QA from GPT

Lecture 6 - Logistic regression
Quanying Liu
2024.03.12

0. What is Logistic regression?

Logistic regression is a statistical method used for analyzing a dataset in which there are one or more independent variables that determine an outcome. The outcome is measured with a dichotomous variable (in which there are only two possible outcomes). It is used extensively in various fields, including machine learning, most often for binary classification problems.

Here's a more detailed breakdown of logistic regression:

1. **Binary Outcome**: Logistic regression is used when the dependent variable is binary in nature. For example, it could be used to predict whether an email is spam (1) or not spam (0), or whether a tumor is malignant (1) or benign (0).

2. **Odds and Logit Function**: Unlike linear regression, which predicts a continuous outcome, logistic regression estimates the probability of a binary event occurring. To deal with the issue of predicting a binary outcome, logistic regression calculates the odds of the event occurring (probability of the event divided by the probability of the event not occurring), and then takes the logarithm of the odds (the logit).

3. **Sigmoid Function**: The core of logistic regression is the sigmoid function, which maps any input into a value between 0 and 1, making it suitable for calculating probabilities. The sigmoid function is defined as \(\sigma(z) = \frac{1}{1 + e^{-z}} \), where \(z \) is the input to the function, typically the linear combination of the input features weighted by the model's coefficients (plus a bias term).

4. **Estimation**: The coefficients (weights) in logistic regression are typically estimated using maximum likelihood estimation. This method seeks to find the set of coefficients that maximizes the likelihood of the observed set of outcomes given the input features.

5. **Interpretation**: The coefficients in logistic regression represent the change in the log odds of the outcome for a one-unit change in the predictor variable. For example, in a logistic regression model predicting the likelihood of having a disease based on age, a coefficient of 0.05 for age would mean that each one-year increase in age increases the log odds of having the disease by 0.05.

6. **Model Evaluation**: The performance of a logistic regression model can be evaluated using metrics like accuracy, precision, recall, F1-score, and the area under the ROC curve

(AUC-ROC). Confusion matrices are also commonly used to understand the performance of the model in terms of true positives, false positives, true negatives, and false negatives.

Logistic regression is a powerful tool for binary classification problems, but it has its limitations, such as the assumption of linearity between the log odds of the outcome and the input variables, and the requirement that the observations be independent of each other. Despite these limitations, it remains a popular choice due to its simplicity, interpretability, and the fact that it can serve as a baseline for more complex models.

为什么逻辑回归被叫做回归，而不是分类？

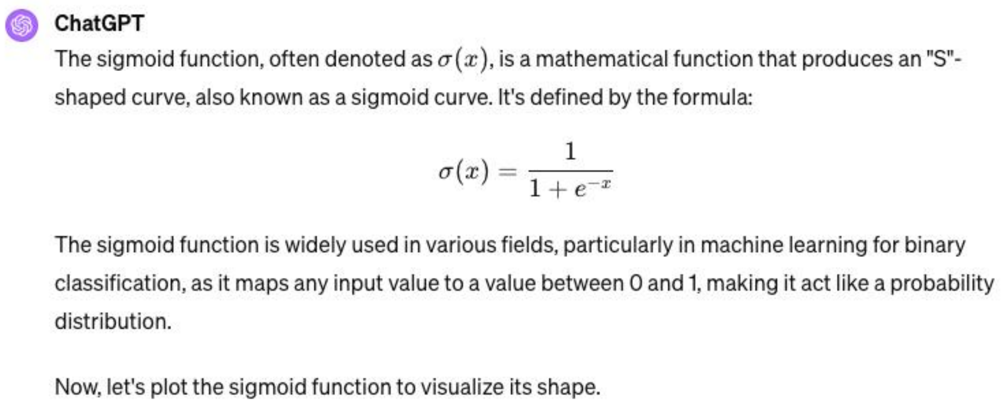
逻辑回归虽然主要用于分类问题，特别是二分类问题，但之所以被称为“回归”，是因为它在本质上是通过**估计概率**来进行回归分析的。这里的“回归”指的是对概率的连续预测，而不是直接对类别的预测。

逻辑回归的目标是找到一个能够预测出现某个结果的概率的模型，这个概率值是连续的，从 0 到 1 变化。在逻辑回归中，我们试图建立一个模型来预测给定输入变量时，输出变量（通常是二元的，如 0 或 1）为 1 的概率。这个概率是通过对输入特征的线性组合应用 sigmoid 函数来计算的，这个过程本质上是一种回归。

更具体地说，逻辑回归模型首先计算特征的加权和（加上一个偏置项），然后将这个加权和通过 sigmoid 函数转换为一个介于 0 和 1 之间的值。这个值被解释为属于类别 1 的概率。尽管最终的输出被用于分类（通常是通过设置一个阈值，比如 0.5，概率高于这个阈值的被分类为 1，低于则被分类为 0），但模型本身是通过估计一个连续的概率来实现这一点的，这就是为什么它被称为“回归”。

总结来说，尽管逻辑回归常用于分类任务，但它被称为“回归”是因为它实际上是在对概率进行回归分析，即在连续的概率空间中寻找一个最佳的连续函数来预测一个事件发生的概率。

1. what is Sigmoid function? can you plot it?



