CISC 271 Class 34

Classification – Logistic Regression and Loss Functions

Text Correspondence: Hastie *et al.*, 2009 [6], pp 119–122 and pp 130–132

Main Concepts:

- *Logistic regression – binary model*
- *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?

Details of descent methods are topics in numerical optimization for machine learning.

34.1 Models Of Residual Error

Neural networks and logistic regression both 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 observation $\underline{x_i}$ 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 $\vec{r}$, which we can write as

$$E_2(X) \quad = \quad \sum_{i=1}^{m} (y_i - \phi_i)^2 \tag{34.1}$$

The second objective uses the logarithm of the likelihood that the activation value $\phi(u_i)$ matches the label y_i. To ensure that positive numbers arise, the negative logarithm of the likelihood is defined as

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

or

$$l_i \quad \stackrel{\text{def}}{=} \quad (1 - y)(-\ln(1 - \phi(u_i))) + y(-\ln(\phi(u_i))) \tag{34.2}$$

Using Equation 34.2, the negative-log objective function is written as

$$E_L(X) \quad = \quad \sum_{i=1}^{m} l_i \tag{34.3}$$

The differences between the squared-error objective of Equation 34.1 and the negative-log objective of Equation 34.3 can be visualized by plotting the error for a scalar argument z. Figure 34.1 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.

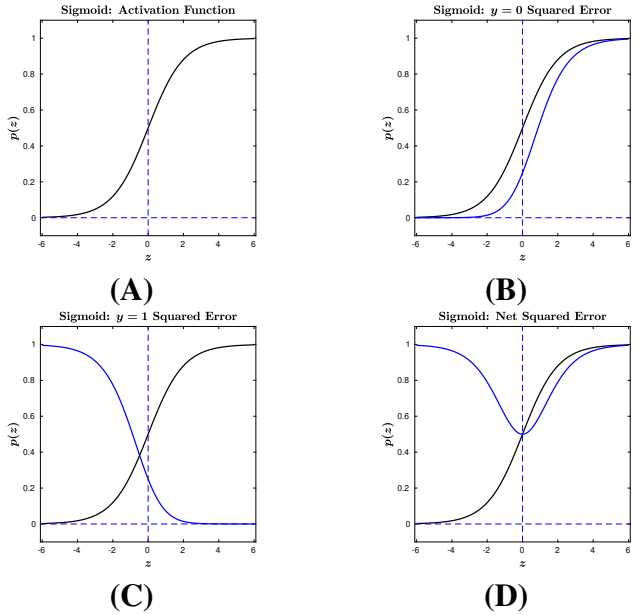


Figure 34.1: Logistic function and squared residual error. (A) Logistic function of a scalar argument z. (B) Error for label $y = 0$ is shown in blue. (C) Error for label $y = 1$ is shown in blue. (D) Total error, which is the sum of the error for the label values, is shown in blue.

Figure 34.2 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.

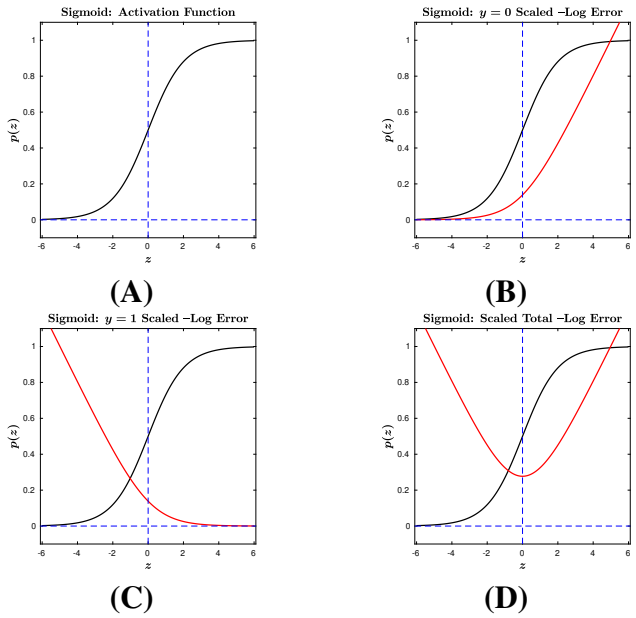


Figure 34.2: Logistic function and logarithmic residual error, the latter scaled to fit the plot. (A) Logistic function of a scalar argument z. (B) Error for label $y = 0$ is shown in red. (C) Error for label $y = 1$ is shown in red. (D) Total error, which is the sum of the error for the label values, is shown in red.

34.2 Implementations Of Logistic Activation

Using the logistic activation function, the squared-error objective can be implemented for a single artificial neuron by using Algorithm 29.1.

