Logistic Regression: A Statistical Learning Approach

Latreche Sara

June 10, 2025

Contents

1 Introduction to Logistic Regression

Definition

Logistic Regression is a statistical model used in supervised learning for binary classification. Unlike linear regression which predicts continuous values, logistic regression predicts probabilities using the logistic (sigmoid) function.

1.1 Motivation

Suppose we want to predict whether a person has a disease ($y = 1$) or not ($y = 0$) based on their features (age, blood pressure, etc). Logistic regression gives the probability that $y = 1$ given the input features.

Note

The output of logistic regression is a probability, so the prediction is made by thresholding (usually at 0.5).

2 The Logistic Model

3 Logistic Regression

While it might seem reasonable at first to use linear regression to tackle classification tasks—where the output y is categorical—it turns out this approach can lead to unsatisfactory results. One key issue is that linear regression can predict values outside the valid range of classification labels (e.g., less than 0 or greater than 1), which doesn't make sense when $y \in \{0, 1\}$.

To address this, we introduce a different model: instead of predicting the outcome directly with a linear combination of the features, we pass that combination through a nonlinear function that restricts the output to the range $[0, 1]$.

The function we use is the **sigmoid function**, also known as the **logistic function**, defined as:

$$g(z) = \frac{1}{1 + e^{-z}}$$

We then define our hypothesis function as:

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

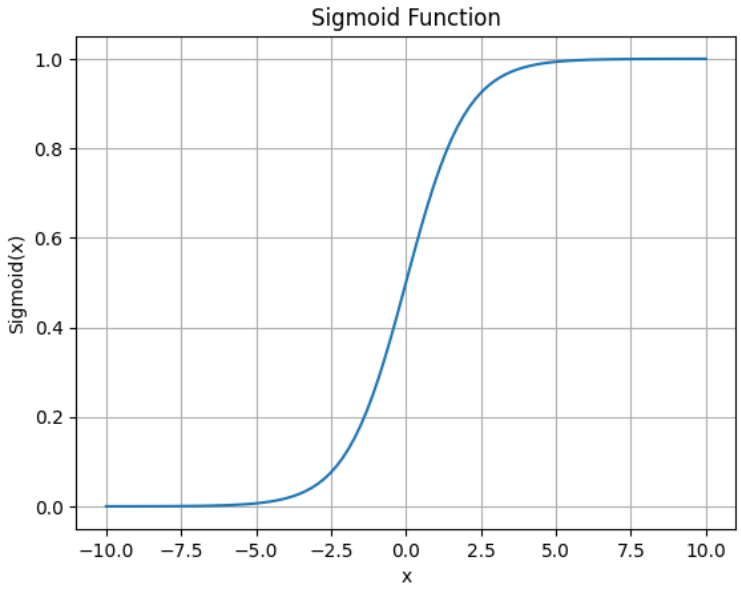
This formulation ensures that the output of $h_\theta(x)$ always lies strictly between 0 and 1, making it interpretable as the **probability** that the class label is 1, given the input features x. That is:

$$h_\theta(x) = P(y = 1 \mid x; \theta)$$

This is crucial in binary classification, where we want to decide between two classes. The logistic function behaves as follows:

- As $\theta^T x \to +\infty$, $h_\theta(x) \to 1$
- As $\theta^T x \to -\infty$, $h_\theta(x) \to 0$

A graph of $g(z)$ shows the familiar "S-shaped" curve.



This smooth curve transitions from 0 to 1, providing a natural mapping from real-valued inputs to probabilities. It's especially useful in cases where the decision boundary between the two classes isn't sharply defined.

To ensure compatibility with the intercept term, we continue to use the convention of augmenting the feature vector x with a bias term, setting $x_0 = 1$, so that the full linear expression becomes:

$$\theta^T x = \theta_0 + \theta_1 x_1 + \cdots + \theta_d x_d$$

An important analytical property of the sigmoid function is that its derivative has a simple and elegant form:

$$\frac{d}{dz}g(z) = g(z)(1 - g(z))$$

This derivative becomes particularly helpful when we derive the gradient of the cost function during model training.

While other activation functions could in theory be used to squeeze values between 0 and 1, the sigmoid function has several desirable mathematical properties that make it a standard choice, especially in the context of **generalized linear models (GLMs)** and probabilistic frameworks.

Maximum Likelihood Estimation of θ

With the logistic regression model in place, the next task is to determine how to optimally choose the parameters θ. Inspired by the probabilistic interpretation of logistic regression, we can estimate θ using the principle of **maximum likelihood**.

