Classification

To attempt classification, one method is to use linear regression and map all predictions greater than 0.5 as a 1 and all less than 0.5 as a 0. However, this method doesn't work well because classification is not actually a linear function.

The classification problem is just like the regression problem, except that the values we now want to predict take on only a small number of discrete values. For now, we will focus on the **binary classification problem** in which y can take on only two values, 0 and 1. (Most of what we say here will also generalize to the multiple-class case.) For instance, if we are trying to build a spam classifier for email, then x^{(i)}$x^{(i)}$ may be some features of a piece of email, and y may be 1 if it is a piece of spam mail, and 0 otherwise. Hence, y∈{0,1}. 0 is also called the negative class, and 1 the positive class, and they are sometimes also denoted by the symbols “-” and “+.” Given x^{(i)}$x^{(i)}$, the corresponding y^{(i)}$y^{(i)}$ is also called the label for the training example.

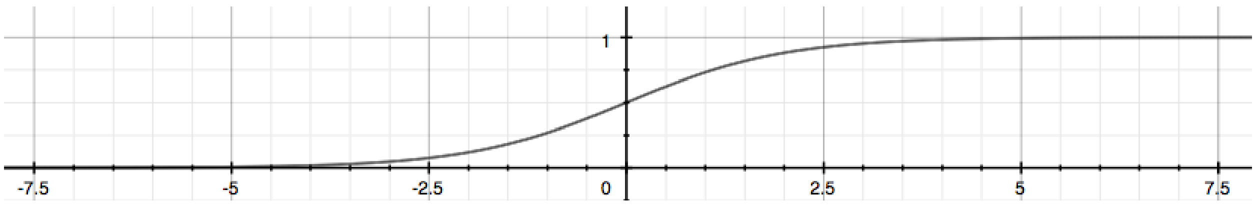
Hypothesis Representation

We could approach the classification problem ignoring the fact that y is discrete-valued, and use our old linear regression algorithm to try to predict y given x. However, it is easy to construct examples where this method performs very poorly. Intuitively, it also doesn’t make sense for h_\theta (x)$h_\theta(x)$ to take values larger than 1 or smaller than 0 when we know that y ∈ {0, 1}. To fix this, let’s change the form for our hypotheses h_\theta (x)$h_\theta(x)$ to satisfy 0 \leq h_\theta (x) \leq 1$0 \leq h_\theta(x) \leq 1$. This is accomplished by plugging \theta^Tx$\theta^T x$ into the Logistic Function.

Our new form uses the "Sigmoid Function," also called the "Logistic Function":

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

The following image shows us what the sigmoid function looks like:



The function g(z), shown here, maps any real number to the (0, 1) interval, making it useful for transforming an arbitrary-valued function into a function better suited for classification.

h_\theta(x)$h_\theta(x)$ will give us the **probability** that our output is 1. For example, h_\theta(x)=0.7$h_\theta(x)=0.7$ gives us a probability of 70% that our output is 1. Our probability that our prediction is 0 is just the complement of our probability that it is 1 (e.g. if probability that it is 1 is 70%, then the probability that it is 0 is 30%).

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

Decision Boundary

Decision Boundary

In order to get our discrete 0 or 1 classification, we can translate the output of the hypothesis function as follows:

$$h_\theta(x)\geq 0.5 \rightarrow y=1 \quad h_\theta(x)<0.5 \rightarrow y=0$$

The way our logistic function g behaves is that when its input is greater than or equal to zero, its output is greater than or equal to 0.5:

$$g(z)\geq 0.5 \, when \, z\geq 0$$

Remember.

$$z=0, e^{0}=1 \Rightarrow g(z)=1/2 \quad z\rightarrow\infty, e^{-\infty}\rightarrow 0 \Rightarrow g(z)=1 \quad z\rightarrow-\infty, e^{\infty}\rightarrow\infty \Rightarrow g(z)=0$$

So if our input to g is \theta^T X$\theta^T X$, then that means:

$$h_\theta(x)=g(\theta^T x)\geq 0.5 \, when \, \theta^T x\geq 0$$

From these statements we can now say:

$$\theta^T x\geq 0 \Rightarrow y=1 \quad \theta^T x<0 \Rightarrow y=0$$

The **decision boundary** is the line that separates the area where y = 0 and where y = 1. It is created by our hypothesis function.

