CISC 271 Class 29

Classification – Logistic Regression

Text Correspondence: Hastie *et al.*, 2009 [6], pp 119–122

Main Concepts:

- *Linear activation – limited ability to separate*
- *Semilinear – early use of logistic function*
- *Models of errors – least squares, probabilities*
- *Bridge – other areas of machine learning*

Sample Problem, Machine Inference: How can we learn the "best" binary separation of data?

In a landmark 1969 book – re-issued in 1988 and 2017 – Marvin Minsky and Seymour Papert [10] proved that entire classes of seemingly simple problems cannot be solved by Perceptron-like models of an artificial neuron. The models included the original Perceptron of Frank Rosenblatt [11], plus the Widrow-Hoff original model [16] and variants [17]. The criticism applied also to a single "layer" of simple artificial neurons.

An example of such unlearnable data are shown in Figure 29.1.

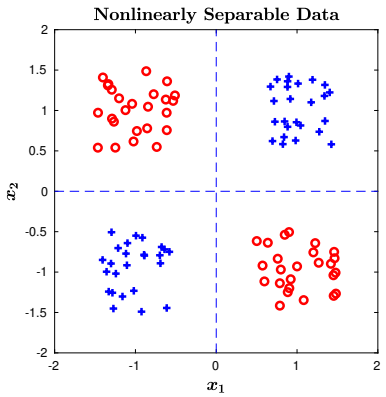


Figure 29.1: Sample data that cannot be correctly "learned" with a single simple artificial neuron. These are sometimes called "exclusive-or", or XOR, data because their pattern is visually like the truth table of the logical function.

Work that is now considered foundations for neural networks, was first published as a technical report by Rumelhart *et al.* in 1985 [12] and communicated by Hinton in a 1986 article that is widely

cited [13]. This work re-discovered work by Paul Werbos in 1974, re-published in 1994 [15], that described how to automatically differentiate complicated compositions of functions.

The concept is simple. Instead of using the linear weighted sum of an observation that is in an augmented "vector" $\hat{x}_i$, which we compute as $u_i = \hat{x}_i^T \hat{w}$, they considered a nonlinear activation function $\phi(\cdot)$. This activation function must have a few key attributes by being:

- Easy to compute
- Continuous
- Differentiable
- Optionally, bounded derivatives

In the original technical report, on Page 9, Rumelhart *et al.* we emphasize that they wrote:

"... we have used the *logistic* activation function ..."

As we have explored earlier, the logistic function satisfies these key attributes.

29.1 Semilinear Activation – Logistic Function As A Sigmoid Curve

Previously, we created a score z_i for the i^{th} observation that was represented as a "vector" $\vec{x}_i$. We represented a separating hyperplane $\mathbb{H}$ by using a unit vector $\vec{n}$ and a bias scalar a so that the score, which was the distance to the hyperplane, could be written as

$$z_i = \vec{n}^T \vec{x}_i + a \tag{29.1}$$

We also explored how the logistic function is one way to map a linear summation to a probability. We represented this activation function of the score as

$$p(\vec{x}_i) = \frac{1}{1 + e^{-\vec{n}^T \vec{x}_i + a}} \tag{29.2}$$

In artificial neural networks, Equation 29.2 is modified. The computations of the *unit* weight vector $\vec{n}$ and its associated bias scalar a are typically omitted. Instead, the weight vector $\vec{w}$ and the bias scalar b are used directly. The logistic activation function for neural networks can be written, in terms of the augmented "vector" $\hat{x}_i$ for the i^{th} observation and the augmented weight vector $\hat{w}$, as

$$\phi_i = \phi(\hat{w}; \hat{x}_i) = \frac{1}{1 + e^{-\hat{x}_i^T \hat{w}}} \tag{29.3}$$

The scaling effects that distinguish Equation 29.3 from Equation 29.2 are empirically negligible in extensive iterative computations. When plotted, the curve of Equation 29.3 is "S"-shaped and the computation is sometimes called a *sigmoid* activation function.

29.2 Models Of Residual Error

Neural networks and logistic regression both use Equation 29.3 to perform binary classification of observations. An important technical detail is the accumulation of residual errors into a single scalar value that can be optimized.