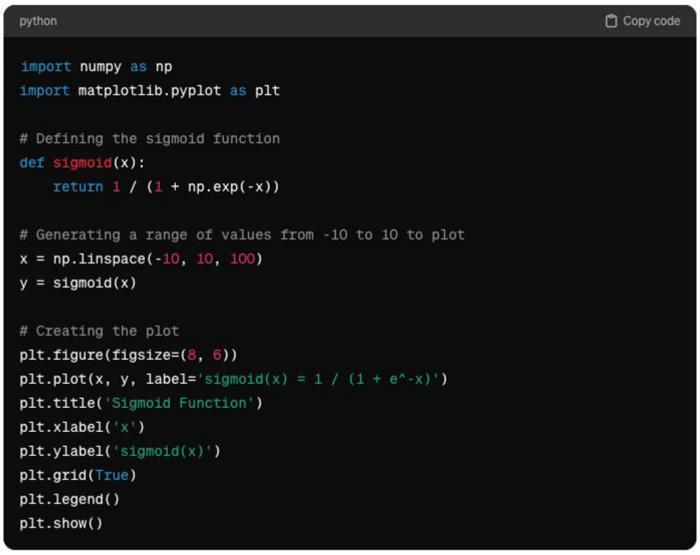
ChatGPT

The sigmoid function, often denoted as $\sigma(x)$, is a mathematical function that produces an "S"-shaped curve, also known as a sigmoid curve. It's defined by the formula:

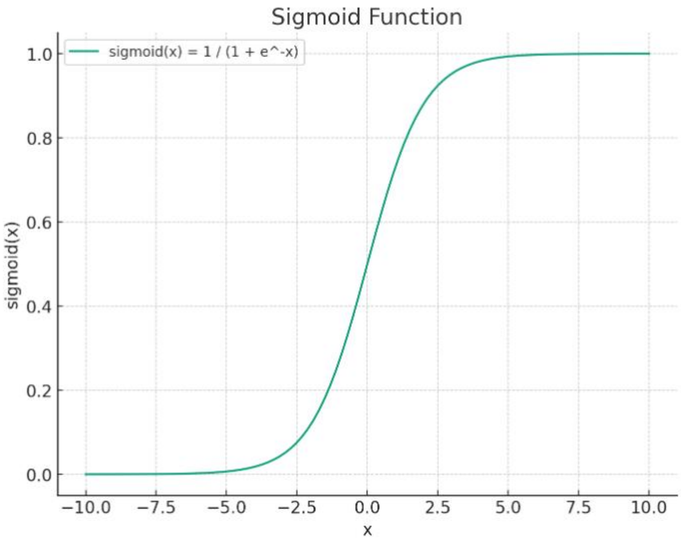
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

The sigmoid function is widely used in various fields, particularly in machine learning for binary classification, as it maps any input value to a value between 0 and 1, making it act like a probability distribution.

Now, let's plot the sigmoid function to visualize its shape.



```python
import numpy as np
import matplotlib.pyplot as plt

# Defining the sigmoid function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Generating a range of values from -10 to 10 to plot
x = np.linspace(-10, 10, 100)
y = sigmoid(x)

# Creating the plot
plt.figure(figsize=(8, 6))
plt.plot(x, y, label='sigmoid(x) = 1 / (1 + e^-x)')
plt.title('Sigmoid Function')
plt.xlabel('x')
plt.ylabel('sigmoid(x)')
plt.grid(True)
plt.legend()
plt.show()
```



Here's the plot of the sigmoid function, $\sigma(x) = \frac{1}{1+e^{-x}}$. As you can see, it produces an "S"-shaped curve, where the output values range between 0 and 1, making it ideal for applications like logistic regression in machine learning where it can represent probabilities.

2. In machine learning, why do we often use the sigmoid function to estimate probability?

The sigmoid function is widely used in machine learning, particularly in binary classification tasks, to estimate probabilities due to several key properties:

1. **Output Range**: The sigmoid function maps any input value to a range between 0 and 1, making its output interpretable as a probability. This is particularly useful for binary classification, where you need to predict the probability of an input belonging to a certain class.

2. **Shape**: The S-shaped curve of the sigmoid function provides a natural transition from 0 to 1. As the input gets larger, the output asymptotically approaches 1, and as the input gets smaller, the output asymptotically approaches 0. This gradual transition is useful for

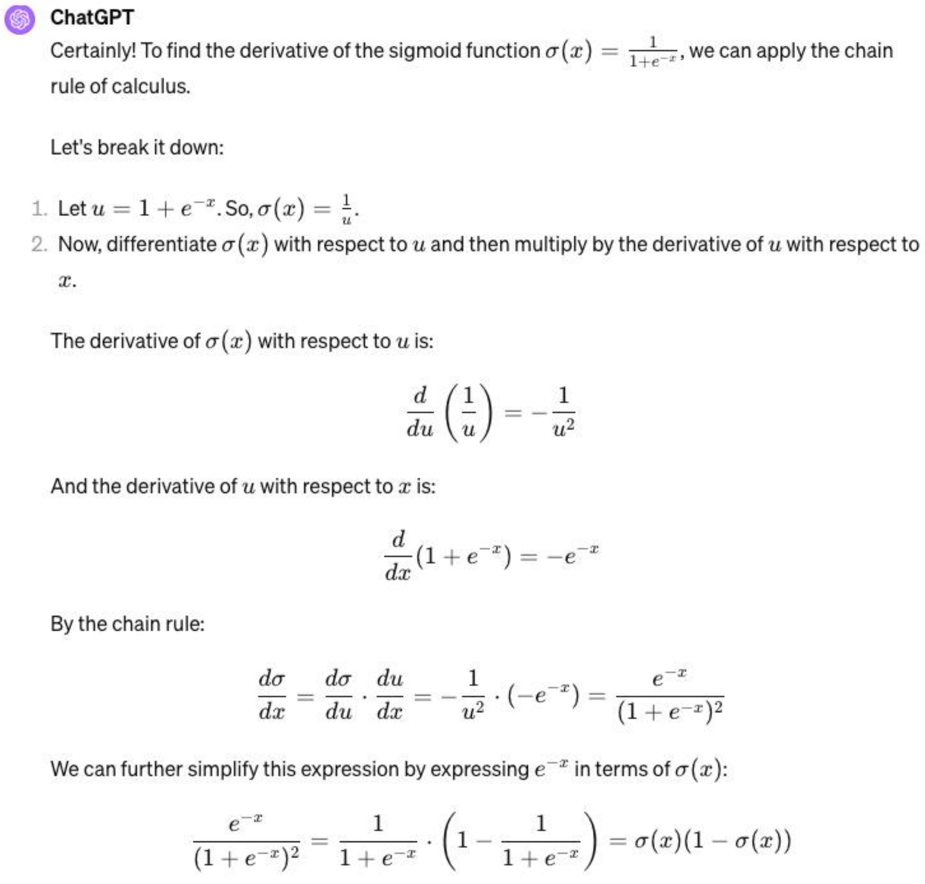
distinguishing between the two classes in binary classification.

