The calculus you need for linear and logistic regression

These notes attempt to explain the multivariable calculus that you need to understand in order to implement logistic regression from scratch. We assume familiarity with matrix and vector notation. Our goal is to provide intuition rather than rigor, but we do give some guidance about how to make these ideas rigorous.

1 Defining the derivative in multivariable calculus

In single-variable calculus, the derivative of a function $f : \mathbb{R} \to \mathbb{R}$ is typically defined as follows:

$$f'(a) = \lim_{x \to a} \frac{f(x) - f(a)}{x - a}.$$

Visually, $\frac{f(x)-f(a)}{x-a}$ is the slope of the "secant line" connecting the points $\begin{bmatrix} a \\ f(a) \end{bmatrix}$ and $\begin{bmatrix} x \\ f(x) \end{bmatrix}$ on the graph of f. As x approaches a, the slope of the secant line approaches the slope of the tangent line to the graph of f at the point $\begin{bmatrix} a \\ f(a) \end{bmatrix}$.

Often, however, we are interested in functions which take a list of numbers as input and return a list of numbers as output. If $f : \mathbb{R}^n \to \mathbb{R}^m$, the above definition does not make sense, because we can't divide by a vector. Dividing by a vector is an undefined operation.

There is a slightly different way of thinking about the derivative, however, which has the advantage that it still makes perfect sense when $f : \mathbb{R}^n \to \mathbb{R}^m$. In high school algebra, we learn that a line with slope m which goes through the point $\begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$ has equation

$$y = y_0 + m(x - x_0).$$

This is called the "point-slope" form of the equation for a line. If $f : \mathbb{R} \to \mathbb{R}$ is differentiable at a, then the tangent line to the graph of f at the point $\begin{bmatrix} a \\ f(a) \end{bmatrix}$ has slope $m = f'(a)$. Thus, the equation for this tangent line is

$$y = f(a) + f'(a)(x - x_0).$$

When x is close to a, this tangent line provides a good approximation to the value of f:

$$f(x) \approx f(a) + f'(a)(x - a). \tag{1}$$

The point a is our "home base", and we are approximating the value of f at a nearby point x. This approximation is sometimes called the "tangent line approximation" to f near a, or alternatively the "local linear approximation" to f near a. It is called a "linear" approximation because the graph of the function on the right in (1) is a straight line. It is a "local" approximation because the approximation is only good when x is close to a. Another name which I use frequently for the approximation (1) is "Newton's approximation".

Local linear approximation is the key idea of calculus. While nonlinear functions can be rather complicated, linear functions are simple and easy to deal with. The **fundamental strategy of calculus** is to replace a nonlinear function f (difficult) with its local linear approximation (easy). When we do this, calculations are greatly simplified, and the approximation is often good enough that the results are still useful. Most of the great formulas of calculus (such as the chain rule and the product rule) can be discovered easily by using the tangent line approximation (1) at the crucial moment.

One of the advantages of equation (1) is that it still makes sense when $f : \mathbb{R}^n \to \mathbb{R}^m$. Let's parse Newton's approximation in this case:

$$\underbrace{f(x)}_{m\times 1} \approx \underbrace{f(a)}_{m\times 1} + \underbrace{f'(a)}_{?}\underbrace{(x-a)}_{n\times 1}.$$

Now x and a are points in $\mathbb{R}^n$, so $x - a \in \mathbb{R}^n$ (it is an $n \times 1$ column vector). And $f(x)$ and $f(a)$ are points in $\mathbb{R}^m$ (so they have shape $m \times 1$). What type of object should $f'(a)$ be in order for this equation to make sense? We can see that if $f'(a)$ is an $m \times n$ *matrix*, then Newton's approximation makes perfect sense.

$$\underbrace{f(x)}_{m\times 1} \approx \underbrace{f(a)}_{m\times 1} + \underbrace{f'(a)}_{m\times n}\underbrace{(x-a)}_{n\times 1}.$$

So, in multivariable calculus, $f'(a)$ is a *matrix*.

