Regression Tasks - Logistic Regression

Astrini Sie

asie@seattleu.edu

Week 4

ECEGR4750 - Introduction to Machine Learning
Seattle University

October 10, 2023

Recap and Updates

- Guest Lecture Series: David Boe and Paula Kosasih
- Lab Take Home Assignment due this Thursday before class
- Office Hours
  - T, Th 12-1p at Bannan 224
  - W 7-9p via Zoom
  - F 9-9.45a via Zoom
- Zoom Link: `https://seattleu.zoom.us/j/7519782079?pwd=cnhCM2tPcHJKVWwxZVArS2VHSUNJZz09`
  - Meeting ID: 751 978 2079
  - Passcode: 22498122
- Review Syllabus
- How to read and present a research paper
- Linear Regression: LMS, GD, BGD, SGD, Noise

Overview

1. Supervised Learning

2. Classification
   - Binary Classification
   - Logistic and Likelihood Function
   - Optimization: Maximum Log Likelihood Estimate (MLE)
   - Optimization: Newton's Method
   - Multi-Class Classification

Supervised Learning

Goal of supervised learning

Given training set with known features and labels, produce a prediction function

- During training: given input data ('**training set**') with **features** and **labels**, learn the relationship ('**prediction function**') between them
- During inference: given a brand new data with **features**, use the **prediction function** to predict the **labels**

Supervised Learning

There are two types of supervised learning tasks:

Regression

if Y is continuous

- Estimating the relationships between a dependent variable ('**label**') and one or more independent variables ('**features**').
- Example: X is data of house dimensions and locations, predict Y the price of the house.

Classification

if Y is discrete

- Categorizing a given set of input data into categories ('**classes**') based on one or more variables ('**features**').
- Example: X is an image, predict if Y is a "cat" or a "dog".

Supervised Learning

There are two types of supervised learning tasks:

Regression

if Y is continuous

- Estimating the relationships between a dependent variable ('**label**') and one or more independent variables ('**features**').
- Example: X is data of house dimensions and locations, predict Y the price of the house.

Classification

if Y is discrete

- Categorizing a given set of input data into categories ('**classes**') based on one or more variables ('**features**').
- Example: X is an image, predict if Y is a "cat" or a "dog".

Binary Classification

Definition

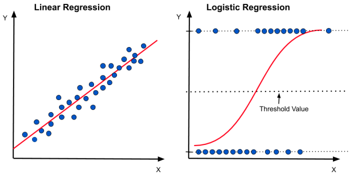
Binary Classification: y can only take on two values, 0 and 1

Given a training set $\{(x^{(i)}, y^{(i)}$ for $i = 1, \ldots, n\}$, let $y^{(i)} \in \{0, 1\}$.

We want to find the prediction $h_\theta(x) \in [0, 1]$.

Binary Classification

Model

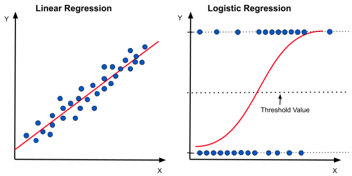


In the case of Linear Regression, h is a **line** function:

$$h_\theta(x^{(i)}) = \sum_{j=0}^{d} \theta_j^{(i)} x_j^{(i)}$$

Binary Classification

Model



In the case of Linear Regression, h is a **line** function:

$$h_\theta(x^{(i)}) = \sum_{j=0}^{d} \theta_j^{(i)} x_j^{(i)}$$

In the case of Logistic Regression, let's pick h as a **sigmoid** function:

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

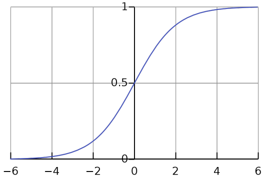
Sigmoid/Logistic Function

The model is a **sigmoid function** or a **logistic function**, which can be re-written as:

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

where

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



As $z \to \infty$, $g(z) \to 1$.
As $z \to -\infty$, $g(z) \to 0$.
$g(z)$, and also $h_\theta(x)$ is always bounded between 0 and 1.

Likelihood Function

How do fit θ in $h_\theta(x)$?

Likelihood Function

How do fit θ in $h_\theta(x)$?