3. **Differentiability**: The sigmoid function is smooth and differentiable everywhere, which is crucial for optimization algorithms like gradient descent. Since many machine learning algorithms rely on gradient-based optimization to find the best parameters for the model, having a differentiable function ensures that we can compute gradients and update our model's parameters.

4. **Gradient Values**: The gradient of the sigmoid function has nice properties—it is at its maximum when the input is 0, and it decreases as the input moves away from 0. This property is useful during training, as it means the model can learn more effectively when it's unsure (when the output is around 0.5) and make smaller adjustments when it's more confident (when the output is near 0 or 1).

However, it's worth noting that the sigmoid function isn't perfect. It can lead to issues like vanishing gradients, where the gradients become very small for extreme input values, slowing down the learning process. This is one reason why other functions like the **softmax function** for multi-class classification or the **ReLU** (Rectified Linear Unit) function for hidden layers in neural networks have become popular alternatives in certain contexts.

3. Can you derive the derivative of a sigmoid function? Can you plot it?



ChatGPT

Certainly! To find the derivative of the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$, we can apply the chain rule of calculus.

Let's break it down:

1. Let $u = 1 + e^{-x}$. So, $\sigma(x) = \frac{1}{u}$.
2. Now, differentiate $\sigma(x)$ with respect to u and then multiply by the derivative of u with respect to x.

The derivative of $\sigma(x)$ with respect to u is:

$$\frac{d}{du}\left(\frac{1}{u}\right) = -\frac{1}{u^2}$$

And the derivative of u with respect to x is:

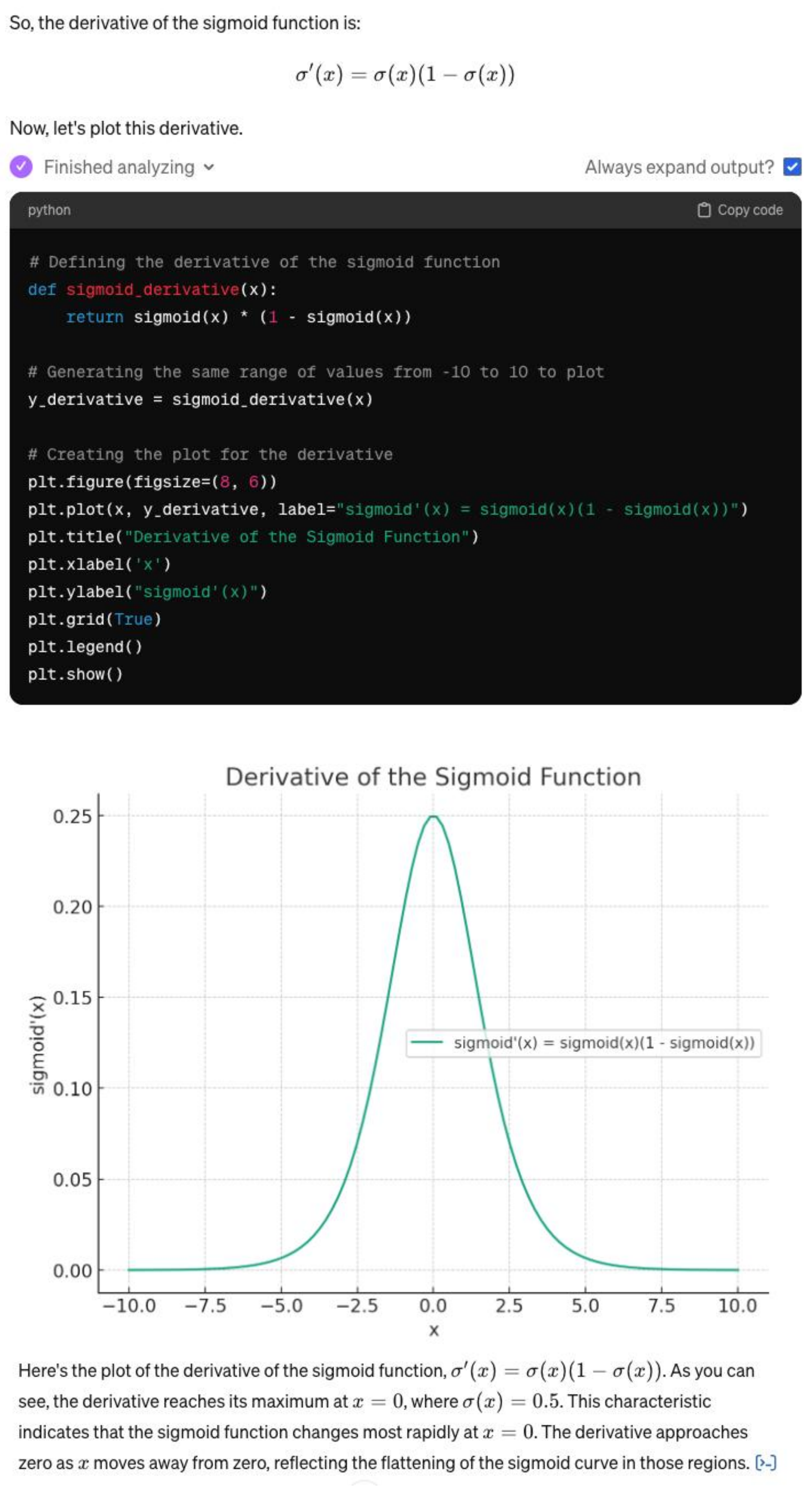
$$\frac{d}{dx}(1 + e^{-x}) = -e^{-x}$$

By the chain rule:

$$\frac{d\sigma}{dx} = \frac{d\sigma}{du} \cdot \frac{du}{dx} = -\frac{1}{u^2} \cdot (-e^{-x}) = \frac{e^{-x}}{(1 + e^{-x})^2}$$

