Logistic Regression and Classification

Rao Vemuri

UC Davis

Logistic Regression

- This is a regression method when the dependent variable is <u>binary</u>.
  - predict whether a patient will live or die in a particular treatment regime?
  - Tell whether a patient has a disease or not
  - Tell whether a message is spam or ham
  - Tell whether a system will crash in 24 hours

  In these cases, dependent variable can only have two values.

Binary Classification Problem

- A general classification is similar to regression, except the predictions, *y* values, take only a small number of discrete values.
- In a binary classification, y takes on values like 0 or 1, (-1 or +1, Class A or Class B)
- Given $x^{(i)}$, the corresponding *y* value is called class label

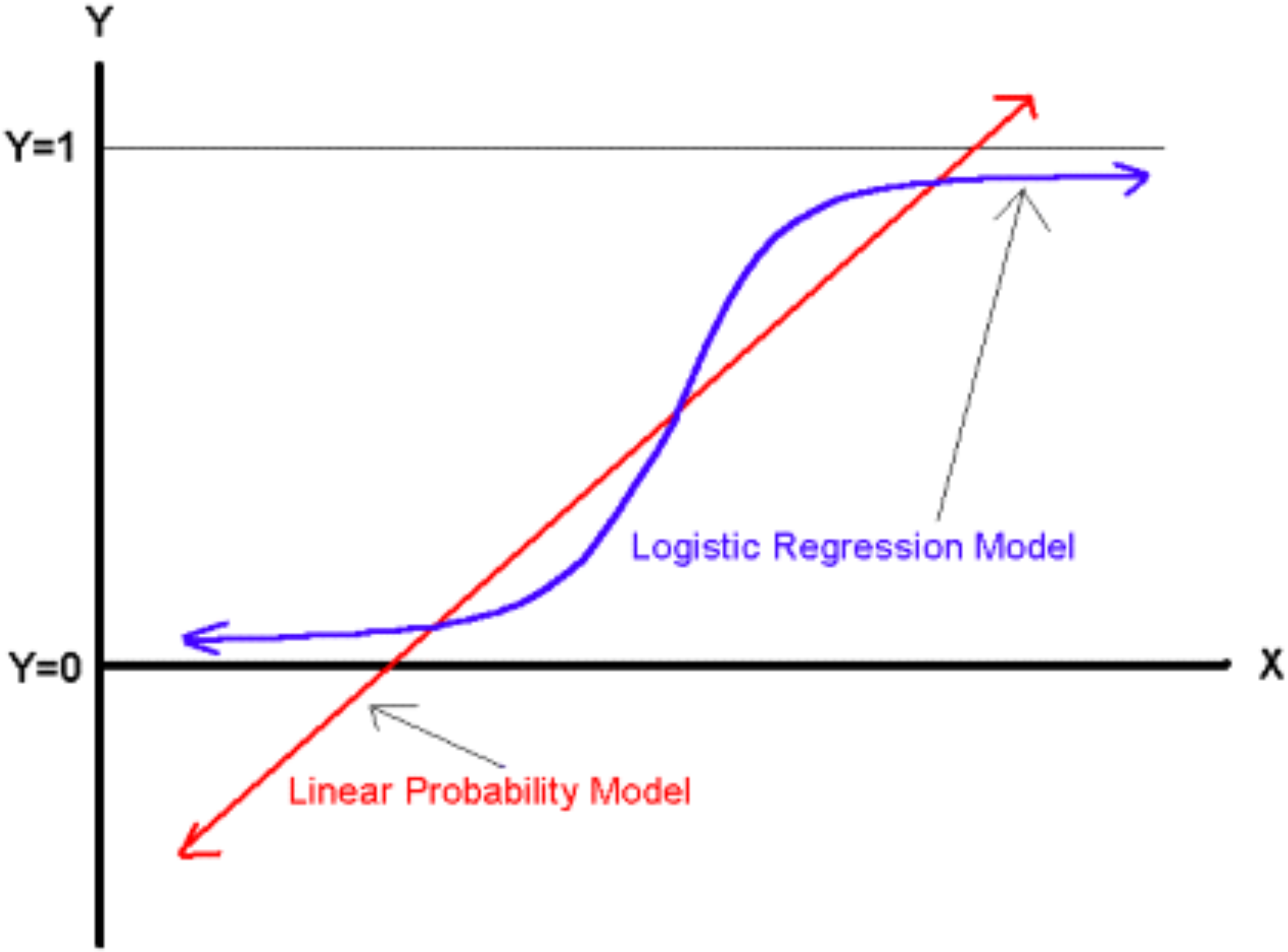
Classification as a Special Case of Regression

- Pretend *y* is continuous as before
- But it doesn't make sense to allow hypotheses that allow *y* to assume *any* value when we know it lies in [0 , 1]
- So allow only the class of functions that look like

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

- This is called logistic function or sigmoid function

Comparing the LP and Logit Models



Binary Classification Problem

- Consider a two-class classification problem with

$$y \in \{0,1\}$$

$$h_\theta(x) \in [0,1]$$

- Although y takes on TWO values we are choosing a richer hypothesis space that allows all intermediate values between 0 and 1
- I will TRY using *h* for any function and *g or σ* for logistic function

Logistic Regression Model

- There is a function in mathematics that assumes only values between 0 and 1. What is that?
- Probability! So assign a probability distribution over the two labels such that the labels further away from the decision boundary are more likely to be correct. That is,

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

where

$$h(z) = g(z) = \frac{1}{1+e^{-z}} = \text{logistic function}$$

– Note that g(-z) = 1- g(z)   (HW: prove this!)