If $f : \mathbb{R}^n \to \mathbb{R}^m$ is differentiable at a, then $f'(a)$ is an $m \times n$ matrix.

Note: The above "definition" of $f'(a)$ is only an intuitive definition (we are saying that "$f'(a)$ is the matrix that appears in Newton's approximation"). However, it is possible to make the definition precise, as follows. To say that a function $f : \mathbb{R}^n \to \mathbb{R}^m$ is "differentiable" at a point $a \in \mathbb{R}^n$ means that there exists an $m \times n$ matrix $f'(a)$ such that the function

$$e(x) = f(x) - f(a) - f'(a)(x - a),$$

which tells us the error in Newton's approximation, satisfies

$$\lim_{x\to a} \frac{e(x)}{\|x - a\|} = 0. \tag{2}$$

This is a precise way of saying that the error in Newton's approximation is "small" when x is close to a. In fact, the above equation tells us that the error is small even when compared with $\|x - a\|$.

Fact: If f is differentiable at a then the matrix $f'(a)$ is unique. To prove this, suppose that A and B are real $m \times n$ matrices which satisfy

$$\lim_{x\to a} \frac{f(x) - f(a) - A(x-a)}{\|x - a\|} = 0 \qquad \text{and} \qquad \lim_{x\to a} \frac{f(x) - f(a) - B(x-a)}{\|x - a\|} = 0.$$

It follows that

$$0 = \lim_{x\to a} \frac{(A - B)(x - a)}{\|x - a\|}.$$

Let $e_1 = \begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix}^T$. Taking $x = a + te_1$, we see that

$$0 = \lim_{t\to 0} \frac{(A - B)(te_1)}{\|te_1\|} = (A - B)e_1.$$

This shows that the first column of $A - B$ is 0. Similarly, the remaining columns of $A - B$ are also 0. Hence, $A = B$. □

Definition: A function $f : \mathbb{R}^n \to \mathbb{R}^m$ which is differentiable at each point of $\mathbb{R}^n$ is called a "differentiable" function.

Note: If we define $\Delta x = x - a$, then Newton's approximation can be written equivalently as

$$f(a + \Delta x) \approx f(a) + f'(a)\Delta x. \tag{3}$$

The approximation is good when Δx is small. Sometimes this way of writing Newton's approximation is slightly more convenient.

If we also define

$$\Delta f = f(a + \Delta x) - f(a),$$

so that Δf is the change in the *output* of f when the *input* changes from a to $a + \Delta x$, then Newton's approximation can be written as

$$\Delta f \approx f'(a)\Delta x. \tag{4}$$

In words, if we change the input to f by a small amount, then the change in the output is proportional to the change in the input (to a good approximation, at least). The "constant of proportionality" (loosely speaking) is the matrix $f'(a)$.

Note: This matrix represents a linear transformation which is denoted $Df(a)$, and you might consider the linear transformation to be the more fundamental object. Here is what the mathematician Dieudonne had to say about the definition of the derivative.

> That presentation, which throughout adheres strictly to our general "geometric" outlook on analysis, aims at keeping as close as possible to the fundamental idea of calculus, namely the "local" approximation of functions by linear functions. In the classical teaching of calculus, this idea is immediately obscured by the accidental fact that, on a one-dimensional vector space, there is a one-to-one correspondence between linear forms and numbers, and therefore the derivative at a point is defined as a number instead of a linear form. This slavish subservience to the shibboleth of numerical interpretation at any cost becomes much worse when dealing with functions of several variables : one thus arrives, for instance, at the classical formula (8.9.2) giving the partial derivatives of a composite function, which has lost any trace of intuitive meaning, whereas the natural statement of the theorem is of course that the (total) derivative of a composite function is the composite of their derivatives (8.2.1), a very sensible formulation when one thinks in terms of linear approximations.

2 Two examples

In this section we'll look at two important examples of computing the derivative in multivariable calculus. In the examples below, a is a point in $\mathbb{R}^n$, and Δx is a small vector in $\mathbb{R}^n$.