Recall that we interpret the output of the model as a probability:

$$P(y = 1 \mid x; \theta) = h_\theta(x), \quad P(y = 0 \mid x; \theta) = 1 - h_\theta(x)$$

These two cases can be unified into a single expression:

$$P(y \mid x; \theta) = (h_\theta(x))^y (1 - h_\theta(x))^{1-y}$$

Assuming that the training data consists of n i.i.d. examples $\{(x^{(i)}, y^{(i)})\}_{i=1}^n$, the joint likelihood of the entire dataset is the product of the individual probabilities:

$$L(\theta) = \prod_{i=1}^{n} P(y^{(i)} \mid x^{(i)}; \theta) = \prod_{i=1}^{n} (h_\theta(x^{(i)}))^{y^{(i)}} (1 - h_\theta(x^{(i)}))^{1-y^{(i)}}$$

Rather than maximizing the likelihood directly, it's more convenient to work with the **log-likelihood**, which transforms the product into a sum:

$$\ell(\theta) = \log L(\theta) = \sum_{i=1}^{n} \left[y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log\big(1 - h_\theta(x^{(i)})\big) \right]$$

This log-likelihood function serves as our **objective function** for optimization. To find the optimal θ, we can apply techniques such as gradient ascent (or use gradient descent on the negative log-likelihood, also referred to as the logistic loss or cross-entropy loss).

The logistic regression model, therefore, emerges as both a principled probabilistic classifier and a tractable optimization problem.

Maximizing the Likelihood

To find the optimal parameters θ, we maximize the log-likelihood $\ell(\theta)$ using **gradient ascent**, since $\ell(\theta)$ is a concave function. Unlike gradient descent (which minimizes a function), gradient ascent updates the parameters in the direction of the positive gradient:

$$\theta := \theta + \alpha \nabla_\theta \ell(\theta),$$

where α is the learning rate.

Let us derive the gradient of the log-likelihood for a single training example (x, y).

Recall:

$$\ell(\theta) = y \log(h_\theta(x)) + (1 - y) \log(1 - h_\theta(x)), \quad \text{where} \quad h_\theta(x) = \sigma(\theta^T x).$$

We compute the partial derivative with respect to θ_j:

$$\frac{\partial}{\partial \theta_j} \ell(\theta) = y \cdot \frac{1}{h_\theta(x)} \cdot \frac{\partial h_\theta(x)}{\partial \theta_j} + (1 - y) \cdot \frac{1}{1 - h_\theta(x)} \cdot \left(- \frac{\partial h_\theta(x)}{\partial \theta_j} \right).$$

Factoring:

$$\frac{\partial \ell(\theta)}{\partial \theta_j} = \left(\frac{y}{h_\theta(x)} - \frac{1 - y}{1 - h_\theta(x)} \right) \cdot \frac{\partial h_\theta(x)}{\partial \theta_j}.$$

Now recall the derivative of the sigmoid function:

$$\frac{d}{dz} \sigma(z) = \sigma(z)(1 - \sigma(z)), \quad \text{and} \quad \frac{\partial h_\theta(x)}{\partial \theta_j} = h_\theta(x)(1 - h_\theta(x)) \cdot x_j.$$

Putting it all together:

$$\frac{\partial \ell(\theta)}{\partial \theta_j} = \left(\frac{y}{h_\theta(x)} - \frac{1 - y}{1 - h_\theta(x)} \right) \cdot h_\theta(x)(1 - h_\theta(x)) \cdot x_j.$$

This simplifies to:

$$\frac{\partial \ell(\theta)}{\partial \theta_j} = (y - h_\theta(x)) \cdot x_j.$$

Therefore, the **stochastic gradient ascent** update rule (using one training example at a time) is:

$$\theta_j := \theta_j + \alpha (y - h_\theta(x)) x_j.$$

More compactly in vector form:

$$\theta := \theta + \alpha (y - h_\theta(x)) x.$$

When using the full dataset (batch gradient ascent), we average the updates across all m examples:

$$\theta := \theta + \alpha \sum_{i=1}^{m} (y^{(i)} - h_\theta(x^{(i)}))x^{(i)}.$$

This completes the derivation of the logistic regression update rule using gradient ascent on the log-likelihood.

4 Gradient Ascent for Logistic Regression

Algorithm 1 Gradient Ascent for Logistic Regression

Require: Dataset $\{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$, learning rate α, iterations T
Ensure: Optimized parameters θ

1: Initialize $\theta \leftarrow 0$
2: **for** each iteration $t = 1, \ldots, T$ **do**
3: Compute predictions: $h_\theta(x^{(i)}) = \sigma(\theta^T x^{(i)})$
4: Compute gradient of log-likelihood:

$$\nabla \ell(\theta) = \sum_{i=1}^{m} \left(y^{(i)} - h_\theta(x^{(i)})\right) x^{(i)}$$