Example:

$$\theta=\begin{bmatrix}5\\-1\\0\end{bmatrix} \quad y=1 \, if \, 5+(-1)x_1+0x_2\geq 0 \quad 5-x_1\geq 0 \quad -x_1\geq -5 \quad x_1\leq 5$$

In this case, our decision boundary is a straight vertical line placed on the graph where x_1 = 5$x_1=5$, and everything to the left of that denotes y = 1, while everything to the right denotes y = 0.

Again, the input to the sigmoid function g(z) (e.g. \theta^T X$\theta^T X$) doesn't need to be linear, and could be a function that describes a circle (e.g. z = \theta_0 + \theta_1 x_1^2 +\theta_2 x_2^2$z=\theta_0+\theta_1 x_1^2+\theta_2 x_2^2$) or any shape to fit our data.

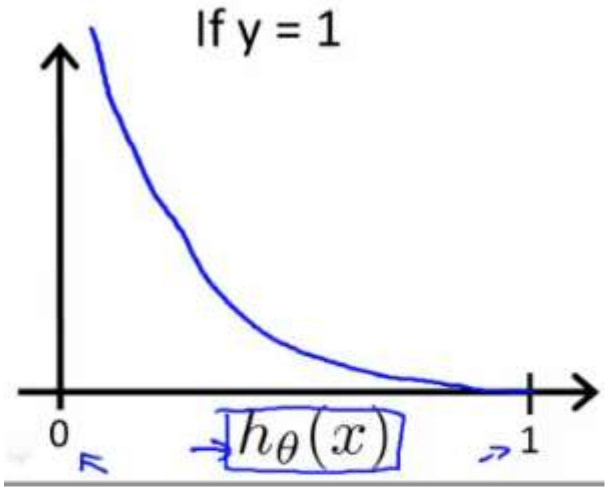
Cost Function

We cannot use the same cost function that we use for linear regression because the Logistic Function will cause the output to be wavy, causing many local optima. In other words, it will not be a convex function.

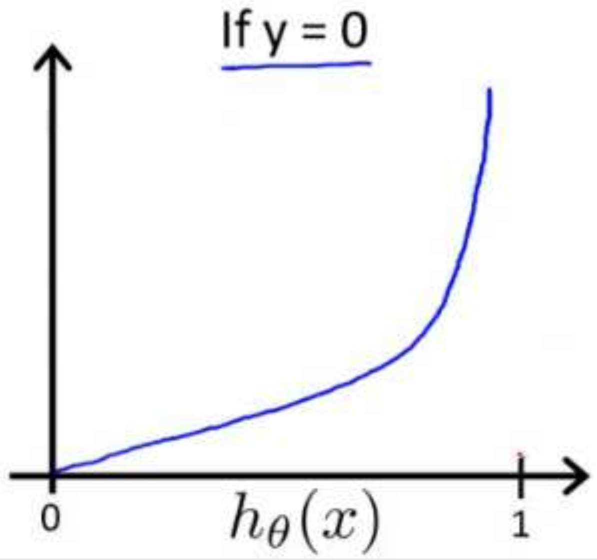
Instead, our cost function for logistic regression looks like:

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

When y = 1, we get the following plot for J(\theta)$J(\theta)$ vs h_\theta (x)$h_\theta(x)$:



Similarly, when y = 0, we get the following plot for J(\theta)$J(\theta)$ vs h_\theta (x)$h_\theta(x)$:



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

If our correct answer 'y' is 0, then the cost function will be 0 if our hypothesis function also outputs 0. If our hypothesis approaches 1, then the cost function will approach infinity.

If our correct answer 'y' is 1, then the cost function will be 0 if our hypothesis function outputs 1. If our hypothesis approaches 0, then the cost function will approach infinity.

Note that writing the cost function in this way guarantees that J(θ) is convex for logistic regression.