2.1 Example 1: $f(x) = Ax$

Let A be an $m \times n$ matrix and let $f : \mathbb{R}^n \to \mathbb{R}^m$ be the function defined by

$$f(x) = Ax \quad \text{for all } x \in \mathbb{R}^n.$$

Then

$$\begin{aligned} f(a + \Delta x) &= A(a + \Delta x) \\ &= \underbrace{Aa}_{f(a)} + A\Delta x. \end{aligned}$$

Comparing this with Newton's approximation (3) reveals that

$$f'(a) = A.$$

Comment: In this example, the error $e(x)$ in Newton's approximation is 0, so certainly equation (2) is satisfied.

2.2 Example 2: $f(x) = \|x\|^2$

In the example below, we'll make use of the following facts about vector arithmetic. Let $u, v \in \mathbb{R}^n$ (so u and v are $n \times 1$ column vectors). The dot product of u and v is denoted $\langle u, v\rangle$. We have

1. $\langle u, v\rangle = u^T v = v^T u$.
2. $\|u\|^2 = \langle u, u\rangle$.
3. The expression $\|u+v\|^2$ can be expanded as

$$\begin{aligned}\|u+v\|^2 &= \langle u+v, u+v\rangle \\ &= \langle u,u\rangle + \langle u,v\rangle + \langle v,u\rangle + \langle v,v\rangle \qquad \text{(FOIL rule for dot products)} \\ &= \|u\|^2 + 2u^T v + \|v\|^2.\end{aligned}$$

   (Compare this with the formula $(a+b)^2 = a^2 + 2ab + b^2$ that we learn in high school algebra.)

We are ready to present our next example. Let $f : \mathbb{R}^n \to \mathbb{R}$ be defined by

$$f(x) = \|x\|^2.$$

Then

$$\begin{aligned}f(a+\Delta x) &= \|a+\Delta x\|^2 \\ &= \|a\|^2 + 2a^T\Delta x + \underbrace{\|\Delta x\|^2}_{\text{negligible}} \\ &\approx \|a\|^2 + 2a^T \Delta x.\end{aligned}$$

The term $\|\Delta x\|^2$ is negligibly small because we are squaring an already small number. Comparing the above approximation with Newton's approximation (3), we discover that

$$f'(a) = 2a^T.$$

Comment: In this example, the error in Newton's approximation is $e(x) = \|x-a\|^2$, which I claimed is "negligibly small". To be more precise, the condition (2) is indeed satisfied because

$$\lim_{x\to a} \frac{e(x)}{\|x-a\|} = \lim_{x\to a} \frac{\|x-a\|^2}{\|x-a\|} = \lim_{x\to a} \|x-a\| = 0.$$

3 The chain rule

Let's use Newton's approximation to discover the chain rule. Suppose that $h : \mathbb{R}^n \to \mathbb{R}^m$ is differentiable at a and $g : \mathbb{R}^m \to \mathbb{R}^k$ is differentiable at $h(a)$. Let $f : \mathbb{R}^n \to \mathbb{R}^k$ be the function defined by

$$f(x) = g(h(x)) \qquad \text{for all } x \in \mathbb{R}^n$$

If $\Delta x \in \mathbb{R}^n$ is small, then we have

$$\begin{aligned}f(a+\Delta x) &= g(h(a+\Delta x)) \\ &\approx g(h(a) + h'(a)\Delta x) \\ &\approx \underbrace{g(h(a))}_{f(a)} + g'(h(a))h'(a)\Delta x.\end{aligned}$$

We used Newton's approximation twice, first on h and then on g. Look at how the chain rule just pops out! Comparing this result with Newton's approximation (3) reveals that

$$f'(a) = g'(h(a))h'(a).$$

It's easy to discover the chain rule by using Newton's approximation. The above derivation can be made into a rigorous proof by keeping track of the error terms when using Newton's approximation.

