(1-\sigma(\boldsymbol{w}^T\boldsymbol{x}_i)\big)\right] \\
&= \sum_{i=1}^{N} -y_i \nabla_{\boldsymbol{w}}\left[\ln\big(\sigma(\boldsymbol{w}^T\boldsymbol{x}_i)\big)\right] - (1-y_i)\nabla_{\boldsymbol{w}}\left[\ln\big(1-\sigma(\boldsymbol{w}^T\boldsymbol{x}_i)\big)\right] \\
&= \sum_{i=1}^{N} -\frac{y_i}{\sigma(\boldsymbol{w}^T\boldsymbol{x}_i)}\nabla_{\boldsymbol{w}}\left[\sigma(\boldsymbol{w}^T\boldsymbol{x}_i)\right] - \frac{1-y_i}{1-\sigma(\boldsymbol{w}^T\boldsymbol{x}_i)}\nabla_{\boldsymbol{w}}\left[1-\sigma(\boldsymbol{w}^T\boldsymbol{x}_i)\right] \\
&= \sum_{i=1}^{N} -y_i(1-\sigma(\boldsymbol{w}^T\boldsymbol{x}_i))\nabla_{\boldsymbol{w}}\left[\boldsymbol{w}^T\boldsymbol{x}_i\right] + (1-y_i)\sigma(\boldsymbol{w}^T\boldsymbol{x}_i)\nabla_{\boldsymbol{w}}\left[\boldsymbol{w}^T\boldsymbol{x}_i\right] \\
&= \sum_{i=1}^{N} \big(-y_i(1-\sigma(\boldsymbol{w}^T\boldsymbol{x}_i)) + (1-y_i)\sigma(\boldsymbol{w}^T\boldsymbol{x}_i)\big)\,\boldsymbol{x}_i \\
&= \sum_{i=1}^{N}(\sigma(\boldsymbol{w}^T\boldsymbol{x}_i) - y_i)\boldsymbol{x}_i
\end{aligned}
$$

By using this gradient formula within a gradient descent training algorithm, we can tune the parameters $\boldsymbol{w}$ so as to maximize the likelihood of the training data.

Conclusion: Logistic regression, when applied to binary classification, can be thought of as providing a feature-sensitive estimate of a coin's bias parameter. If $\sigma(\boldsymbol{w}^T\boldsymbol{x}) = 0.9$ under conditions $\boldsymbol{x}$, the logistic model is telling you that by predicting "heads" (positive class) it thinks you will be correct 90% of the time, at least with respect to the training data, even if though it estimates there still a 10% chance that you should have predicted "tails" (negative) instead.