We can further simplify this expression by expressing e^{-x} in terms of $\sigma(x)$:

$$\frac{e^{-x}}{(1 + e^{-x})^2} = \frac{1}{1 + e^{-x}} \cdot \left(1 - \frac{1}{1 + e^{-x}}\right) = \sigma(x)(1 - \sigma(x))$$



So, the derivative of the sigmoid function is:

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

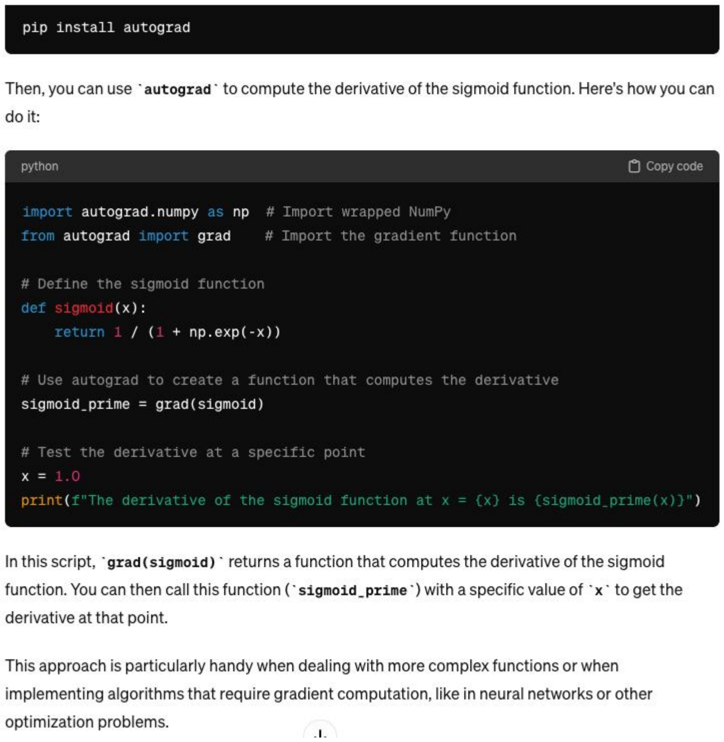
Now, let's plot this derivative.

```python
# Defining the derivative of the sigmoid function
def sigmoid_derivative(x):
    return sigmoid(x) * (1 - sigmoid(x))

# Generating the same range of values from -10 to 10 to plot
y_derivative = sigmoid_derivative(x)

# Creating the plot for the derivative
plt.figure(figsize=(8, 6))
plt.plot(x, y_derivative, label="sigmoid'(x) = sigmoid(x)(1 - sigmoid(x))")
plt.title("Derivative of the Sigmoid Function")
plt.xlabel('x')
plt.ylabel("sigmoid'(x)")
plt.grid(True)
plt.legend()
plt.show()
```

Derivative of the Sigmoid Function
sigmoid'(x)
0.25
0.20
0.15
0.10
0.05
0.00
sigmoid'(x) = sigmoid(x)(1 - sigmoid(x))
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
x

Here's the plot of the derivative of the sigmoid function, $\sigma'(x) = \sigma(x)(1 - \sigma(x))$. As you can see, the derivative reaches its maximum at $x = 0$, where $\sigma(x) = 0.5$. This characteristic indicates that the sigmoid function changes most rapidly at $x = 0$. The derivative approaches zero as x moves away from zero, reflecting the flattening of the sigmoid curve in those regions.

注意：另一种办法，autograd，自动计算梯度，而无需手动导出导数函数。



```
pip install autograd
```

Then, you can use `autograd` to compute the derivative of the sigmoid function. Here's how you can do it:

```python
import autograd.numpy as np  # Import wrapped NumPy
from autograd import grad    # Import the gradient function

# Define the sigmoid function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Use autograd to create a function that computes the derivative
sigmoid_prime = grad(sigmoid)

# Test the derivative at a specific point
x = 1.0
print(f"The derivative of the sigmoid function at x = {x} is {sigmoid_prime(x)}")
```

In this script, `grad(sigmoid)` returns a function that computes the derivative of the sigmoid function. You can then call this function (`sigmoid_prime`) with a specific value of `x` to get the derivative at that point.

This approach is particularly handy when dealing with more complex functions or when implementing algorithms that require gradient computation, like in neural networks or other optimization problems.

4. What is Bernoulli distribution? Take some examples?

The Bernoulli distribution is a discrete probability distribution for a random variable that can take one of two possible outcomes, often labeled as 1 (success) and 0 (failure). It is a simple distribution with only one parameter, \(p \), which represents the probability of success. The probability of failure is then \(1 - p \). The Bernoulli distribution is a special case of the binomial distribution where there is only one trial.

The probability mass function (PMF) of a Bernoulli random variable \(X \) is given by:

$$P(X = x) = \begin{cases} p & \text{if } x = 1, \\ 1 - p & \text{if } x = 0. \end{cases}$$

Here are some examples of phenomena that can be modeled with a Bernoulli distribution:

1. **Coin Toss**: A simple example is a fair coin toss where the outcome can be head or tail. If we define "head" as a success (1) and "tail" as a failure (0), and assuming the coin is fair, then \(p = 0.5 \). This is a Bernoulli trial because there are only two possible outcomes.