Another explanation: Here is a slightly different, and perhaps even more direct, way to understand the chain rule. The change in the input is Δx. The change in the output of h is

$$\Delta h \approx h'(a)\Delta x.$$

The corresponding change in the output of g is

$$\Delta g \approx g'(h(a))\Delta h \approx g'(h(a))h'(a)\Delta x.$$

In other words, when the input changes by Δx, we can first multiply by $h'(a)$ to get the change in the output of h, and then multiply by $g'(h(a))$ to get the change in the output of g, and so we find that change in the output of f is

$$\Delta f \approx g'(h(a))h'(a)\Delta x.$$

Comparing this with equation (4), we discover that

$$f'(a) = g'(h(a))h'(a).$$

4 The gradient vector

Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is differentiable at a point $a \in \mathbb{R}^n$. Then $f'(a)$ is a $1 \times n$ matrix (also called a "row vector"). If we take this row vector and flip it sideways (that is, transpose it) we get a column vector which is called the gradient of f at a, and which is denoted $\nabla f(a)$:

$$\nabla f(a) = f'(a)^T.$$

When I define the gradient in this way, I'm following the convention that the gradient is a column vector rather than a row vector. This convention is common in the world of applied math and optimization, but it's not a universally adopted convention. For example, Terence Tao calls the row vector $f'(a)$ the "gradient" of f at x in his book Analysis II.

Newton's approximation $f(a + \Delta x) \approx f(a) + f'(a)\Delta x$ can be expressed in terms of the gradient vector as follows:

$$f(a + \Delta x) \approx f(a) + \nabla f(a)^T \Delta x = f(a) + \langle \nabla f(a), \Delta x \rangle.$$

The gradient vector $\nabla f(x)$ has a nice geometric interpretation, which we will now explain. Suppose that we start at the location a and move a short distance t in the direction of a unit vector u. Our new location is $a + tu$, and the value of f at our new location is

$$f(a + tu) \approx f(a) + \langle \nabla f(a), tu \rangle = f(a) + t\langle \nabla f(a), u \rangle.$$

Question: In which direction should the unit vector u be pointing if we want the value of f to increase as much as possible? This is equivalent to asking for which unit vector u is the quantity $\langle \nabla f(a), u \rangle$ as large as possible.

Answer: Recall that

$$\langle \nabla f(a), u \rangle = \|\nabla f(a)\| \underbrace{\|u\|}_{1} \cos(\theta) = \|\nabla f(a)\| \cos(\theta)$$

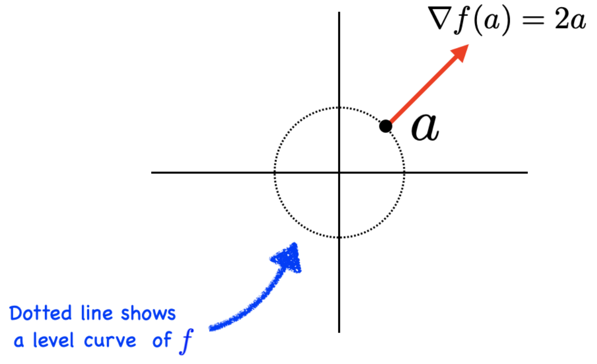

Figure 1: We are visualizing a gradient vector $\nabla f(a)$ for the function $f(x) = \|x\|^2$. The gradient vector points in the direction of steepest ascent, which in this example is directly away from the origin.

where θ is the angle between $\nabla f(a)$ and u. The largest possible value of $\cos(\theta)$ is 1, which occurs when $\theta = 0$, which means that u points in the same direction as $\nabla f(a)$. We conclude that

the gradient vector $\nabla f(a)$ points in the *direction of steepest ascent.*

In other words, if we are located at a and we want the value of f to increase as rapidly as possible, we should move in the same direction as $\nabla f(a)$.

If we want the value of f to *decrease* as rapidly as possible, we should move in the opposite direction.

