Wednesday, February 24, 2021

Logistic Regression

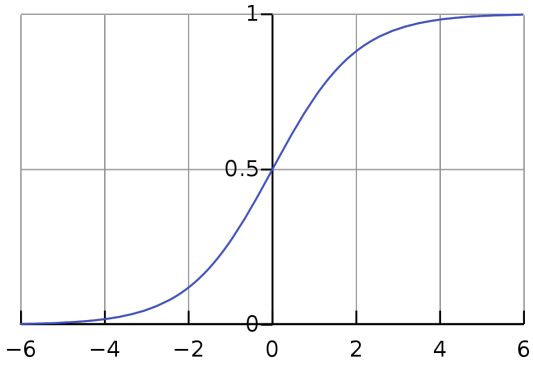
Logistic regression is a classification algorithm despite its name has regression in it. It predicts two binary dependent output either as 0 or 1 based on the input variables.

The regression in its name means we are using the same algorithm as used in linear regression. The difference is that the output is mapped using a logistic/sigmoid function so that it will be in a range between 0 and 1. Any output value < 0.5 will be classified as 0 and any value >= 0.5 will be classified as 1.

Hypothesis:

- Use the linear regression hypothesis, but need to transform the arbitrary valued function into a function that is better for classification
- Apply sigmoid/logistic function to the linear regression hypothesis, thus the mapped output is between at 0 and 1.

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



Sigmoid function

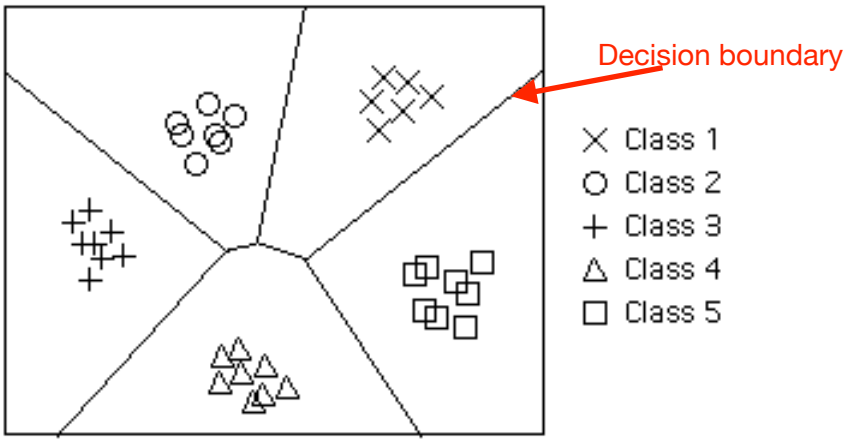
https://en.wikipedia.org/wiki/Sigmoid_function#/media/File:Logistic-curve.svg

- The output is between (0, 1), which is the estimated probability that the output is 1.
  - For example, if h = 0.7, then the probability of output is 1 is 70%, while for output being 0 is 30%.

Decision boundary:

The surfaces that separate the decision regions (each region represent a class/ category). The decision boundary represent points where there are ties between two or more categories. Where the hypothesis is h = 0.5.

- Linear decision boundary
- Non-linear decision boundary



https://www.cs.princeton.edu/courses/archive/fall08/cos436/Duda/PR_simp/bndrys.htm

Cost function: logistic loss

- We can't use the same cost function as used in linear regression since it is no longer a convex function with the proposed hypothesis above.
- Instead it has a lot of local minima, which makes the optimization process difficult to find the global minima.

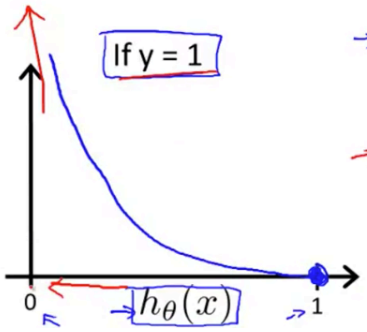
$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \text{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

$$\text{Cost}(h_\theta(x), y) = -\log(h_\theta(x)) \qquad \text{if } y = 1$$
$$\text{Cost}(h_\theta(x), y) = -\log(1 - h_\theta(x)) \qquad \text{if } y = 0$$

- With this cost function: miss classifying will result in larger penalty
  - When y = 1, if h =1, cost = 0; if h=0, cost $\rightarrow\infty$
  - When y = 0, if h =0, cost = 0; if h=1, cost $\rightarrow\infty$

Logistic regression cost function

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$



$\text{Cost} = 0$ if $y = 1, h_\theta(x) = 1$
But as $h_\theta(x) \to 0$
$Cost \to \infty$

By Andrew

Captures intuition that if $h_\theta(x) = 0$, (predict $P(y = 1|x; \theta) = 0$), but $y = 1$, we'll penalize learning algorithm by a very large cost.

By Andrew Ng

Simplified Cost function

This cost function is derived using the principle of maximum likelihood. *(I think I have seen this before, review this later)*

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

Gradient descent (same process as in linear regression)

It turns out the weight updating equation is in the same form as linear regression. Note the hypothesis h is different now.

Gradient Descent

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

Want $\min_\theta J(\theta)$:

$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}$

Repeat {

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

(simultaneously update all θ_j)

}

$h_\theta(x) = \theta^T x$

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

Algorithm looks identical to linear regression!

By Andrew Ng

Optimizing algorithms

- Conjugate gradient, BFGS, L-BFGS
- Advantages:
  - No need to pick up the learning rate
  - Faster
- Disadvantages: more complex

Example:

$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$

$J(\theta) = (\theta_1 - 5)^2 + (\theta_2 - 5)^2$

$\frac{\partial}{\partial \theta_1} J(\theta) = 2(\theta_1 - 5)$

$\frac{\partial}{\partial \theta_2} J(\theta) = 2(\theta_2 - 5)$

```
function [jVal, gradient]
            = costFunction(theta)
  jVal = (theta(1)-5)^2 + ...
         (theta(2)-5)^2;
  gradient = zeros(2,1);
  gradient(1) = 2*(theta(1)-5);
  gradient(2) = 2*(theta(2)-5);
```

```
options = optimset('GradObj', 'on', 'MaxIter', '100');
initialTheta = zeros(2,1);
[optTheta, functionVal, exitFlag] ...
    = fminunc(@costFunction, initialTheta, options);
```

$\texttt{theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$

```
function [jVal, gradient] = costFunction(theta)
    jVal = [code to compute J(theta)];
    gradient(1) = [code to compute ∂/∂θ0 J(θ)];
    gradient(2) = [code to compute ∂/∂θ1 J(θ)];
    ⋮
    gradient(n+1) = [code to compute ∂/∂θn J(θ)];
```

Example in Octave by Andrew Ng