5: Update parameters using gradient ascent:

$$\theta \leftarrow \theta + \alpha \nabla \ell(\theta)$$

6: **end for**

Cross-Entropy Loss: A Simple Example

Consider a binary classification problem where the true label y for a single data point is 1 (positive class). Thus, the true distribution is:

$$p = \begin{cases} 1 & \text{if } y = 1 \\ 0 & \text{if } y = 0 \end{cases}$$

Suppose the model predicts the probability of the positive class as $q = 0.9$. Then, the predicted probability of the negative class is $1 - q = 0.1$.
The cross-entropy loss for this example is given by

$$H(p, q) = -\big(y \log q + (1 - y) \log(1 - q)\big).$$

Since $y = 1$, this simplifies to

$$H(p, q) = -\log(0.9) \approx 0.105.$$

If instead the model predicts $q = 0.1$ (incorrect prediction), the cross-entropy loss becomes

$$H(p, q) = -\log(0.1) \approx 2.302.$$

Interpretation: When the model assigns a high probability to the true class, the cross-entropy loss is low, indicating a good prediction. Conversely, when the model assigns a low probability to the true class, the loss is high, signaling a poor prediction.

5 Example: Email Spam Detection

Example: Step-by-Step Gradient Ascent

Consider the following dataset with two email samples, where the feature is the count of spam keywords:

$$\begin{cases} x^{(1)} = [1, 50], & y^{(1)} = 1 \quad \text{(spam)} \\ x^{(2)} = [1, 2], & y^{(2)} = 0 \quad \text{(not spam)} \end{cases}$$

We want to fit $\theta = [\theta_0, \theta_1]$ using gradient ascent.

Initialization: $\theta^{(0)} = [0, 0]$, learning rate $\alpha = 0.1$.

Iteration 1:

- Compute predictions:

$$h_{\theta^{(0)}}(x^{(1)}) = \sigma(0 \times 1 + 0 \times 50) = 0.5$$

$$h_{\theta^{(0)}}(x^{(2)}) = \sigma(0 \times 1 + 0 \times 2) = 0.5$$

- Compute gradient:

$$\begin{aligned} \nabla \ell(\theta^{(0)}) &= (y^{(1)} - h_{\theta^{(0)}}(x^{(1)}))x^{(1)} + (y^{(2)} - h_{\theta^{(0)}}(x^{(2)}))x^{(2)} \\ &= (1 - 0.5) \times [1, 50] + (0 - 0.5) \times [1, 2] \\ &= [0.5, 25] + [-0.5, -1] = [0, 24] \end{aligned}$$

- Update parameters:

$$\theta^{(1)} = \theta^{(0)} + \alpha \nabla \ell(\theta^{(0)}) = [0, 0] + 0.1 \times [0, 24] = [0, 2.4]$$

After the first iteration, the weight corresponding to the number of spam keywords increased to 2.4, indicating that the model starts to associate higher counts with spam.

Interpretation of the First Update

After the first iteration, the updated parameters are:

$$\theta^{(1)} = [0, 2.4]$$

This means:

- The intercept term θ_0 remains 0, indicating no baseline bias change yet.
- The weight for the spam keyword count, θ_1, increased to 2.4.

Since θ_1 is positive and relatively large, the model is learning that as the number of spam keywords increases, the probability that an email is spam also increases.
To see this concretely, compute the new prediction for the second email $x^{(2)} = [1, 2]$:

$$h_{\theta^{(1)}}(x^{(2)}) = \sigma(0 \times 1 + 2.4 \times 2) = \sigma(4.8) \approx 0.991$$

Originally, this email had a predicted probability of 0.5 of being spam. Now, the model predicts a 99.1% chance it is spam, which is incorrect since $y^{(2)} = 0$. This shows the model has begun to overfit to the first example but has not yet learned to differentiate well.

In further iterations, the parameters will continue to adjust to better fit both examples, decreasing the predicted spam probability for $x^{(2)}$ and increasing it for $x^{(1)}$ appropriately.

6 Decision Boundary

A logistic model defines a decision boundary where:

$$\theta^T x = 0 \quad \text{or} \quad h_\theta(x) = 0.5.$$

This boundary separates the feature space into two regions: predicted 0 or 1.

Note

If data is linearly separable, logistic regression can classify perfectly. For nonlinear cases, feature engineering or kernel methods are needed.

7 Multiclass Logistic Regression (Softmax)

To extend to $K > 2$ classes, we use the **softmax function**:

$$P(y = k \mid x) = \frac{e^{\theta_k^T x}}{\sum_{j=1}^{K} e^{\theta_j^T x}}.$$

This is called **Multinomial Logistic Regression**.

8 Conclusion

Logistic regression is a simple, powerful classification model with a solid statistical foundation. It is interpretable, fast to train, and effective for binary (and multiclass) problems.

Note

Despite its simplicity, logistic regression is widely used in industry for problems like credit scoring, spam filtering, and medical diagnosis.

9 References

References

[1] A. Ng. (2018). *Machine Learning Course*, Stanford University. https://www.coursera.org/learn/machine-learning