The negative gradient vector $-\nabla f(a)$ points in the direction of steepest *descent.*

Example: Let $f : \mathbb{R}^2 \to \mathbb{R}$ be defined by

$$f(x) = \|x\|^2.$$

Visually, $\|x\|$ is the distance from the origin to the point x, and so $f(x)$ tells us the squared distance from the origin to x.

If you are located at a point a, which way should you move to make f increase as rapidly as possible? Of course, without knowing anything about vector calculus, you would say that you should move directly away from the origin. And indeed, this intuition is consistent with what we have learned about vector calculus: $f'(a) = 2a^T$ and $\nabla f(a) = 2a$, which is a vector that points away from the origin. (See figure 1.)

5 Minimizing a differentiable function $f : \mathbb{R}^n \to \mathbb{R}$

Many important problems in applied math or engineering can be formulated as optimization problems. We want to get the most bang for our buck. We want to find a set of parameters which minimizes some cost. In a regression problem, for example, we might want to find a list of parameters for a neural network which minimizes the mean squared error for our predicted target values. In this section we'll learn two strategies for minimizing a differentiable function $f : \mathbb{R}^n \to \mathbb{R}$.

Definition: Let $f : \mathbb{R}^n \to \mathbb{R}$. To say that a point $a \in \mathbb{R}^n$ is a "local minimizer" of f means (roughly speaking) that $f(a) \leq f(x)$ for all nearby points x. More precisely, there is a ball B centered at a such that $f(a) \leq f(x)$ for all $x \in B$.

Definition: To say that a point $a \in \mathbb{R}^n$ is a "global minimizer" of f means that $f(a) \leq f(x)$ for all $x \in \mathbb{R}^n$.

Any global minimizer of f is automatically a local minimizer, of course.

Fact: Suppose that a is a local minimizer for a function $f : \mathbb{R}^n \to \mathbb{R}$. If f is differentiable at a, then $\nabla f(a) = 0$.

Reason: We'll argue by contradiction. Suppose $\nabla f(a) \neq 0$ and t is a small positive number. The value of f at the nearby point $x = a - t\nabla f(a)$ is

$$f(x) \approx f(a) + \langle \nabla f(a), -t\nabla f(a) \rangle = f(a) - t\|\nabla f(a)\|^2 < f(a).$$

This contradicts the fact that a is a local minimizer of f. $\square$

This argument can be summarized as follows: if $\nabla f(a)$ were not equal to 0, then we could reduce the value of f by taking a short step in the negative gradient direction. That would contradict the fact that a is a local minimizer of f.

5.1 Strategy 1: Set the gradient equal to 0 and solve for a

If a differentiable function $f : \mathbb{R}^n \to \mathbb{R}$ has a global minimizer, the above fact gives us a strategy for how to find it: we write down the equation

$$\nabla f(a) = 0 \tag{5}$$

and solve for a. If there is only one solution, it must be the global minimizer of f. If equation (5) has more than one solution, we can check each of them to see which one gives us the least value of f.

As we'll see in section 6 below, this strategy works very well in the important case that $f(x) = \|Ax - b\|^2$ (for some matrix $A \in \mathbb{R}^{m\times n}$ and vector $b \in \mathbb{R}^m$). However, a potential difficulty with this strategy is that for more complicated functions the equation $\nabla f(a) = 0$ is typically a nonlinear system of equations, and solving such a nonlinear system of equations might not be easy.

5.2 Strategy 2: Repeatedly move in the direction of steepest descent

An alternative strategy for minimizing a differentiable function $f : \mathbb{R}^n \to \mathbb{R}$ is to repeatedly take short steps in the direction of steepest descent. Suppose that our current location is $x^k \in \mathbb{R}^n$. (The k here is a superscript, not an exponent.) We choose a small positive number t and move to the new location

$$x^{k+1} = x^k - t\nabla f(x^k).$$

When moving from x^k to the new point x^{k+1}, we are taking a short step in the negative gradient direction, which is the direction of steepest descent. When we do this, the value of f decreases a little bit (assuming that we take a sufficiently small step). The **gradient descent** algorithm repeatedly takes short steps in the negative gradient direction, until eventually converging to a local minimizer of f.

