Linear Predictors, Part 2

September 27, 2021

[Continued from Part 1]

2.2 The Logistic Function

The (as of yet unknown) separating hyperplane χ of a binary logistic-regression classifier defines the locus of indifference between the two classes 0 and 1: If a point $\mathbf{x}$ is on χ (that is, if $\Delta(\mathbf{x}) = 0$), then the point could well be in either class. It then makes sense to map $\Delta(\mathbf{x}) = 0$ to a probability $p = 1/2$ of any $\mathbf{x}$ on χ belonging to class 1. As $\Delta(\mathbf{x})$ increases, point $\mathbf{x}$ gets deeper into the positive half-space, and its probability of belonging to class 1 increases: The bigger $\Delta(\mathbf{x})$, the closer p should be to 1. Similarly, as $\Delta(\mathbf{x})$ becomes negative and decreases, p should tend to zero. Pick a particular $\mathbf{x} = \mathbf{x}_0$ (on either side of χ, or on it), and let

$$\Delta(\mathbf{x}_0) = \Delta_0$$

be its signed distance from χ. Consider the hyperplane through $\mathbf{x}_0$ that is parallel to χ. All points $\mathbf{x}$ on that hyperplane have

$$\Delta(\mathbf{x}) = \Delta_0 \ ,$$

and therefore the probability p of $\mathbf{x}$ being in class 1 should be the same for all those points.

To summarize, the monotonic function f that maps $\Delta(\mathbf{x})$ to p is constant on every hyperplane parallel to χ, is equal to $1/2$ on χ, and tends to 0 or 1 as $\Delta(\mathbf{x})$ tends to negative or positive infinity. A simple function with these properties is the so-called *logistic function*,[1]

$$f(\Delta) \stackrel{\mathsf{def}}{=} \frac{1}{1 + e^{-\Delta}} \tag{1}$$

whose plot is shown in Figure 1. While there are justifications for the specific form of this function, they all require assumptions about the probability distribution of the data. Without these assumptions, our justification is merely mathematical simplicity.

Consider once more what we have done: We *assumed* that there is a single scalar quantity

$$\Delta(\mathbf{x}) \stackrel{\mathsf{def}}{=} \frac{b + \mathbf{w}^T\mathbf{x}}{\|\mathbf{w}\|}$$

that captures everything we need to know about $\mathbf{x}$ in order to assign label $y = 0$ or $y = 1$ to $\mathbf{x}$. This is a very strong assumption, and whether it is at least approximately warranted depends on the specific problem. In the definition of $\Delta(\mathbf{x})$, the parameters b and $\mathbf{w}$ are unknown, and are to be

[1] Verify that the properties mentioned hold for the logistic function.

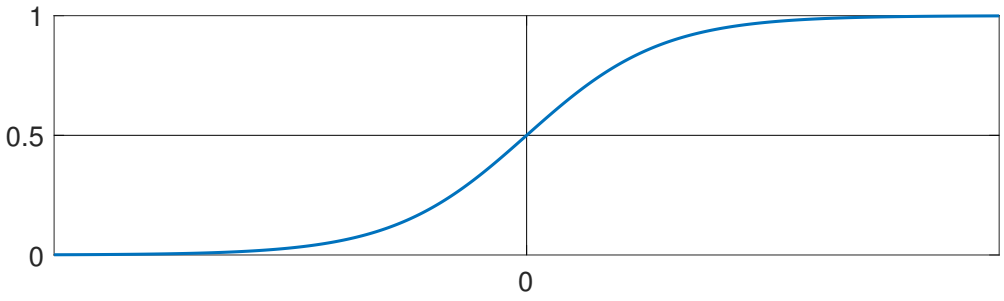

Figure 1: The logistic function.

determined from data through training. Since Δ is an affine function of $\mathbf{x}$, it is constant on every hyperplane parallel to the hyperplane χ with equation

$$b + \mathbf{w}^T\mathbf{x} = 0 \; .$$

Of course, we could consider

$$a(\mathbf{x}) \stackrel{\text{def}}{=} b + \mathbf{w}^T\mathbf{x} \tag{2}$$

instead of $\Delta(\mathbf{x})$, as the two functions differ by a multiplicative factor. If these factors were important to us, we would have to introduce a scaling factor in f:

$$\frac{1}{1 + e^{-\Delta/c}}$$

and then devise some procedure to determine an optimal value for c. However, the logistic function is composed with an activation. Rather than normalizing the activation to obtain a geometric distance and then introducing a scaling factor into the logistic function, it is simpler to allow for an unnormalized activation, and then dispense of the scaling factor c in the logistic function. In summary, we define the *score* of a logistic-regression classifier as the function

$$s(\mathbf{x}) \stackrel{\text{def}}{=} f(a(\mathbf{x})) = \frac{1}{1 + e^{-b-\mathbf{w}^T\mathbf{x}}} \tag{3}$$

(without division by the norm of $\mathbf{w}$ in the exponent).