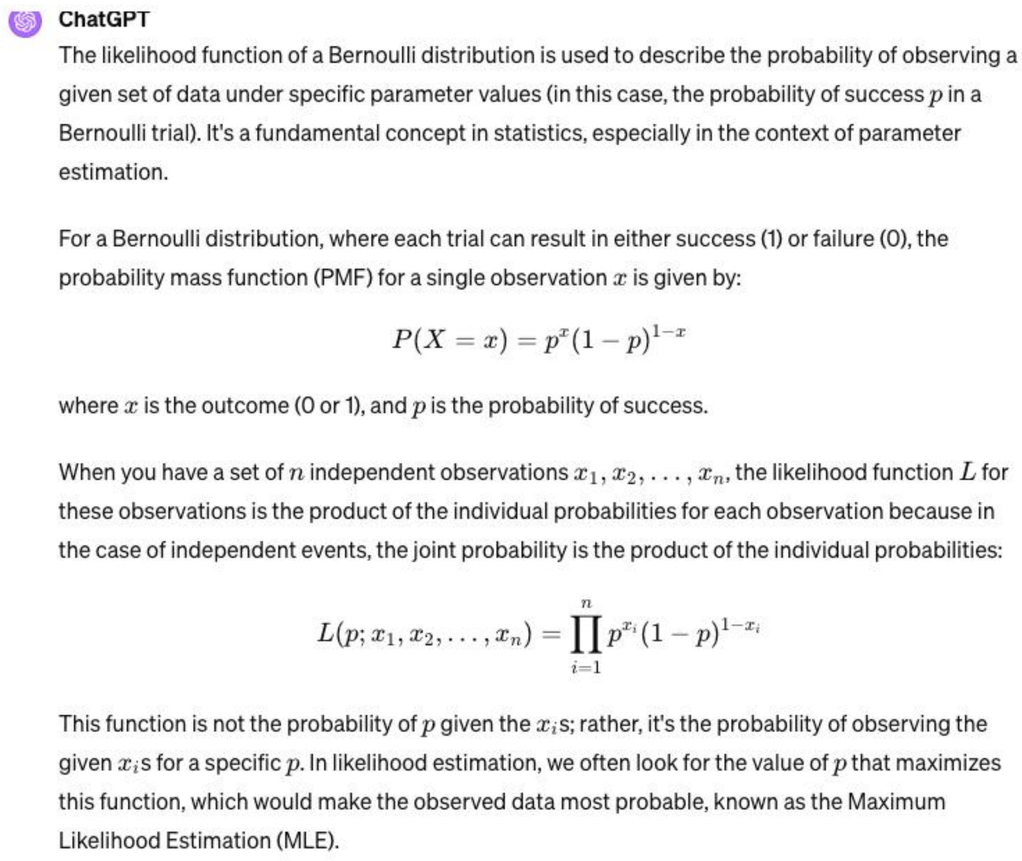
2. **Quality Control**: Consider a quality control inspection where each item is classified as defective (1) or non-defective (0). If 2% of items are defective, then an item picked at random can be modeled with a Bernoulli distribution where \(p = 0.02 \).

3. **Medical Treatment**: In a clinical trial for a new medication, a patient could either respond positively to the treatment (1) or not (0). If the probability of a positive response is known from previous studies or is being tested in the current study, the outcome for any given patient can be described by a Bernoulli distribution.

4. **Click-through Rate**: In online advertising, the click-through rate (CTR) is a common metric. Each ad impression can result in a click (1) or no click (0). If historical data suggests that the probability of an ad being clicked is \(p \), then each ad impression follows a Bernoulli distribution with that probability.

These examples illustrate how the Bernoulli distribution can model various real-world scenarios where the outcome of an event is binary.

5. What is likelihood function of a Bernoulli distribution?



ChatGPT

The likelihood function of a Bernoulli distribution is used to describe the probability of observing a given set of data under specific parameter values (in this case, the probability of success p in a Bernoulli trial). It's a fundamental concept in statistics, especially in the context of parameter estimation.

For a Bernoulli distribution, where each trial can result in either success (1) or failure (0), the probability mass function (PMF) for a single observation x is given by:

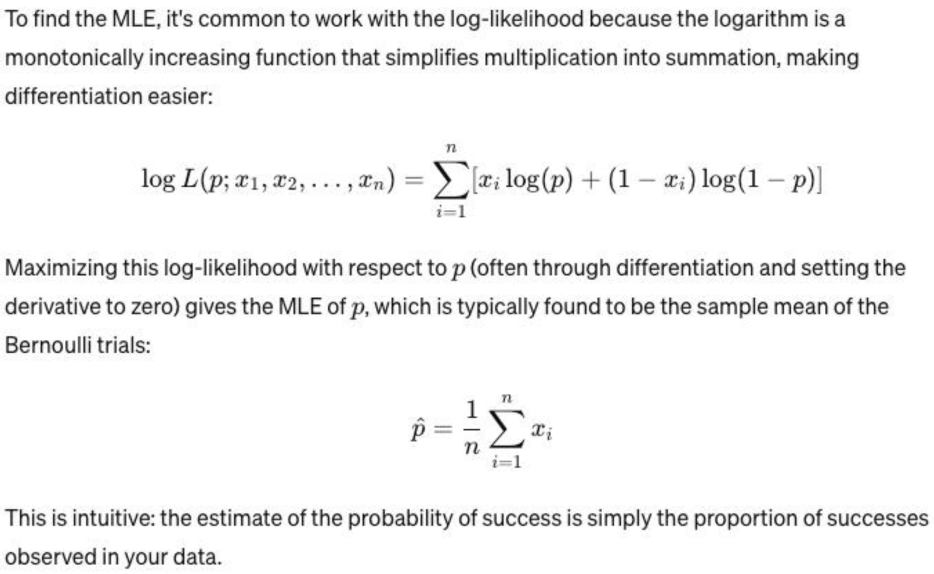
$$P(X = x) = p^x(1 - p)^{1-x}$$

where x is the outcome (0 or 1), and p is the probability of success.

When you have a set of n independent observations $x_1, x_2, \ldots, x_n$, the likelihood function L for these observations is the product of the individual probabilities for each observation because in the case of independent events, the joint probability is the product of the individual probabilities:

$$L(p; x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} p^{x_i}(1 - p)^{1-x_i}$$

This function is not the probability of p given the x_is; rather, it's the probability of observing the given x_is for a specific p. In likelihood estimation, we often look for the value of p that maximizes this function, which would make the observed data most probable, known as the Maximum Likelihood Estimation (MLE).