The parameter $t > 0$ is a "step size" that controls how large our steps are. A larger value of t is more aggressive. If t is too large, then the gradient descent iteration might not converge at all. A small value of t is safe, conservative, and leads to slow convergence. There are various strategies for choosing a good value for t, including adaptive step size strategies where the value of t changes from one iteration to another according to a certain rule. The simplest approach, however, is to use a fixed step size t which is chosen by trial and error.

6 Solving least squares problems

Let $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^n$. In linear algebra, we often want to solve the linear system of equations $Ax = b$. If b is not in the column space of A, then there is no solution. What should we do then? Should we just give up? No! Although we can't solve $Ax = b$ exactly, we can do the next best thing: we'll find a vector x such that $\|Ax - b\|$ is as small as possible. Equivalently, we choose x to minimize the function

$$f(x) = \|Ax - b\|^2.$$

(Squaring the norm of the residual here makes the math work out more nicely, and it does not change the optimal choice of x.) The problem of finding a vector x which minimizes this function f is called a "least squares problem", and a point x which minimizes f is called a "least squares approximate solution" to $Ax = b$.

How can we find a point x which minimizes f? We can use the strategy described in section 5.1 — we set the gradient of f equal to 0 and solve for x. Let's compute the derivative of f using the chain rule. Notice that

$$f(x) = g(h(x))$$

where

$$h(x) = Ax - b \quad \text{and} \quad g(u) = \|u\|^2.$$

We have already discovered that the derivatives of g and h are

$$h'(x) = A \quad \text{and} \quad g'(u) = 2u^T.$$

By the chain rule, the derivative of f is

$$\begin{aligned} f'(x) &= g'(h(x))h'(x) \\ &= 2(Ax - b)^T A. \end{aligned}$$

Thus, the gradient of f at a point x is

$$\nabla f(x) = f'(x)^T = 2A^T(Ax - b).$$

Setting the gradient equal to 0, and multiplying through by 1/2, we find that

$$A^T(Ax - b) = 0$$

or equivalently

$$A^T Ax = A^T b. \tag{6}$$

This is a famous equation. This linear system of equations is called the "normal equations", and it is the key to solving least squares problems. We can solve for x using a technique such as Gaussian elimination.

Comment: Because the function f must have a minimizer, which necessarily satisfies $f'(x) = 0$, the equation (6) is guaranteed to have a solution. If the coefficient matrix $A^T A$ is invertible, then the solution is unique.

7 Linear regression

In a regression problem, we are given a training dataset consisting of feature vectors $x_1, \ldots, x_N \in \mathbb{R}^d$ and corresponding target values $y_1, \ldots, y_N \in \mathbb{R}$. Our goal is to find a "prediction function" $f : \mathbb{R}^d \to \mathbb{R}$ which has the property that

$$f(x_i) \approx y_i \quad \text{for } i = 1, \ldots, N.$$

We hope that our prediction function f will make continue to make good predictions on data which is not included in the training dataset, but there is no guarantee that this will be the case.

In linear regression, we assume that f is a linear function (or technically, an affine function). In other words, we assume that

$$f(\underset{\substack{\uparrow \\ \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix}}}{x}) = \beta_0 + \beta_1 x_1 + \cdots + \beta_d x_d$$

for some scalars $\beta_0, \beta_1, \ldots, \beta_d$.