For the i^{th} observation, the data are represented in an augmented "vector" $\hat{x}_i$ and the label is $y_i \in \{0, 1\}$. There are two common formulations of the objective function that accumulates the residual errors for the observations. These are often called the *squared error* and the *negative log* error. The first objective is one that we previously explored: this is the squared Euclidean norm of the residual error vector $\vec{e}$, which we can write as

$$E_2(\hat{X}) \quad = \quad \sum_{i=1}^{m} (y_i - \phi_i)^2 \tag{29.4}$$

The second objective uses the logarithm of the likelihood that the activation value ϕ_i matches the label y_i. To ensure that positive numbers arise, the negative logarithm of the likelihood is defined as

$$l_i \stackrel{\text{def}}{=} \begin{cases} -\ln(1 - \phi_i) & \text{if} \quad y = 0 \\ -\ln(\phi_i) & \text{if} \quad y = 1 \end{cases} \tag{29.5}$$

Using Equation 29.5, the negative-log objective function is written as

$$E_L(\hat{X}) \quad = \quad \sum_{i=1}^{m} l_i \tag{29.6}$$

The differences between the squared-error objective of Equation 29.4 and the negative-log objective of Equation 29.6 can be visualized by plotting the error for a scalar argument z. Figure 29.2 show the logistic function and – using the squared error objective – the error for the label $y = 0$, the error for the label $y = 1$, and the total error as the argument z is varied over a small domain. We can see that the individual errors are bounded below at 0 and are bounded above at 1. The total error is bounded below at $z = 0$, bounded above at 1, and asymptotically approach 1 as z increases or decreases without limit.

Figure 29.3 show the logistic function and – using the negative logarithmic likelihood objective – the error for the label $y = 0$, the error for the label $y = 1$, and the total error as the argument z is varied over a small domain. We can see that the individual errors are bounded below at 0 and increase without bounds. The total error is bounded below at $z = 0$ and increases without bounds as z increases or decreases without limit.

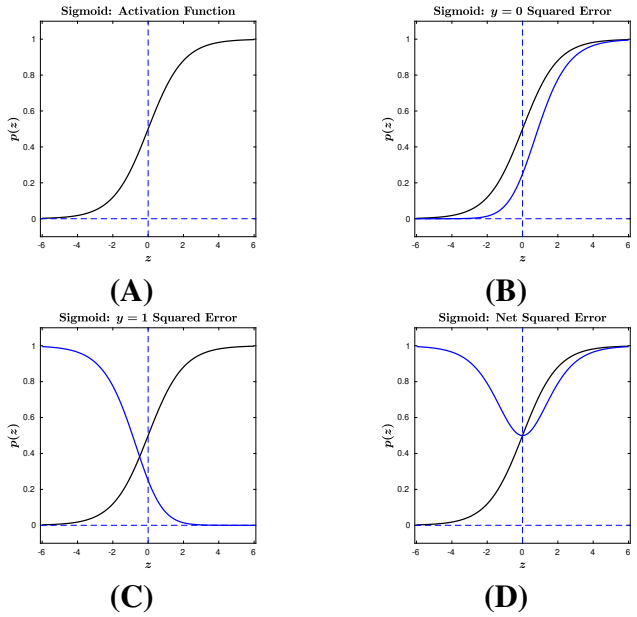


Figure 29.2: Logistic function and squared residual error. (A) Logistic function of a scalar argument z. (B) Error for label $y = 0$ is shown in blue. (C) Error for label $y = 1$ is shown in blue. (D) Total error, which is the sum of the error for the label values, is shown in blue.