Let's assume:

$$P(y = 1|x; \theta) = h_\theta(x)$$
$$P(y = 0|x; \theta) = 1 - h_\theta(x)$$

Likelihood Function

How do fit θ in $h_\theta(x)$?

Let's assume:

$$P(y = 1|x;\theta) = h_\theta(x)$$
$$P(y = 0|x;\theta) = 1 - h_\theta(x)$$

Which can be compactly written as:

$$p(y|x;\theta) = (h_\theta(x))^y (1 - h_\theta(x))^{1-y}$$

Likelihood Function

Let's define the Likelihood Function, assuming we have n training samples that are independent:

$$L(\theta) = P(y|X;\theta)$$

Likelihood Function

Let's define the Likelihood Function, assuming we have n training samples that are independent:

$$\begin{aligned} L(\theta) &= P(y|X;\theta) \\ &= \prod_{i=1}^{n} p(y^{(i)}|x^{(i)};\theta) \\ &= \prod_{i=1}^{n} \left(h_\theta(x^{(i)})\right)^{y^{(i)}} \left(1 - h_\theta(x^{(i)})\right)^{1-y^{(i)}} \end{aligned}$$

Likelihood Function

Let's define the Likelihood Function, assuming we have n training samples that are independent:

$$\begin{aligned} L(\theta) &= P(y|X;\theta) \\ &= \prod_{i=1}^{n} p(y^{(i)}|x^{(i)};\theta) \\ &= \prod_{i=1}^{n} \left(h_\theta(x^{(i)})\right)^{y^{(i)}} \left(1 - h_\theta(x^{(i)})\right)^{1-y^{(i)}} \end{aligned}$$

Taking a log of the likelihood function gives us:

Log Likelihood Function

$$\ell(\theta) = \log L(\theta) = \sum_{i=1}^{n} y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log (1 - h(x^{(i)}))$$

Maximum Log Likelihood Estimate (MLE)

Similar to the case of Least Mean Squares (LMS) in linear regression, we want to find the value θ that maximizes the Log Likelihood Function. This can be done using gradient descent (or ascent, in case of maximization) technique, updating θ with the following rule:

$$\theta := \theta + \alpha \nabla_\theta \ell(\theta) \tag{1}$$

Let's start with one training sample:

Maximum Log Likelihood Estimate (MLE)

Similar to the case of Least Mean Squares (LMS) in linear regression, we want to find the value θ that maximizes the Log Likelihood Function. This can be done using gradient descent (or ascent, in case of maximization) technique, updating θ with the following rule:

$$\theta := \theta + \alpha \nabla_\theta \ell(\theta) \tag{2}$$

Let's start with one training sample:

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} \ell(\theta) &= \frac{\partial}{\partial \theta_j} \left(y \log h(x) + (1-y) \log(1 - h(x)) \right) \\
&= \frac{\partial}{\partial \theta_j} \Big(y \log g(\theta^T x) + (1-y) \log(1 - g(\theta^T x) \Big) \\
&= \left(y \frac{1}{g(\theta^T x)} + (1-y) \frac{1}{1 - g(\theta^T x)} \right) \frac{\partial}{\partial \theta_j} g(\theta^T x)
\end{aligned}$$

Maximum Log Likelihood Estimate (MLE)

Let's solve for $\frac{\partial}{\partial \theta_j} g(\theta^T x)$ separately:

$$\begin{aligned}
\frac{d}{dz} g(z) &= \frac{d}{dz} \frac{1}{1 + e^{-z}} \\
&= \frac{1}{(1 + e^{-z})^2} (e^{-z}) \\
&= \frac{1}{1 + e^{-z}} \left(1 - \frac{1}{1 + e^{-z}} \right) \\
&= g(z)(1 - g(z))
\end{aligned}$$

Plugging this back in:

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} \ell(\theta) &= \left(y \frac{1}{g(\theta^T x)} + (1 - y) \frac{1}{1 - g(\theta^T x)} \right) g(\theta^T x)(1 - g(\theta^T x)) \frac{\partial}{\partial \theta_j} \theta^T x \\
&= \left(y(1 - g(\theta^T x)) - (1 - y) g(\theta^T x) \right) x_j \\
&= (y - g(\theta^T x)) x_j \\
&= (y - h_\theta(x)) x_j
\end{aligned}$$