How should we choose the coefficients β_j? The most common approach is to choose them to minimize the mean squared error

$$L(\underset{\substack{\uparrow \\ \begin{bmatrix}\beta_0 \\ \beta_1 \\ \vdots \\ \beta_d\end{bmatrix}}}{\beta}) = \frac{1}{N}\sum_{i=1}^{N}\Big(f(\underset{\substack{\uparrow \\ \begin{bmatrix}x_{i1} \\ \vdots \\ x_{id}\end{bmatrix}}}{x_i}) - y_i\Big)^2 = \frac{1}{N}\sum_{i=1}^{N}(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_d x_{id} - y_i)^2.$$

The mean squared error can be written concisely using matrix and vector notation as follows:

$$L(\beta) = \frac{1}{N}\|X\beta - y\|^2$$

where

$$X = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1d} \\ 1 & x_{21} & \cdots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N1} & \cdots & x_{Nd} \end{bmatrix}, \quad \beta = \begin{bmatrix}\beta_0 \\ \beta_1 \\ \vdots \\ \beta_d\end{bmatrix}, \quad \text{and} \quad y = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}.$$

Notice that minimizing $L(\beta)$ is a least squares problem, so we can minimize L using the same approach that we used in section 6. The gradient of L is

$$\nabla L(\beta) = \frac{1}{N}X^T(X\beta - y).$$

We can minimize $L(\beta)$ by setting the gradient equal to 0 and solving for β. The resulting system of equations

$$X^T(X\beta - y) = 0 \quad \text{or equivalently} \quad X^T X\beta = X^T y$$

is called the "normal equations", as discussed in section 6.

8 Logistic regression

In a classification problem with two classes (called class 0 and class 1), we are given a training dataset consisting of feature vectors $x_1, \ldots, x_N \in \mathbb{R}^d$ and corresponding labels $y_1, \ldots, y_N \in \{0, 1\}$. Our goal is to find a "prediction function" $f : \mathbb{R}^d \to [0, 1]$ such that $f(x_i) \approx y_i$ for $i = 1, \ldots, N$. Given a feature vector $x \in \mathbb{R}^d$, we think of $f(x)$ as being an estimated probability that the example described by x belongs to class 1. We can think of a corresponding label $y \in \{0, 1\}$ as being a "ground truth" probability which reflects certainty about whether or not the example described by x belongs to class 1.

In logistic regression, we take the prediction function f to have the form

$$f(\underset{\substack{\uparrow \\ \begin{bmatrix}x_1 \\ \vdots \\ x_d\end{bmatrix}}}{x}) = \sigma(\beta_0 + \beta_1 x_1 + \cdots + \beta_d x_d)$$

where $\sigma : \mathbb{R} \to \mathbb{R}$ is the "sigmoid function" defined by

$$\sigma(u) = \frac{e^u}{1 + e^u}.$$

Notice that $0 < \sigma(u) < 1$, so the output of σ can be interpreted as a probability. The sigmoid function is useful in machine learning because it converts real numbers into probabilities. It is arguably the simplest and most elegant smooth function which does this job for us. (Just try to think of anything simpler.)

Our goal is to choose the coefficients $\beta_0, \beta_1, \dots, \beta_d$ so that the estimated probability $f(x_i)$ tends to agree well with the ground truth probability y_i (for $i = 1, \dots, N$). To measure the agreement between an estimated probability q and a ground truth probability p, we will use the "binary cross-entropy" loss function defined by

$$\ell(p, q) = -p \log(q) - (1 - p) \log(1 - q).$$

The inputs p and q are required to satisfy $0 \leq p \leq 1$ and $0 < q < 1$. Although the formula for $\ell(p, q)$ looks strange and unmotivated, it is in some sense a very natural and beautiful way to measure the agreement between an estimated probability q and a ground truth probability p. If q agrees closely with p, then $\ell(p, q)$ is small. If q does not agree well with p, then $\ell(p, q)$ is large. For any given $p \in (0, 1)$, the value of q that minimizes $\ell(p, q)$ is $q = p$. One way to discover this function $\ell(p, q)$ is to view the label of example i as a Bernoulli random variable Y_i which is arbitrarily assumed to satisfy

$$P(Y_i = 1) = \sigma(\beta_0 + \beta_1 x_1 + \cdots + \beta_d x_d),$$

and then use maximum likelihood estimation to estimate the coefficients $\beta_0, \beta_1, \dots, \beta_d$. When we work out the details, the binary cross-entropy loss function appears in the log-likelihood formula.