The "unnormalized distance" $a(\mathbf{x})$ defined in equation 2 is technically no longer a distance, not even an unsigned one, and is called the *activation*. The score $s(\mathbf{x})$ could be viewed formally as a probability, since it is a number between 0 and 1, and this is how we interpreted it in our derivation. However, the various arbitrary choices we made along the way should convince you that this interpretation is a bit of a stretch. Hence the word "score" rather than "probability."

The idea now is that if we find an efficient way to fit a score (that is, determine b and $\mathbf{w}$) that "explains" our training set T, then we have a way to define a linear classifier: Compute the score of a point and compare it to threshold 1/2 to determine the point's class. "Explaining" here means that points in class 1 should ideally have scores greater than 1/2 and points in class 0 should have scores below that threshold. However, since training sets are rarely linearly separable, it may be impossible to do this exactly. Instead, we define a loss that measures the cost we incur for a poor score, and find the parameters b and $\mathbf{w}$ that minimize the loss. Since our score is a sigmoid, we

reduce the severity of the issue illustrated in Figure 2 of Part 1 on linear predictors, namely, that a linear score would overly emphasize the influence of points that are distant from the decision boundary. With a sigmoid score rather than a linear one, points that are far from χ have scores that are close to either 0 or 1. Because of this, as long as they are on the correct side of χ, these points incur very small losses, and they therefore have little influence on the position of χ.

We discuss a convenient loss function next.

2.3 The Cross-Entropy Loss

The performance of a classifier is typically measured by how many mistakes it makes. If y is the true label of data point $\mathbf{x}$ and $\hat{y} = \hat{h}(\mathbf{x})$ is the class returned by the classifier $\hat{h}$, the zero-one loss is then

$$\ell_{0\text{-}1}(y, \hat{y}) = \begin{cases} 0 & \text{if } y = \hat{y} \\ 1 & \text{otherwise} \end{cases}$$

as we saw in an earlier note. Unfortunately, minimizing the zero-one loss is exactly equivalent to finding the optimal separating hyperplane, which, as mentioned earlier, is an NP-hard problem except in the unlikely event that the training set is linearly separable.

Our way to avoid this difficulty is to define a loss function ℓ that is a differentiable function of the *score* $p = s(\mathbf{x})$, rather than a function of $\hat{y}$. Notice the shell game: The zero-one loss is really what we are after, but minimizing that would lead to a complex combinatorial problem, because $\hat{y}$ (and therefore $\ell_{0\text{-}1}$) is a discontinuous function of the score that has zero gradient almost everywhere. Therefore, there is no useful gradient to drive the parameters b and $\mathbf{w}$ of the predictor towards values that achieve a small risk. In order to find a solution efficiently, we instead use the score $p = s(\mathbf{x})$ as a proxy for $\hat{y}$, and define a loss function in terms of that:

$$\ell(y, p) \quad \text{rather than} \quad \ell(y, \hat{y}) \; .$$

This substitution is reasonable. After all, $\hat{y} \in \{0, 1\}$ and $p \in [0, 1]$, and a good classifier assigns scores close to 1 to data points with label 1, and scores close to 0 to data points with label 0.

Since the score function s is differentiable in the parameters b and $\mathbf{w}$, we can compute the gradient ∇L_T of the risk[2]

$$L_T(b, \mathbf{w}) = \frac{1}{N} \sum_{n=1}^{N} \ell(y_n, s(\mathbf{x}_n \,;\, b, \mathbf{w})) \tag{4}$$

with respect to b and $\mathbf{w}$. In addition, since s is a strictly monotonic function in any of its parameters[3], the gradient of L_T is rarely zero. We can then use any gradient descent method to search for parameters $b, \mathbf{w}$ that make L_T small. For logistic regression, we can do even better: We can pick the form of the loss ℓ to make the risk L_T a *convex* function in the parametetrs, so that if we find a minimum we know that the minimum is global.

The rest of this Section introduces and describes a loss function with these properties, called the cross-entropy loss, and proves convexity of the resulting risk function. Appendix A establishes a connection between this measure of loss and probabilistic and information-theoretic considerations. While these connections are conceptually interesting, the true justification for the cross-entropy loss is that it makes the risk convex and its gradient simple to compute, as we will see.

[2]The risk is a function of the classifier h, which is in turn a function of the classifier parameters b and $\mathbf{w}$, so we write more directly $L_T(b, \mathbf{w})$, as usual, rather than $L_T(h)$. We also make the dependence of s on b and $\mathbf{w}$ explicit.

[3]Verify this by direct differentiation.

The *cross-entropy loss* of assigning score p to a data point with true label y is defined as follows:

$$\ell(y,p) \stackrel{\text{def}}{=} \begin{cases} -\log p & \text{if } y = 1 \\ -\log(1-p) & \text{if } y = 0. \end{cases}$$

Since y is binary, it can be used as a "switch" to rewrite this function like this:

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

In these expressions, the base of the logarithm is unimportant, as long as the same base is used throughout, because logarithms in different bases differ by a multiplicative constant. In Appendix A, the logarithm base 2 is used, as is customary in information theory.