To find the MLE, it's common to work with the log-likelihood because the logarithm is a monotonically increasing function that simplifies multiplication into summation, making differentiation easier:

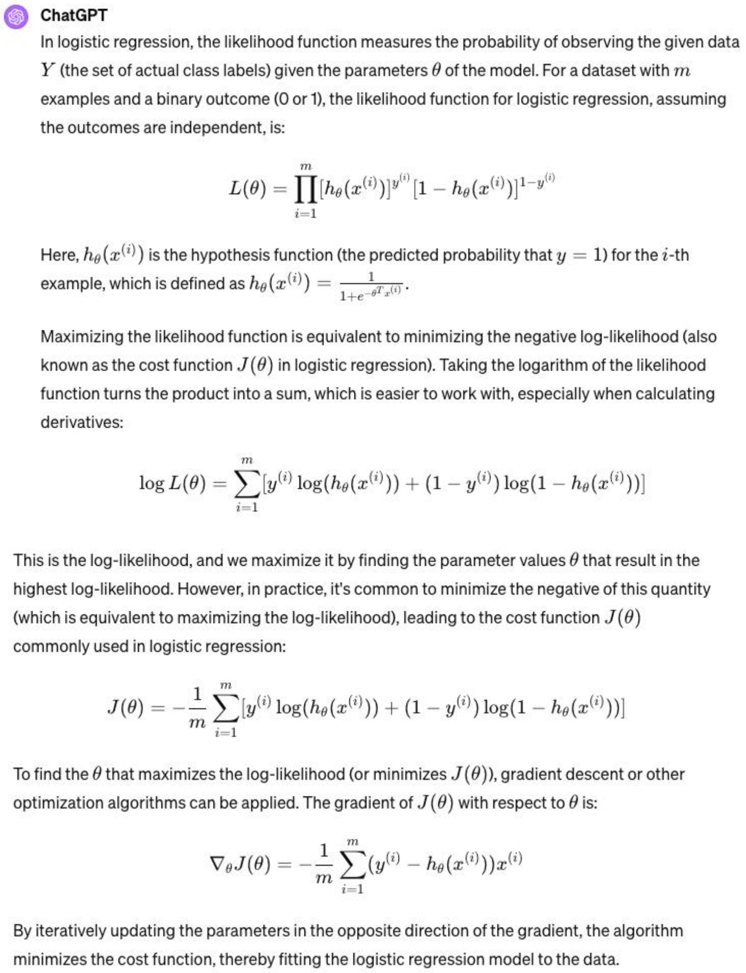
$$\log L(p; x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} [x_i \log(p) + (1 - x_i) \log(1 - p)]$$

Maximizing this log-likelihood with respect to p (often through differentiation and setting the derivative to zero) gives the MLE of p, which is typically found to be the sample mean of the Bernoulli trials:

$$\hat{p} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

This is intuitive: the estimate of the probability of success is simply the proportion of successes observed in your data.

6. Please maximize log-likelihood of logistic regression?



ChatGPT

In logistic regression, the likelihood function measures the probability of observing the given data Y (the set of actual class labels) given the parameters θ of the model. For a dataset with m examples and a binary outcome (0 or 1), the likelihood function for logistic regression, assuming the outcomes are independent, is:

$$L(\theta) = \prod_{i=1}^{m} [h_\theta(x^{(i)})]^{y^{(i)}} [1 - h_\theta(x^{(i)})]^{1-y^{(i)}}$$

Here, $h_\theta(x^{(i)})$ is the hypothesis function (the predicted probability that $y = 1$) for the i-th example, which is defined as $h_\theta(x^{(i)}) = \frac{1}{1+e^{-\theta^T x^{(i)}}}$.

Maximizing the likelihood function is equivalent to minimizing the negative log-likelihood (also known as the cost function $J(\theta)$ in logistic regression). Taking the logarithm of the likelihood function turns the product into a sum, which is easier to work with, especially when calculating derivatives:

$$\log L(\theta) = \sum_{i=1}^{m} [y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))]$$

This is the log-likelihood, and we maximize it by finding the parameter values θ that result in the highest log-likelihood. However, in practice, it's common to minimize the negative of this quantity (which is equivalent to maximizing the log-likelihood), leading to the cost function $J(\theta)$ commonly used in logistic regression:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} [y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))]$$

To find the θ that maximizes the log-likelihood (or minimizes $J(\theta)$), gradient descent or other optimization algorithms can be applied. The gradient of $J(\theta)$ with respect to θ is:

$$\nabla_\theta J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} (y^{(i)} - h_\theta(x^{(i)})) x^{(i)}$$

By iteratively updating the parameters in the opposite direction of the gradient, the algorithm minimizes the cost function, thereby fitting the logistic regression model to the data.

7. Can you have an analytical solution of it?

In logistic regression, unlike linear regression, there is no closed-form or analytical solution for the optimal parameters \(\theta \) that maximize the likelihood function. This is due to the **non-linear nature** of the logistic function, which leads to a cost function \(J(\theta) \) that is not a quadratic function of \(\theta \). Therefore, we cannot apply a direct analytical solution like the normal equation used in linear regression.

Instead, the optimization of the cost function in logistic regression is typically performed using iterative optimization methods such as:

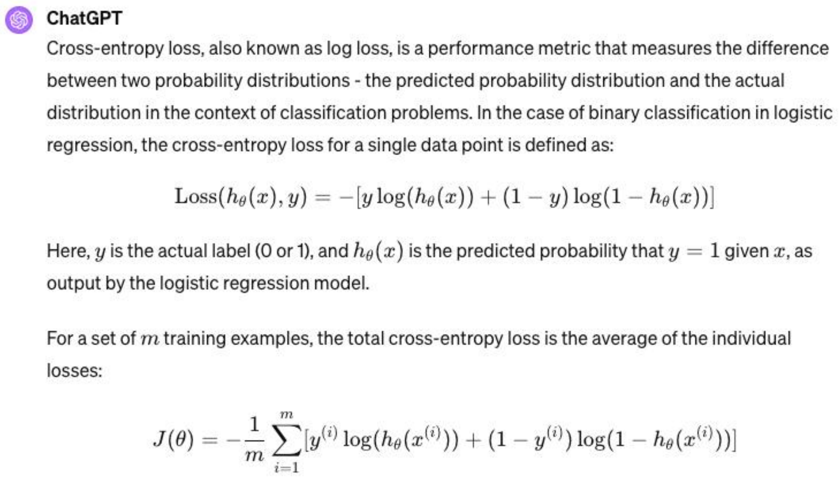
1. **Gradient Descent**: The most common approach is gradient descent, where the parameters are updated iteratively based on the gradient of the cost function.