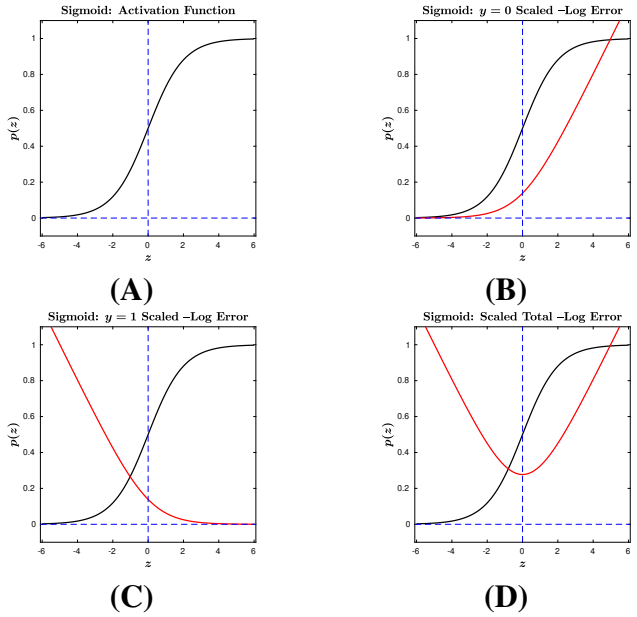


Figure 29.3: Logistic function and logarithmic residual error, the latter scaled to fit the plot. (A) Logistic function of a scalar argument z. (B) Error for label $y = 0$ is shown in red. (C) Error for label $y = 1$ is shown in red. (D) Total error, which is the sum of the error for the label values, is shown in red.

29.3 Implementations Of Logistic Activation

Using the logistic activation function, the squared-error objective can be implemented for a single artificial neuron by an optimization method known as steepest descent. The derivation requires nonlinear data analysis that is beyond the scope of this course but the equation is not difficult to implement in `MATLAB`. The iteration looks like a modification of the Perceptron Rule and uses a "learning rate" η, written in our notation as

$$\hat{w}_k = \hat{w}_{k-1} + \eta(y_i - \phi_i)\phi_i(1 - \phi_i)\hat{x}_i^T \tag{29.7}$$

The difference between Equation 29.7 and the Perceptron Rule are the term $\eta\phi_i(1-\phi_i)$, which combines the learning rate η and the derivative of the logistic function $\phi_i(1-\phi_i)$.

The negative-log objective function can be computed using external software. For example, `MATLAB` provides logistic regression as part of its *Statistics and Machine Learning* Toolbox. The `glmfit` computes a generalized linear model. Specifically, we can use a binomial model because each label y_i can have exactly one of two values. For technical reasons, the probability unit or "probit" function is preferred as the link function. This binomial regression scales the weight vector $\hat{w}$ to maximally separate the data, which can produce numerically large weights.

29.3.1 Example – Fisher's Iris Data

We can test these implementations by using Fisher's Iris data. Consider the petal data, which we have used previously and which is readily available from many sources. We will use binary labels so that the "beach-head" plant is assigned to label 1 and the other species are assigned to label 0. The data can be plotted as shown in Figure 29.4.

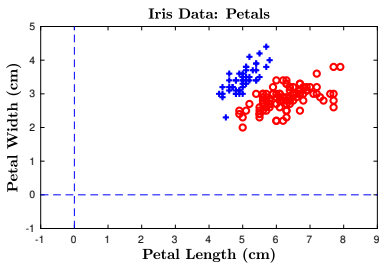
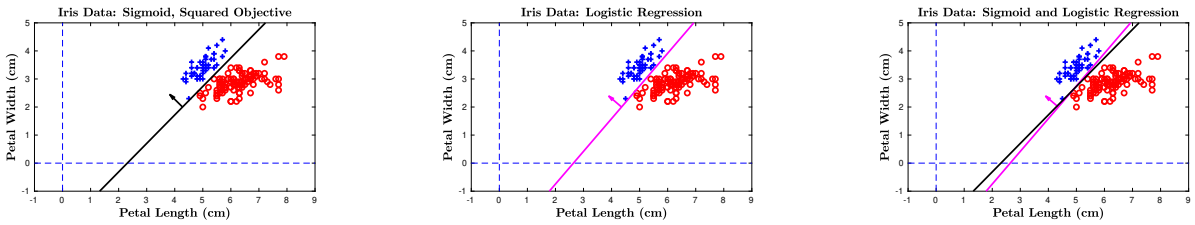


Figure 29.4: Fisher's Iris data for the petals measurements. The horizontal axis is each petal's length and the vertical axis is each petal's width.