Figure 2 shows a plot of the function ℓ. The domain is the cross product

$$\{0,1\} \times [0,1] \ ,$$

that is, two segments on the plane (left in the Figure), and the right panel in the Figure shows the values of the loss on these two segments in corresponding colors. From either formula or Figure, we see that

$$\ell(1,p) = \ell(0,1-p) \ ,$$

reflecting the symmetry between the two labels and their scores. The two curves therefore meet when $p = 1/2$:

$$\ell(1,1/2) = \ell(0,1/2) = -\log(1/2) \ .$$

If base-2 logarithms are used, this value is 1. With natural logarithms it is $\ln 2 \approx 0.693$.

The cross-entropy function is a plausible measure of loss: When the true label y is one (blue), no cost is incurred when the score p is one as well, since p scores the hypothesis that $y = 1$. As p decreases for this value of the true label, a bigger and bigger cost is incurred, and as $p \to 0$ the cost grows to infinity. Even this unbounded loss makes sense: The score p tends to zero when the distance $\Delta(\mathbf{x})$ between the data point and the separating hyperplane χ approaches $-\infty$ (see equation 1). When this happens, $\mathbf{x}$ tends to be infinitely far away from the separating hyperplane *and on the wrong side of it*, because points with a negative Δ (or score p less than $1/2$) are classified with label 0 but the true label y is 1. Thus, the estimated label of that data point is *way* wrong, and an infinite cost is plausible. Similar considerations hold when $y = 0$.

2.4 Convexity of the Risk Function

We saw in an earlier note that a function $f(\mathbf{v})$ is (weakly) convex if for every pair of points $\mathbf{v}, \mathbf{v}'$ in the domain of f the graph of f is never above the segment that joins the two graph points $(\mathbf{v}, f(\mathbf{v}))$ and $(\mathbf{v}', f(\mathbf{v}'))$:

$$f(u\mathbf{v} + (1-u)\mathbf{v}') \ \le \ uf(\mathbf{v}) + (1-u)f(\mathbf{v}') \quad \text{for all } u \in [0,1].$$

We also saw that to check that a scalar function $f \ : \ \mathbb{R}^m \to \mathbb{R}$ is (weakly) convex you check that its Hessian H is positive semidefinite. This means that for every $\mathbf{v} \in \mathbb{R}^m$ we have

$$\mathbf{v}^T H \mathbf{v} \ \ge \ 0 \ .$$

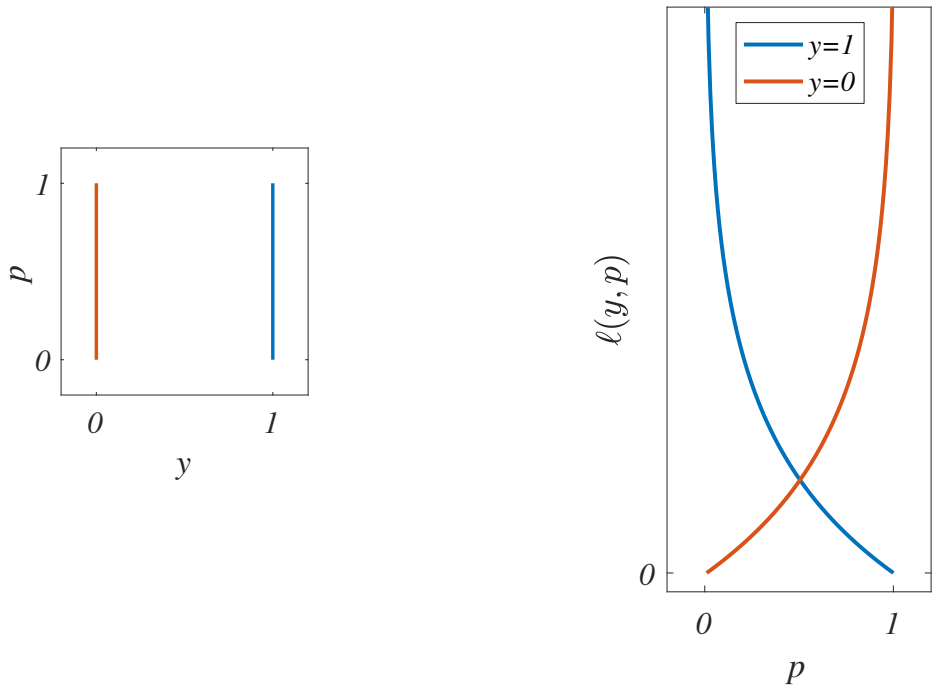

Figure 2: (Left) The two colored segments are the domain of any score-based loss function for the binary logistic-regression classifier. (Right) The cross-entropy loss. The two curves are defined on the segments of corresponding colors on the left. They asymptote to infinity for $p \to 0$ (when $y = 1$, blue) and for $p \to 1$ (when $y = 0$, red).