The coefficients β_j (for $j = 0, \dots, d$) are chosen to minimize the average binary cross-entropy

$$L(\underset{\begin{array}{c}\uparrow\\ \begin{bmatrix}\beta_0\\ \beta_1\\ \vdots\\ \beta_d\end{bmatrix}\end{array}}{\beta}) = \frac{1}{N} \sum_{i=1}^{N} \ell(y_i, f(x_i)).$$

We can minimize L using the gradient descent algorithm. But first, we must derive a formula for the gradient of L.

8.1 Computing the gradient of L

The formula for f can be written more concisely as

$$f(\underset{\begin{array}{c}\uparrow\\ \begin{bmatrix}x_1\\ \vdots\\ x_d\end{bmatrix}\end{array}}{x}) = \sigma(\hat{x}_i^T \beta) \quad \text{where } \hat{x} \text{ is the "augmented" feature vector} \quad \hat{x} = \begin{bmatrix}1\\ x_1\\ \vdots\\ x_d\end{bmatrix}.$$

With this notation, the average cross-entropy can be written as

$$\begin{aligned} L(\beta) &= \frac{1}{N} \sum_{i=1}^{N} \ell(y_i, \sigma(\hat{x}_i^T \beta)) \\ &= \frac{1}{N} \sum_{i=1}^{N} L_i(\hat{x}_i^T \beta) \end{aligned}$$

where $L_i : \mathbb{R} \to \mathbb{R}$ is defined by

$$L_i(u) = \ell(y_i, \sigma(u)).$$

By the chain rule,

$$L'(\beta) = \frac{1}{N} \sum_{i=1}^{N} L_i'(\hat{x}_i^T \beta) \hat{x}_i^T. \tag{7}$$

Thus, computing the derivative of L has been reduced to computing the derivative of the L_i. But, L_i is a function of a single variable, so computing $L_i'(u)$ is merely a problem from single-variable calculus.

Here is my hint for computing $L_i'(u)$: simplify the formula for $L_i(u)$ as much as possible first. Using the formulas for $\ell(p, q)$ and $\sigma(u)$, we have

$$\begin{aligned} L_i(u) &= -y_i \log\left(\frac{e^u}{1+e^u}\right) - (1-y_i)\log\left(1 - \frac{e^u}{1+e^u}\right) \\ &= -y_i \log\left(\frac{e^u}{1+e^u}\right) - (1-y_i)\log\left(\frac{1}{1+e^u}\right) \\ &= -y_i \log(e^u) + y_i \log(1+e^u) + (1-y_i)\log(1+e^u) \\ &= -y_i u + \log(1+e^u). \end{aligned}$$

Look how much that simplified! We are now ready to compute $L_i'(u)$ using single-variable calculus:

$$L_i'(u) = -y_i + \frac{e^u}{1+e^u} = \sigma(u) - y_i. \tag{8}$$

This is a delightfully simple result. It has an intuitive meaning also: if the estimated probability $\sigma(u)$ agrees perfectly with the ground truth probability y_i, then the derivative is 0, suggesting that no change to the value of u is necessary. The simplicity of the formula (8) makes me feel that the sigmoid function and the binary cross-entropy loss function are meant to be together, united in the beautiful function $L_i(u)$. This is why Pytorch provides the `BCEWithLogits` loss function.

Combining equation (8) with equation (7), we see that

$$L'(\beta) = \frac{1}{N} \sum_{i=1}^{N} (\sigma(\hat{x}_i^T \beta) - y_i)\hat{x}_i^T.$$

Thus,

$$\nabla L(\beta) = L'(\beta)^T = \frac{1}{N} \sum_{i=1}^{N} (\sigma(\hat{x}_i^T \beta) - y_i)\hat{x}_i.$$

With this formula for the gradient of L, we are now ready to implement logistic regression from scratch, using gradient descent to find a vector $\beta \in \mathbb{R}^{d+1}$ which minimizes $L(\beta)$.