2. **Stochastic Gradient Descent (SGD)**: A variant of gradient descent where the parameters are updated for each training example or small batches. It's particularly useful for large datasets.

3. **Advanced Optimization Algorithms**: Methods like Conjugate Gradient, BFGS, or L-BFGS are more sophisticated optimization algorithms that can converge faster than gradient descent, especially for large-scale problems.

These methods rely on the iterative update of the weights and do not provide a direct formula to compute the optimal parameters. The iterative process continues until the cost function converges to a minimum, indicating that the best possible parameters \(\theta \) under the logistic regression model have been found given the training data.

8. What is cross-entropy loss? Why we use it in logistic regression, rather than MSE loss?



ChatGPT

Cross-entropy loss, also known as log loss, is a performance metric that measures the difference between two probability distributions - the predicted probability distribution and the actual distribution in the context of classification problems. In the case of binary classification in logistic regression, the cross-entropy loss for a single data point is defined as:

$$\text{Loss}(h_\theta(x), y) = -[y \log(h_\theta(x)) + (1-y)\log(1-h_\theta(x))]$$

Here, y is the actual label (0 or 1), and $h_\theta(x)$ is the predicted probability that $y = 1$ given x, as output by the logistic regression model.

For a set of m training examples, the total cross-entropy loss is the average of the individual losses:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))]$$

We use cross-entropy loss in logistic regression rather than mean squared error (MSE) loss for several reasons:

1. **Probability Interpretation**: Logistic regression models the probability of the target variable. The cross-entropy loss is naturally suited for measuring the difference between two probability distributions, making it a more appropriate choice than MSE.

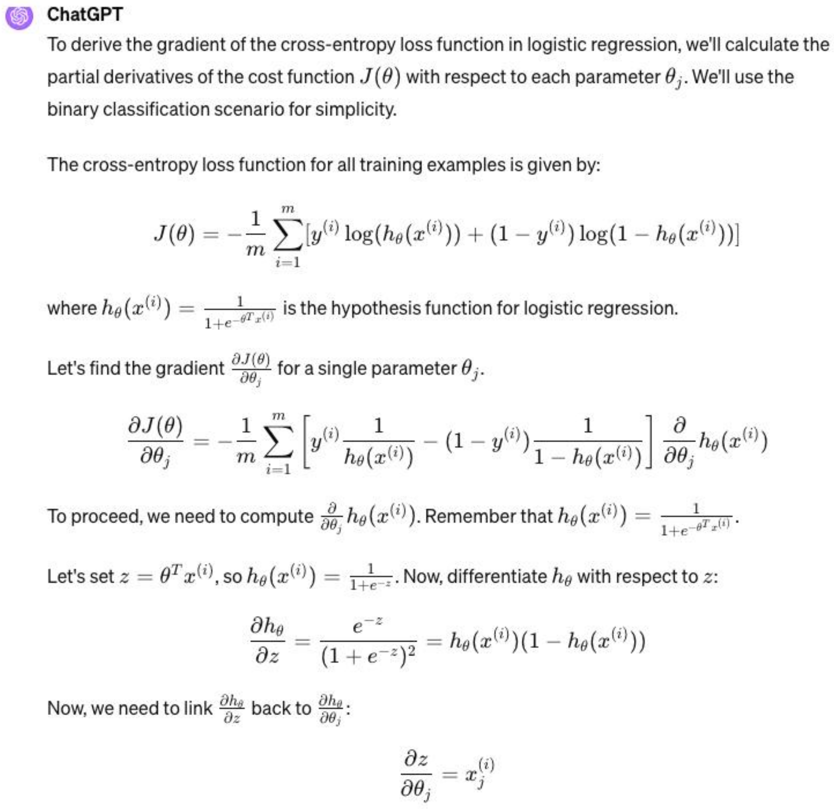
2. **Gradient Behavior**: The gradient of the cross-entropy loss with respect to the model parameters is more stable and less likely to be affected by the vanishing or exploding gradients problem, especially compared to MSE. This stability helps in more effective and reliable training of the model using gradient-based optimization methods.

3. **Non-convexity of MSE**: For logistic regression, using MSE as the loss function leads to a non-convex cost function with multiple local minima. This non-convexity can make the optimization process dependent on the initial values and potentially lead to suboptimal solutions. In contrast, the cross-entropy loss results in a convex cost function for logistic regression, ensuring that gradient descent methods converge to the global minimum.

4. **Model Calibration**: Cross-entropy loss directly penalizes the model for the confidence of its predictions in addition to their accuracy. This means that it not only encourages correct predictions but also discourages overconfident wrong predictions, leading to better-calibrated probability estimates.

In summary, cross-entropy loss is preferred in logistic regression because it aligns with the probabilistic outputs of the model, provides stable gradients for optimization, ensures convexity in the loss function, and encourages well-calibrated predicted probabilities.

9. Please derive the gradient of the cross-entropy loss function in logistic regression?



ChatGPT

To derive the gradient of the cross-entropy loss function in logistic regression, we'll calculate the partial derivatives of the cost function $J(\theta)$ with respect to each parameter θ_j. We'll use the binary classification scenario for simplicity.