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 $\vec{w}$ to maximally separate the data, which can produce numerically large weights.

34.2.1 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 34.3.

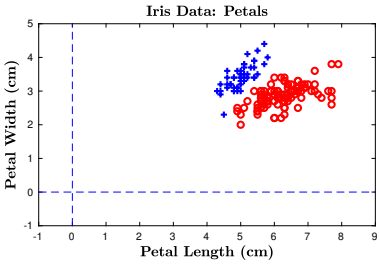


Figure 34.3: Fisher's Iris data for the petals measurements. The horizontal axis is each petal's length and the vertical axis is each petal's width.

We can compute a separating hyperplane using the squared-error objective of Equation 34.1. We can also compute a logistic regression in `MATLAB`, using the negative-logarithmic likelihood objective function of Equation 34.3. As shown in Figure 34.4, both implementations can perfectly separate the petal data.

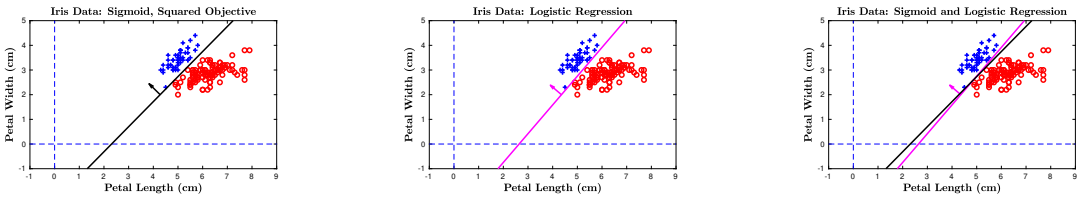
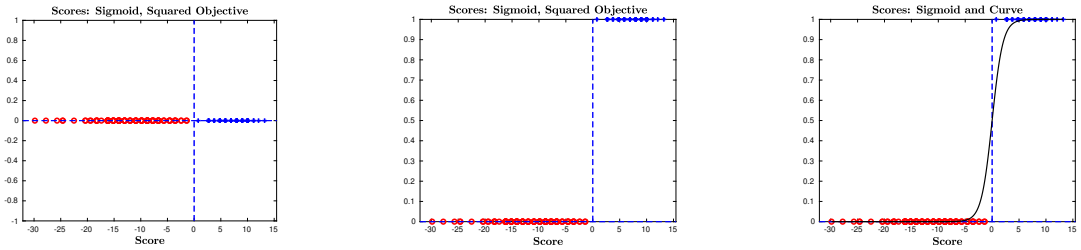


Figure 34.4: Fisher's Iris data and separating hyperplanes. (A) Separation using a single artificial neuron with logistic activation and a squared-error objective; the hyperplane is shown in black. (B) Separation using logistic regression and the negative-logarithmic likelihood objective function; the hyperplane is shown in magenta. (C) The hyperplanes superimposed on the data.

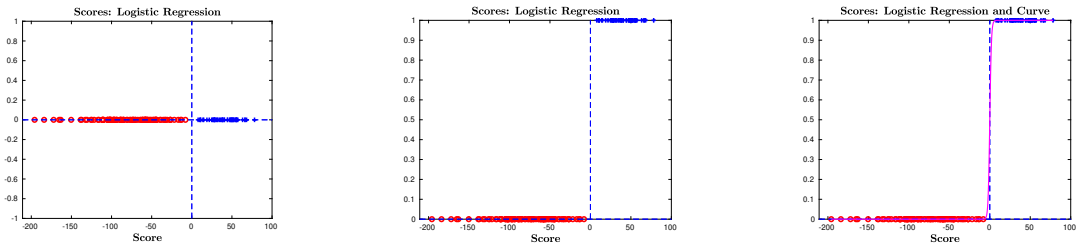
We can gain a deeper understanding of these implementations by plotting the scores, which are $z_i = \underline{x_i}\vec{w}$. For the squared-error objective and the artificial-neuron computation, Figure 34.5 shows the labels as colored shapes according to their scores. Figure 34.5(A) is a conventional plot. Figure 34.5(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 34.5(C), which shown that the upper-bounded behavior of the squared-error objective has preserved the smooth behavior of the logistic function.

We can repeat this process for the computation of logistic regression that used the negative-logarithmic likelihood as the objective function. Figure 34.6 shows the labels and logistic function in the same way that is shown in Figure 34.5. Logistic regression produced scores that have values, roughly, in the interval $[-200 \ \ +80]$.



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

Figure 34.5: 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.



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

Figure 34.6: 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.