Home Work (Do not submit)

- Show that $g(-z) = 1 - g(z)$
- Show that $g'(z) = g\,(1 - g)$
- Plot (use code) $g(z)$ and $g'(z)$

Probabilistic Interpretation

- We will give a probabilistic interpretation

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

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

- Combining and writing compactly

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

- This trick comes in handy at many places

Enter Likelihood Function

- The quantity $P(y|x; \theta)$ is called the likelihood function
- Given a model with parameters θ, this function tells us the chances of observing y as output for a specified input x.
- We write this as $L(\theta) = P(y|x; \theta)$
- Our goal is to pick θ to maximize this likelihood

Likelihood Function

- If *m* training examples {x, y} are generated independently, we can write the likelihood function as ......(now, x and y are vectors)

$$L(\theta) = p(y|x;\theta) = p(y^{(1)}, y^{(2)}, \ldots y^{(m)} | \mathrm{x}, \theta)$$

$$= \prod_{i=1}^{m} p(y^{(i)} \mid x^{(i)}; \theta)$$

$$= \prod_{i=1}^{m} \left(h_\theta(x^{(i)})\right)^{y^{(i)}} \bullet \left(1 - h_\theta(x^{(i)})\right)^{1-y^{(i)}}$$

Maximizing Log Likelihood

- It is easier to work with the log of the likelihood:

$$l(\theta) = \log L(\theta)$$

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

- No damage done because both ($L(\theta) and\ l(\theta)$ attain their maximum at the same place
- The equation on the right is also called the cross-entropy function

Gradient Ascent

- To maximize this, we cannot use direct methods; we use the gradient ascent:

$$\theta \leftarrow \theta + \eta \nabla_{\theta} l(\theta)$$

- Let us use the stochastic gradient version by using one training example at a time

Aside: Can also use Gradient Descent

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

Here we define J as the negative of the log likelihood and minimize it

Verify at home!

- An outline of the derivation for the stochastic (incremental case) is shown in the next slide for the case when h = sigmoid. While going from step 1 to step 2 we used the fact that the derivative h’ = h (1 – h) for sigmoid
- Do the indicated calculations and convince yourself that there are no errors
- HW: Repeat the case when h = hyperbolic tangent (do not submit)

Calculations for Gradient Method

- Using one training pair (x, y), differentiate *l*(θ):

$$\frac{\partial}{\partial\theta_j} l(\theta) = \left(y\frac{1}{h(\theta^T x)} - (1-y)\frac{1}{(1-h(\theta^T x))} \right) \frac{\partial}{\partial\theta_j} h(\theta^T x)$$

$$= \left(y\frac{1}{h(\theta^T x)} - (1-y)\frac{1}{(1-h(\theta^T x))} \right) h(\theta^T x)(1-h(\theta^T x)) \frac{\partial}{\partial\theta_j} (\theta^T x)$$

$$= \left(y(1-h(\theta^T x) - (1-y)h(\theta^T x) \right) x_j$$

$$= \left(y - h(\theta^T x) \right) x_j = error * input\ j$$

Important to Note

- If we use the right error function, (like the cross entropy we used here) something nice happens:
- The gradient of the logistic and the gradient of the error function cancel each other, as it happened from step 2 to step 3. This is NOT the case when we used LMS criterion

Weight Update Rule

- The weight update rule for the stochastic gradient method using the logistic nonlinearity is:

$$\theta_j \leftarrow \theta_j + \eta\,(y^{(i)} - h_\theta(x^{(i)}))\,x_j^i$$

- Compare with LMS rule! Both equations “look” alike, but they are NOT the same.
- Here, h(θ) is the non-linear, sigmoid function
- New θ = old θ + eta * (error) * input

Coincidence?

- It's surprising that we end up with the same update rule for a rather different algorithm and learning problem.
- Is this coincidence, or is there a deeper reason behind this?
- We'll answer this when get get to GLM models.

Multi-Class Classification

- We can extend this to multi-class classification.
- In that case, we use a generalization of the cross-entropy function, known as the "softmax" error function
- We will deal with this issue separately

Special Case: Perceptron

- The famous neural net, Perceptron is a special case of this. The non-linear function for a classical Perceptron is the Signum function:

$$h_\theta(z) = g(\theta^T x) = \mathrm{sgn}(\theta^T x) = \begin{cases} +1, z \geq 0 \\ -1, z < 0 \end{cases}$$

$$\theta_j \leftarrow \theta_j + \eta\,(y^{(i)} - h_\theta(x^{(i)}))\,x_j^i$$

Comment on Perceptron

- Even though the perceptron may be cosmetically similar to the other algorithms we talked about, it is actually a very different type of algorithm than logistic regression and least squares linear regression
- In particular, it is difficult to endow the perceptron's predictions with meaningful probabilistic interpretations, or derive the perceptron as a maximum likelihood estimation algorithm.