We use this concept to verify that the risk function in equation 4 is weakly convex in the vector

$$\mathbf{v} = \begin{bmatrix} b \\ \mathbf{w} \end{bmatrix}$$

of classifier parameters. In the process, we compute the gradient of the risk function as well, and we will be able to use the gradient to search for the parameter vector $\mathbf{v}$ that minimizes the risk. Because of convexity, any minimum is a global minimum.

Appendix B shows that the gradient and Hessian of the risk L_T defined in equation 4 are

$$\begin{aligned} \nabla L_T(\mathbf{v}) &= \frac{1}{N}\sum_{n=1}^{N}[s(\mathbf{x}_n\ ;\ \mathbf{v}) - y_n]\begin{bmatrix} 1 \\ \mathbf{x}_n \end{bmatrix} \\ H_{L_T}(\mathbf{v}) &= \frac{1}{N}\sum_{n=1}^{N} s(\mathbf{x}_n\ ;\ \mathbf{v})\ [1 - s(\mathbf{x}_n\ ;\ \mathbf{v})]\begin{bmatrix} 1 \\ \mathbf{x}_n \end{bmatrix}\begin{bmatrix} 1 & \mathbf{x}_n \end{bmatrix} \end{aligned}$$

and that the Hessian is positive semidefinite, so that the risk is a convex function.

Using any gradient-based optimization algorithm to minimize the risk $L_T(\mathbf{v})$ on the Ames housing data described in Part 1 yields the parameters $\mathbf{v} = (b, \mathbf{w})$ of the classifier whose boundary is shown in black in panel (b) of that Figure.

All lines parallel to the boundary (black) are lines of constant score in terms of home quality and condition. As we saw in Part 1, size (horizontal axis) and price (vertical axis) correlate fairly well to each other for this data, because homes are restricted to a single neighborhood. This is why the regression line (in gold) fits the data points $\mathbf{x} = (x_1, x_2)$ fairly well. The slope of the classifier boundary line (in black), on the other hand, shows that both size and price (or either of them, since they correlate well) are to some extent indicators of home condition and quality, in that increasing either factor increases the quality score (distance form the decision boundary). In other words, expensive (and large) homes are kept generally in better conditions than less expensive ones. This makes some sense: If you can afford to buy an expensive home you may also be able to pay for its upkeep more comfortably.

The data points in Figure 1 of Part 1 are obviously not linearly separable, and the risk incurred by the optimal logistic-regression classifier is therefore greater than zero.

3 Multi-Class Logistic-Regression Classifiers

When generalizing logistic-regression classification to more than two classes, it is useful to rewrite the score function defined in equation 3 in a more symmetric way by multiplying both numerator and denominator by $e^{a/2}$:

$$s(\mathbf{x}) \stackrel{\text{def}}{=} \frac{1}{1+e^{-a}} = \frac{e^{\frac{a}{2}}}{e^{\frac{a}{2}} + e^{-\frac{a}{2}}}$$

where, as before,

$$a = a(\mathbf{x}) = b + \mathbf{w}^T\mathbf{x}\ .$$

When the activation a is positive, it is proportional to how far away from the decision boundary the data point $\mathbf{x}$ is in the half-space for class 1. When a is negative, its magnitude is proportional

to how far away from the decision boundary $\mathbf{x}$ is in the half-space for class 0. The reverse of course holds for a new activation

$$a' = -a \ .$$

If we use both a and a', each class has its own activation, and we can write

$$\begin{aligned} s_0(\mathbf{x}) &\stackrel{\mathsf{def}}{=} \frac{e^{\frac{a'}{2}}}{e^{\frac{a}{2}} + e^{\frac{a'}{2}}} \\ s_1(\mathbf{x}) &\stackrel{\mathsf{def}}{=} \frac{e^{\frac{a}{2}}}{e^{\frac{a}{2}} + e^{\frac{a'}{2}}} \ . \end{aligned}$$

This double definition is redundant, since

$$s_0(\mathbf{x}) + s_1(\mathbf{x}) = 1$$

by construction. However, we can now see more easily than before how to generalize the logistic-regression classifier to multiple classes: Each class has its own a_k and logistic function s_k. The denominator for all s_k is the same, the sum of all numerators, so that the logistic functions for all K classes add up to 1. Before we write this general formula, we make two changes:

- We number classes $1, \ldots, K$ rather than $0, \ldots, K-1$.
- We replace each $a/2$ with a.

Both changes are made for simplicity, and have no substantive consequence, as we now explain.

The first change merely renames the classes. When the two classes (for $K = 2$) were named $y = 0$ and $y = 1$, we could write, quite simply

$$q = y$$

where y is the class label and q is the true probability (with value either 0 or 1) of data point $\mathbf{x}$ being in class 1. If we now rename class 0 to class 1 and class 1 to class 2, then q is reinterpreted as the probability that the true label of a sample is 2. As a consequence, the formula for q is a bit more complicated:

$$q = y - 1 \ .$$

This is because if the true label y of a sample $(\mathbf{x}, y)$ is 2, then the probability q that $\mathbf{x}$ has true label 2 is 1 (that is, $y - 1$). If the true label y is 1, then the probability q that $\mathbf{x}$ has true label 2 is 0 (that is, again, $y - 1$).