The cross-entropy loss function for all training examples is given by:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))]$$

where $h_\theta(x^{(i)}) = \frac{1}{1+e^{-\theta^T x^{(i)}}}$ is the hypothesis function for logistic regression.

Let's find the gradient $\frac{\partial J(\theta)}{\partial \theta_j}$ for a single parameter θ_j.

$$\frac{\partial J(\theta)}{\partial \theta_j} = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\frac{1}{h_\theta(x^{(i)})} - (1-y^{(i)})\frac{1}{1-h_\theta(x^{(i)})}\right]\frac{\partial}{\partial \theta_j}h_\theta(x^{(i)})$$

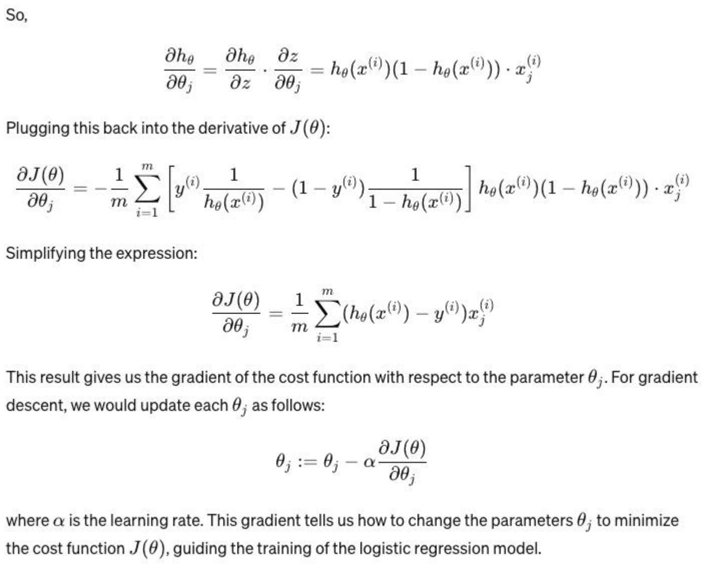
To proceed, we need to compute $\frac{\partial}{\partial \theta_j}h_\theta(x^{(i)})$. Remember that $h_\theta(x^{(i)}) = \frac{1}{1+e^{-\theta^T x^{(i)}}}$.

Let's set $z = \theta^T x^{(i)}$, so $h_\theta(x^{(i)}) = \frac{1}{1+e^{-z}}$. Now, differentiate h_θ with respect to z:

$$\frac{\partial h_\theta}{\partial z} = \frac{e^{-z}}{(1+e^{-z})^2} = h_\theta(x^{(i)})(1-h_\theta(x^{(i)}))$$

Now, we need to link $\frac{\partial h_\theta}{\partial z}$ back to $\frac{\partial h_\theta}{\partial \theta_j}$:

$$\frac{\partial z}{\partial \theta_j} = x_j^{(i)}$$



So,

$$\frac{\partial h_\theta}{\partial \theta_j} = \frac{\partial h_\theta}{\partial z}\cdot\frac{\partial z}{\partial \theta_j} = h_\theta(x^{(i)})(1-h_\theta(x^{(i)}))\cdot x_j^{(i)}$$

Plugging this back into the derivative of $J(\theta)$:

$$\frac{\partial J(\theta)}{\partial \theta_j} = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\frac{1}{h_\theta(x^{(i)})} - (1-y^{(i)})\frac{1}{1-h_\theta(x^{(i)})}\right]h_\theta(x^{(i)})(1-h_\theta(x^{(i)}))\cdot x_j^{(i)}$$

Simplifying the expression:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

This result gives us the gradient of the cost function with respect to the parameter θ_j. For gradient descent, we would update each θ_j as follows:

$$\theta_j := \theta_j - \alpha\frac{\partial J(\theta)}{\partial \theta_j}$$

where α is the learning rate. This gradient tells us how to change the parameters θ_j to minimize the cost function $J(\theta)$, guiding the training of the logistic regression model.

10. Please write the python code using Gradient Descent to learn parameters of logistic regression.

Sure, I'll provide a simple Python implementation of logistic regression using gradient descent. This code will illustrate how to learn the parameters (\(\theta\)) for a logistic regression model on a hypothetical binary classification dataset.

```python
import numpy as np

# Sigmoid function
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Logistic Regression Cost Function
def compute_cost(X, y, theta):
    m = len(y)
    h = sigmoid(X @ theta)
    epsilon = 1e-5    # to avoid log(0)
    cost = (1/m) * ((-y).T @ np.log(h + epsilon) - (1 - y).T @ np.log(1 - h + epsilon))
    return cost

# Gradient Descent Function to minimize the logistic regression cost function
def gradient_descent(X, y, theta, alpha, num_iterations):
    m = len(y)
    for i in range(num_iterations):
        gradient = (1/m) * X.T @ (sigmoid(X @ theta) - y)
        theta = theta - alpha * gradient
    return theta

# Data Preparation
# Let's assume X is our input features and y is our output variable.
# Add an intercept term with X
np.random.seed(0)    # for reproducibility
X = np.random.rand(100, 1)    # 100 examples and 1 feature
intercept = np.ones((X.shape[0], 1))
X = np.concatenate((intercept, X), axis=1)
y = (X[:, 1] > 0.5).astype(int)    # simple threshold for binary classification

# Parameters initialization
theta = np.zeros(X.shape[1])

# Hyperparameters
```

```
alpha = 0.01    # Learning rate
num_iterations = 1000    # Number of iterations

# Run Gradient Descent
theta = gradient_descent(X, y, theta, alpha, num_iterations)

# Output the learned parameters
print("Learned Parameters:", theta)
```

This code defines a logistic regression model and uses gradient descent to minimize the cost function, updating the parameters \(\theta\). The dataset `X` is randomly generated, and `y` is created as a simple binary target based on `X`. In a real-world scenario, you would replace `X` and `y` with your actual dataset.

Keep in mind that this implementation is basic and serves educational purposes. For practical applications, especially with large or complex datasets, it's recommended to use optimized libraries like scikit-learn.