Maximum Log Likelihood Estimate (MLE)

Plugging $\frac{\partial}{\partial \theta_j}\ell(\theta) = (y - h_\theta(x))x_j$ back into Equation 2 returns:

Maximum Log Likelihood Estimate (MLE) Learning Rule for one training sample

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}$$

Maximum Log Likelihood Estimate (MLE)

Let's compare the weight update rules for the case of logistic regression and linear regression:

Least Mean Squares (LMS) Learning Rule for one training sample

$$\theta_j := \theta_j - \alpha \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}$$

Maximum Log Likelihood Estimate (MLE) Learning Rule for one training sample

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}$$

Aside from the differences of each $h_\theta(x^{(i)})$, both update rules are identical. Is this a coincidence that the form of both learning rules are the same? Is there a deeper reason behind this?

Newton's Method

There is another algorithm that finds the value θ that maximizes the Log Likelihood Function $\ell(\theta)$.

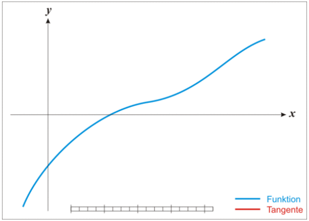
Let's consider Newton's method for finding a zero of a function.

There is another algorithm that finds the value θ that maximizes the Log Likelihood Function $\ell(\theta)$.

Let's consider Newton's method for finding a zero of a function.

Suppose $f : \mathbb{R} \to \mathbb{R}$. Find x such that $f(x) = 0$, where $x \in \mathbb{R}$. Newton Method's performs the following update:

$$x := x - \frac{f(x)}{f'(x)}$$



Newton's Method

There is another algorithm that finds the value θ that maximizes the Log Likelihood Function $\ell(\theta)$.

Let's consider Newton's method for finding a zero of a function.

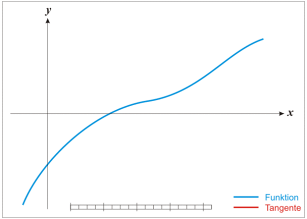
Suppose $f : \mathbb{R} \to \mathbb{R}$. Find x such that $f(x) = 0$, where $x \in \mathbb{R}$. Newton Method's performs the following update:

$$x := x - \frac{f(x)}{f'(x)}$$

1. Starts with an initial guess of x
2. Approximate the function by its tangent line
3. Compute the x intercept of the tangent line
4. Repeat and iterate until we get x in which $f(x) \approx 0$

Newton's Method in Log Likelihood Function

Fisher Scoring



In our case, we have a function, the derivative of the log likelihood function $\ell'(\theta)$, and we want to find the value θ such that $\ell'(\theta) = 0$. Re-writing the Newton's Method for our case:

Newton's Method Learning Rule for one training sample

$$\theta := \theta - \frac{\ell'(\theta)}{\ell''(\theta)}$$

Generalizing Newton's Method to the cases where θ is a vector representation Θ yields:

Newton-Raphson Method

$$\Theta := \Theta - H^{-1}\nabla_{\theta}\ell(\theta)$$

where $H \in R^{(d+1)x(d+1)}$. H is called the **Hessian** matrix:

$$H_{ij} = \frac{\partial^2 \ell(\theta}{\partial\theta_i \partial\theta_j}$$

Newton's Method vs. Gradient Descent

How does Newton's Method compare with Gradient Descent methods (LMS, MLE)?

Newton's Method vs. Gradient Descent

How does Newton's Method compare with Gradient Descent methods (LMS, MLE)?

Pros:

- Newton's Method has a faster convergence than Batch Gradient Descent
- Requires much fewer iterations to get very close to the minima

Cons:

- More expensive per iteration that Batch Gradient Descent, since it requires finding and inverting a $(d+1)x(d+1)$ Hessian. Ok when d is small (< 100).
- Can't be used in modern machine learning cases when d is in the order of billions or trillions.

Multi-Class Classification

References

Chris Re, Andrew Ng, and Tengyu Ma (2023)
CSE229 Machine Learning
Stanford University