This renaming complicates some of the formulas for the two-class case, but in minor ways, and simplifies those for the multi-class case.

Concerning the second change above, while the function $e^a/(e^a + e^{-a})$ is different from the function $e^{a/2}/(e^{a/2}+e^{-a/2})$, we recall that a is an unnormalized activation, so a single multiplicative factor common to all activations is irrelevant.

With these changes, in the K-class case for $K \geq 2$, we define the K score functions

$$s_k(\mathbf{x}) = f(a_k(\mathbf{x})) \stackrel{\mathsf{def}}{=} \frac{e^{a_k(\mathbf{x})}}{\sum_{j=1}^{K} e^{a_j(\mathbf{x})}} \quad \text{where} \quad a_k(\mathbf{x}) \stackrel{\mathsf{def}}{=} b_k + \mathbf{w}_k^T \mathbf{x} \quad \text{for} \quad k = 1, \ldots, K \ .$$

The vector of functions

$$\mathbf{s}(\mathbf{x}) = \begin{bmatrix} s_1(\mathbf{a}) \\ \vdots \\ s_K(\mathbf{a}) \end{bmatrix}$$

is called the *softmax function* of the activations in the vector

$$\mathbf{a}(\mathbf{x}) = \begin{bmatrix} a_1(\mathbf{x}) \\ \vdots \\ a_K(\mathbf{x}) \end{bmatrix}$$

that collects the K activations for data point $\mathbf{x}$.

Here is why this vector of functions is called a "softmax." If one of these activations, say a_i, is much bigger than all of the others, then the sum of the exponentials of the activations, $\mathbf{1}^T \exp(\mathbf{a})$, is approximately equal to $\exp(a_i)$, so that

$$s_i(\mathbf{a}) \approx 1 \quad \text{and} \quad s_j(\mathbf{a}) \approx 0 \quad \text{for} \quad j \neq i \ ,$$

and the function effectively acts as an indicator of the largest entry in $\mathbf{a}$: a "max," but a soft one, because of the approximation. Somewhat more formally, and quite obviously,

$$\lim_{\alpha \to \infty} \mathbf{a}^T \mathbf{s}(\alpha \mathbf{a}) = \max(\mathbf{a}) \ .$$

In summary, given a data point $\mathbf{x} \in \mathbb{R}^d$, the logistic-regression classifier computes the vector $\mathbf{s}(\mathbf{x})$ of K scores, one score per class. These are numbers between 0 and 1 and add up to one, and they therefore formally represent what is called a *categorical* distribution in probability theory:

$$\mathbf{p} = \begin{bmatrix} p_1 \\ \vdots \\ p_K \end{bmatrix} = \mathbf{s}(\mathbf{x}) \quad \text{with} \quad p_k \in [0,1] \quad \text{and} \quad p_1 + \ldots + p_K = 1 \ .$$

Thus, p_k is in some sense the probability that the data point $\mathbf{x}$ belongs to class k. Of course, the same warnings hold here as did for $K = 2$, and the analogy with probabilities is merely formal.

Given now a data point $\mathbf{x}$ with true label y (one of K categories in $Y = \{1, \ldots, K\}$), we can encode the statement $y = k$ with a vector

$$\mathbf{q} = \begin{bmatrix} q_1 \\ \vdots \\ q_K \end{bmatrix} \quad \text{where} \quad q_y = 1 \quad \text{and} \quad q_k = 0 \quad \text{for} \quad k \neq y \ .$$

As an example, if $K = 4$ and $y = 3$, we write

$$\mathbf{q} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \ .$$

Similarly to $\mathbf{p}$, the vector $\mathbf{q}$ is formally a categorical probability distribution, although of an "extreme" form, because one entry is one and all others are zero. This encoding of y is often called a *one-hot* encoding of the true label y.

The cross-entropy loss for the K-class case is the immediate extension of the definition we saw for $K = 2$:

$$\ell(y, \mathbf{p}) = -\log p_y \ ,$$

and we can use the fact that $\mathbf{q}$ is an indicator function to rewrite this loss as follows:

$$\ell(y, \mathbf{p}) = -\sum_{k=1}^{K} q_k \log p_k$$

where $\mathbf{q}$ is the one-hot encoding of y. The two expressions are equivalent, but the latter is more convenient to use when computing derivatives.

The rest of the story is the same as in the case $K = 2$: The risk is still defined as the average loss over the training set (see equation 4):

$$L_T(b, \mathbf{w}) = \frac{1}{N} \sum_{n=1}^{N} \ell(y_n \ ; \ b, \mathbf{w}) \ .$$

Computing the Hessian shows that the resulting function is convex in the parameters b and $\mathbf{w}$, and the gradient can be used to minimize the loss over the training set as a function of the parameters. The calculations are similar but more complex, and we omit them in this course. Several packages exist that perform this optimization.