We can compute a separating hyperplane using Equation 29.7 and the squared-error objective of Equation 29.4. We can also compute a logistic regression in `MATLAB`, using the negative-

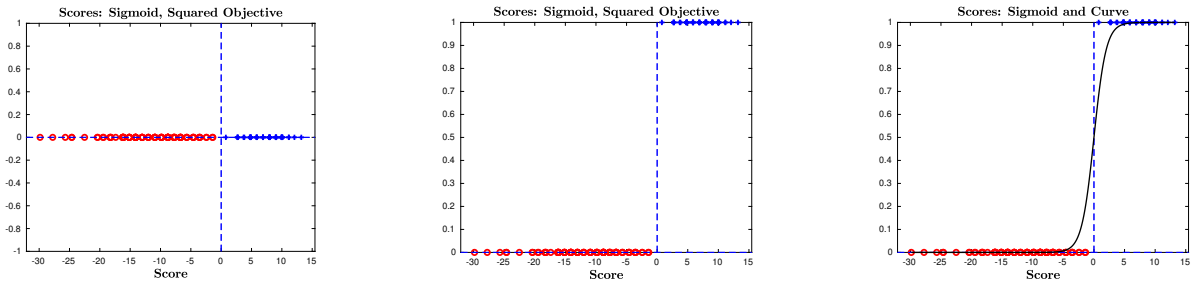
logarithmic likelihood objective function of Equation 29.6. As shown in Figure 29.5, both implementations can perfectly separate the petal data.



(A) **(B)** **(C)**

Figure 29.5: Fisher's Iris data and separating hyperplanes. (A) Separation using a single artificial neuron with logistic activation and a squared-error objective; the hyperplane is shown in black. (B) Separation using logistic regression and the negative-logarithmic likelihood objective function; the hyperplane is shown in magenta. (C) The hyperplanes superimposed on the data.

We can gain a deeper understanding of these implementations by plotting the scores, which are $z_i = \hat{x}_i^T \hat{w}$. For the squared-error objective and the artificial-neuron computation, Figure 29.6 shows the labels as colored shapes according to their scores. Figure 29.6(A) is a conventional plot. Figure 29.6(B) plots the labels as shapes and also uses the label value as a vertical axis, which improves the visualization. We can observe that the scores have values, roughly, in the interval $[-30 \;\; +15]$. We can superimpose the logistic function, shown in Figure 29.6(C), which shown that the upper-bounded behavior of the squared-error objective has preserved the smooth behavior of the logistic function.

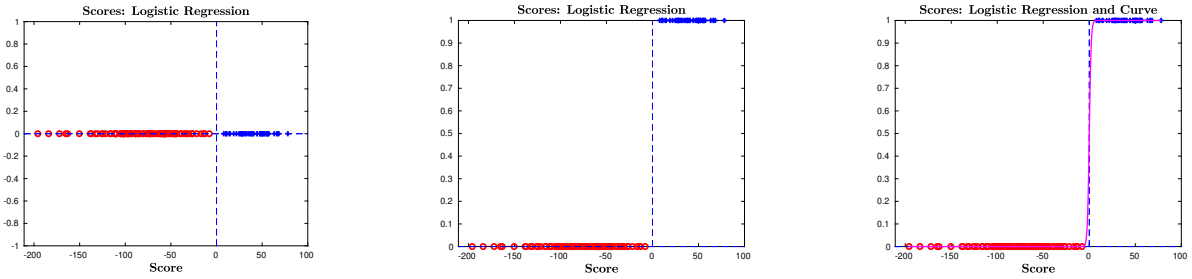


(A) **(B)** **(C)**

Figure 29.6: Scores and labels for some of Fisher's Iris data, separated using an artificial neuron with a squared-error objective. (A) Scores are the horizontal axis; label 1 observations are shown as blue crosses and label 0 observations are shown as red circles. (B) The label value is used as the vertical axis. (C) The logistic function is superimposed on the labels as a black curve.

We can repeat this process for the computation of logistic regression that used the negative-logarithmic likelihood as the objective function. Figure 29.7 shows the labels and logistic function

in the same way that is shown in Figure 29.6. Logistic regression produced scores that have values, roughly, in the interval $[-200 \ +80]$.



(A) **(B)** **(C)**

Figure 29.7: Scores and labels for some of Fisher's Iris data, separated using logistic regression with a negative-logarithmic likelihood objective. (A) Scores are the horizontal axis; label 1 observations are shown as blue crosses and label 0 observations are shown as red circles. (B) The label value is used as the vertical axis. (C) The logistic function is superimposed on the labels as a magenta curve.

The weights for logistic regression are an order of magnitude grater that the weights for our artificial neuron, which is in part because of the unbounded nature of the negative-logarithmic likelihood objective function. The weights are "forcing" the scores as far apart as is computationally acceptable in the implementation that was used to generate these results.