References

[1] T. M. Cover and J. A. Thomas. *Elements of Information Theory, 2nd Edition.* John Wiley and Sons, Inc., Hoboken, NJ, 2006.

[2] D. .J. C. MacKay. *Information Theory, Inference, and Learning Algorithms.* Cambridge University Press, Cambridge, UK, 2003.

[3] R. J. McEliece. *The Theory of Information and Coding.* Addison-Wesley, Reading, MA, 1977.

Appendices

A The Cross Entropy Loss and Information Theory

This Appendix discusses a connection between the cross entropy loss and concepts of probability and information theory. This link is a bit of a rationalization *post facto*. After all, we saw that it is a stretch to interpret the logistic-regression score $p = s(\mathbf{x})$ as a probability. Furthermore, the cross entropy risk (the average of the cross entropy loss over the training set) is not really the risk we would like to minimize: *That* is the zero-one risk. However, at the very least, this connection explains the name of the loss function.

The discussion that follows is limited to binary and independent events, while the theory is much more general. In addition, our discussion is informal. Please refer to standard texts for more detail and proofs [1, 2, 3].

A.1 Coding for Efficiency

The concept of cross entropy has its roots in coding theory. Suppose that you repeatedly observe a binary experiment with possible outcomes 0 and 1. You want to transmit your observations of the outcomes to a friend over an expensive channel. The sequence of observations is assumed to be unbounded in length, so an initial cost of exchanging some communication protocol with your friend is acceptable, and is not part of the cost. However, you would like to define that protocol so that, once it is established, you then save on the number of bits sent to communicate the outcomes. Coding theory studies how to do this. The same considerations apply if instead of sending the information you want to store it on an expensive medium. Thus, coding theory is really at the heart of the digital revolution, as it is used extensively in both communications and computing (and more).

Assume that the *true* probability q that the outcome is 1 is known to both you and your friend. A naive transmission scheme would send a single bit, 1 or 0, for each outcome, so the cost of transmission would be one bit per outcome. It turns out that if $q = 1/2$ that's the best you can do.

However, if q has a different value, you can do better by *coding* the information before sending it, and your friend would then *decode* it at the other end of the channel.[4] This is obvious for extreme cases: If $q = 0$ or $q = 1$, you don't need to communicate anything.

Suppose now that q has some arbitrary value in the interval $[0, 1]$. There are very many ways of coding a stream of bits efficiently, and perhaps the simplest to think about is *Huffman coding*. Rather than sending one bit at a time, you send a sequence of *blocks* of m bits each.[5] There are $n = 2^m$ possible blocks $b_1, \ldots, b_n$ of m bits, and part of establishing the communication protocol is to prepare and share with your friend a table of n *codes* $c_1, \ldots, c_n$, one for each block. Each code is also a sequence of bits, but different codes can have different lengths, in contrast with blocks, which are all m bits long. Then, instead of sending block b_i, you look up its code c_i and send that. Your friend receives c_i and uses a reverse version of the same table to look up b_i. The whole magic is to build the tables so that *likely blocks get short codes*. As a result, you end up sending short blocks often and long blocks rarely, and on average you save.

[4] In the storage application, you encode before you store, and you decode after you retrieve.

[5] This m has nothing to do with the number of parameters in a predictor!

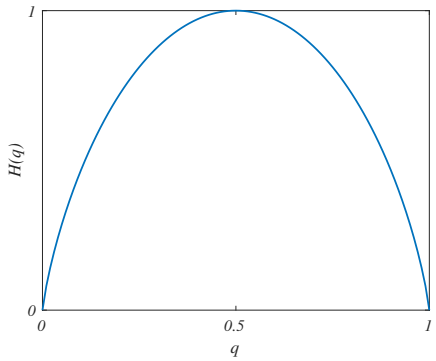

Figure 3: The entropy function.

Huffman codes are built with a very simple procedure, which is not described here.[6] Obviously, in order to do so, you need to know q, so you can figure out the probability of occurrence of each block.[7] Large blocks give you more flexibility, and the code becomes increasingly efficient (but harder to set up) as $m \to \infty$.

A.2 Entropy

Claude Shannon, the inventor of information theory, proved that the best you can do, even using coding schemes different from Huffman's, is to use on average

$$H(q) \stackrel{\mathsf{def}}{=} \mathbb{E}[-\log_2 q] = -q \log_2 q - (1-q) \log_2 (1-q)$$

code bits for each block bit you want to send, and called the quantity $-\log_2 q$ the *information* conveyed by a bit that has probability q of being 1 (and therefore probability $1-q$ of being 0). He called the statistical expectation $H(q)$ of the information the *entropy* of a source that emits a sequence of bits with that distribution.

The expression for entropy requires a small *caveat*: When $q = 0$ or $q = 1$, one of the two terms is undefined, as it is the product of zero and infinity. However, it is easy to use De l'Hospital rule to check that

$$\lim_{q \to 0} q \log_2 q = 0$$

so whenever we see that product with $q = 0$ we can safely replace it with 0. With this *caveat*, the entropy function looks as shown in Figure 3 when plotted as a function of q.

Note that $H(0) = H(1) = 0$, consistently with the fact that when both you and your friend know all the outcomes ahead of time (because q is either 0 or 1) there is nothing to send. Also

$$H(q) \leq 1 \,,$$

meaning that you cannot do worse with a good coding scheme than without it, and $H(1/2) = 1$: When zeros and ones have the same probability of occurring, coding does not help. In addition,

[6] Huffman coding is fun and simple, and it would take you just a few minutes to understand, say, the Wikipedia entry on it.

[7] As stated earlier, we assume independent events, although the theory is more general than that.

entropy is symmetric in q,

$$H(q) = H(1-q) \ ,$$

a reflection of the fact that zeros and ones are arbitrary values (so you can switch them with impunity, as long as everyone knows).

A.3 Cross Entropy

Cross entropy comes in when the table you share with your friend is based on the wrong probability. If the true probability is q but you think it's p, you pay a penalty, and Shannon proved that then the best you can do is to send on average

$$H(q,p) = -q \log_2 p - (1-q) \log_2 (1-p)$$

code bits for every block bit. This quantity is the *cross entropy* between the two distributions: One is the true Bernoulli distribution $(1-q, q)$, the other is the estimated Bernoulli distribution $(1-p, p)$.

Cross entropy is obviously not symmetric,

$$H(q,p) \neq H(p,q) \quad \text{in general,}$$

as it is important to distinguish between true and estimated distribution. It should come as no surprise (but takes a little work with standard inequalities to show) that

$$H(q,p) \geq H(q) \quad \text{for all} \quad q, p \in [0,1] \ .$$

This reflects the fact that you need more bits if you use the wrong coding scheme. Therefore,

> We can view $H(q,p) - H(q)$ as the added average transmission cost of using probability distribution $(1-p, p)$ for coding, when the correct distribution is $(1-q, q)$.

This result holds more generally for arbitrary discrete distributions, not necessarily binary. If there are k possible outcomes rather than 2, the true probability distribution is $\mathbf{q} = (q_1, \ldots, q_k)$, and the estimated distribution is $\mathbf{p} = (p_1, \ldots, p_k)$, then the formulas for entropy and cross entropy generalize in the obvious way:

$$H(\mathbf{q}) = -\sum_{i=1}^{k} q_i \log_2 q_i \quad \text{and} \quad H(\mathbf{q}, \mathbf{p}) = -\sum_{i=1}^{k} q_i \log_2 p_i \ .$$

A.4 The Cross Entropy Loss

In logistic-regression classification, given a data point $\mathbf{x}$, the corresponding true label y is either 0 or 1. This fact is certain, and we can express it probabilistically with an "extreme" distribution, depending on the value of y: If $y = 1$, then we can say that its "true Bernoulli distribution" is $(0, 1)$, that is, $(1-q, q)$ with $q = 1$: The probability of y being 0 is 0, and the probability of y being 1 is 1. If $y = 0$, the situation is reversed: The probability of y being 0 is 1, and the probability of

y being 1 is 0, so the true Bernoulli distribution is $(1, 0)$. In either case, we have $q = y$, a fortunate consequence of our choice of names (0 and 1) for the two classes of the classification problem.

On the other hand, the value p returned by the score function can be interpreted as the Bernoulli distribution $(1 - p, p)$: The score function's estimate of the probability that $y = 0$ is $1 - p$, and the score function's estimate of the probability that $y = 1$ is p.

One way to quantify the cost of estimating that the distribution is $(1 - p, p)$ while the true distribution is $(1 - q, q)$ is to use the difference $H(q, p) - H(q)$. In light of our previous discussion, this effectively means that we reframe classification as a coding problem: Every time I make a mistake, I need to send an amendment, and that costs bits.

Of course, in classification we don't necessarily care about bits as a unit of measure, so we can use any base for the logarithm, not necessarily 2. In addition, since q is "extreme" (either 0 or 1) we can remove $H(q)$, since, as we saw earlier, $H(0) = H(1) = 0$.

This argument therefore suggests using the cross entropy as a measure of loss. Since q and the true label y happen to have the same numerical value, we can replace q with y, which is a binary variable rather than a probability:

$$H(y, p) = -y \log p - (1 - y) \log(1 - p) = \begin{cases} \log p & \text{if } y = 1 \\ \log(1 - p) & \text{if } y = 0. \end{cases}$$

B Derivatives of the Risk of the Logistic-Regression Classifier

This Appendix computes the gradient and Hessian of the risk L_T defined in equation 4 with respect to the vector

$$\mathbf{v} \;=\; \begin{bmatrix} b \\ \mathbf{w} \end{bmatrix}$$

of classifier parameters. It also shows that the Hessian is positive semi-definite, so that $L_T(\mathbf{v})$ is a weakly convex function of $\mathbf{v}$.

The Gradient of the Risk From the chain rule for differentiation and equation 3, the gradient of the loss with respect to $\mathbf{v}$ is

$$\nabla \ell \;=\; \frac{d\ell}{ds}\, \nabla s \;=\; \frac{d\ell}{ds}\, \frac{ds}{d\delta}\, \nabla \delta \,.$$

We have

$$\begin{aligned}
\frac{d\ell}{ds} &= \frac{d}{ds}\, [-y \log s - (1 - y) \log(1 - s)] \;=\; -\frac{y}{s} + \frac{1 - y}{1 - s} \;=\; \frac{s - y}{s\,(1 - s)} \\
\frac{ds}{d\delta} &= \frac{d}{d\delta}\, \frac{1}{1 + e^{-\delta}} \;=\; \frac{e^{-\delta}}{(1 + e^{-\delta})^2} \;=\; \frac{1}{1 + e^{-\delta}}\, \frac{e^{-\delta}}{1 + e^{-\delta}} \\
&= \frac{1}{1 + e^{-\delta}}\, \frac{1 + e^{-\delta} - 1}{1 + e^{-\delta}} \;=\; \frac{1}{1 + e^{-\delta}} \left(1 - \frac{1}{1 + e^{-\delta}}\right) \\
&= s\,(1 - s) \\
\nabla \delta &= \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix} .
\end{aligned}$$

The crucial point here is that the derivative $\frac{ds}{d\delta}$ of the logistic function with respect to δ, that is, $s\,(1-s)$, appears at the denominator of the derivative $\frac{d\ell}{ds}$ of the loss with respect to the output of the logistic function, and a cancellation occurs when computing the product $\frac{d\ell}{ds}\,\frac{ds}{d\delta}$. With this cancellation, we obtain:

$$\nabla \ell(y, s(\mathbf{x}\ ;\ \mathbf{v})) \ = \ [s(\mathbf{x}\ ;\ \mathbf{v}) - y] \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix} .$$

The risk is simply the average loss over the training set, so that the gradient of the risk is

$$\nabla L_T(\mathbf{v}) \ = \ \frac{1}{N} \sum_{n=1}^{N} [s(\mathbf{x}_n\ ;\ \mathbf{v}) - y_n] \begin{bmatrix} 1 \\ \mathbf{x}_n \end{bmatrix} .$$

The Hessian of the Risk We obtain the Hessian of ℓ by differentiating the gradient with respect to $\mathbf{v}$ one more time, and arranging the derivatives into a $(d+1) \times (d+1)$ matrix:

$$H_\ell \ = \ \nabla\nabla\ell \ = \ \nabla \left(s \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix} \right)$$

since y does not depend on $\mathbf{v}$, so that

$$H_\ell \ = \ \nabla s \begin{bmatrix} 1 & \mathbf{x} \end{bmatrix} \ = \ \frac{ds}{d\delta}\, \nabla\delta \begin{bmatrix} 1 & \mathbf{x} \end{bmatrix} \ = \ s\,(1-s) \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix} \begin{bmatrix} 1 & \mathbf{x} \end{bmatrix}$$

and therefore, spelling out the independent variables,

$$H_\ell(\mathbf{v}) \ = \ s(\mathbf{x}\ ;\ \mathbf{v})\ [1 - s(\mathbf{x}\ ;\ \mathbf{v})] \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix} \begin{bmatrix} 1 & \mathbf{x} \end{bmatrix} .$$

The Hessian of the risk is again obtained by averaging over the training set:

$$H_{L_T}(\mathbf{v}) \ = \ \frac{1}{N} \sum_{n=1}^{N} s(\mathbf{x}_n\ ;\ \mathbf{v})\ [1 - s(\mathbf{x}_n\ ;\ \mathbf{v})] \begin{bmatrix} 1 \\ \mathbf{x}_n \end{bmatrix} \begin{bmatrix} 1 & \mathbf{x}_n \end{bmatrix} .$$

Convexity of the Risk The Hessian matrix H_{L_T} is positive semi-definite, because for any $\mathbf{v} \in \mathbb{R}^{d+1}$ we have

$$\mathbf{v}^T H_{L_T} \mathbf{v} \ = \ \mathbf{v}^T \left\{ \frac{1}{N} \sum_{n=1}^{N} s\,(1-s) \begin{bmatrix} 1 \\ \mathbf{x}_n \end{bmatrix} \begin{bmatrix} 1 & \mathbf{x}_n \end{bmatrix} \right\} \mathbf{v} \ = \ \frac{1}{N} \sum_{n=1}^{N} s\,(1-s) \left(\mathbf{v}^T \begin{bmatrix} 1 \\ \mathbf{x}_n \end{bmatrix} \right)^2 \ \geq \ 0$$

since all the terms in the last summation are nonnegative. Therefore, the risk L_T for the logistic-regression classifier is a (weakly) convex function of the classifier parameters b ad $\mathbf{w}$ collected in $\mathbf{v}$.