Simplified Cost Function and Gradient Descent

Simplified Cost Function and Gradient Descent

Note: [6:53 - the gradient descent equation should have a 1/m factor]

We can compress our cost function's two conditional cases into one case:

\mathrm{Cost}(h_\theta(x),y) = - y \; \log(h_\theta(x)) - (1 - y) \log(1 - h_\theta(x))$\mathrm{Cost}(h_\theta(x),y)=-y\log(h_\theta(x))-(1-y)\log(1-h_\theta(x))$

Notice that when y is equal to 1, then the second term (1-y)\log(1-h_\theta(x))$(1-y)\log(1-h_\theta(x))$ will be zero and will not affect the result. If y is equal to 0, then the first term -y \log(h_\theta(x))$-y\log(h_\theta(x))$ will be zero and will not affect the result.

We can fully write out our entire cost function as follows:

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

A vectorized implementation is:

$$h=g(X\theta) \quad J(\theta)=\frac{1}{m}\cdot\left(-y^T\log(h)-(1-y)^T\log(1-h)\right)$$

Gradient Descent

Remember that the general form of gradient descent is:

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

We can work out the derivative part using calculus to get:

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

Notice that this algorithm is identical to the one we used in linear regression. We still have to simultaneously update all values in theta.

A vectorized implementation is:

\theta := \theta - \frac{\alpha}{m} X^{T} (g(X \theta) - \vec{y}) $\theta:=\theta-\frac{\alpha}{m}X^T(g(X\theta)-\vec{y})$

Advanced Optimization

Note: [7:35 - '100' should be 100 instead. The value provided should be an integer and not a character string.]

"Conjugate gradient", "BFGS", and "L-BFGS" are more sophisticated, faster ways to optimize θ that can be used instead of gradient descent. We suggest that you should not write these more sophisticated algorithms yourself (unless you are an expert in numerical computing) but use the libraries instead, as they're already tested and highly optimized. Octave provides them.

We first need to provide a function that evaluates the following two functions for a given input value θ:

$$J(\theta) \quad \frac{\partial}{\partial\theta_j}J(\theta)$$

ulticlass Classification: One-vs-all

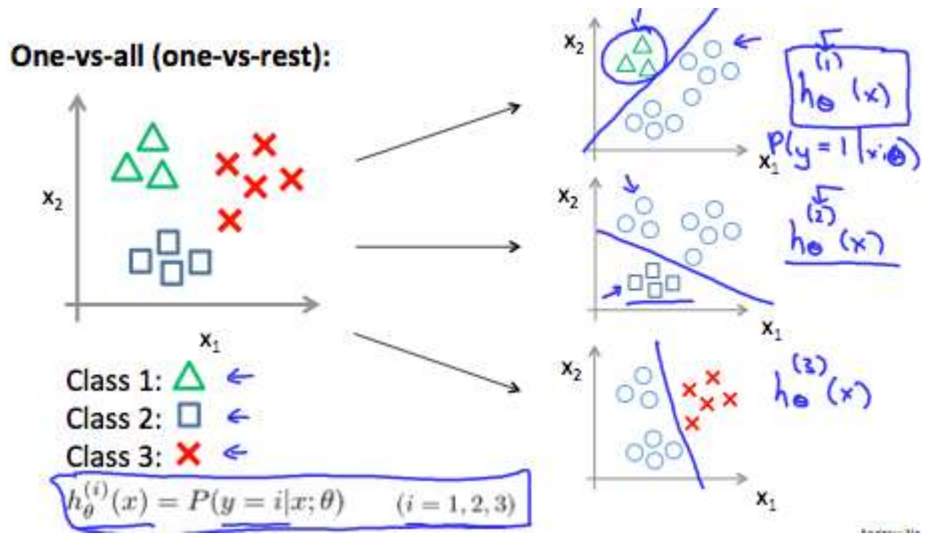
Now we will approach the classification of data when we have more than two categories. Instead of y = {0,1} we will expand our definition so that y = {0,1...n}.

Since y = {0,1...n}, we divide our problem into n+1 (+1 because the index starts at 0) binary classification problems; in each one, we predict the probability that 'y' is a member of one of our classes.

$$y \in \{0,1...n\} h_\theta^{(0)}(x) = P(y=0|x;\theta) h_\theta^{(1)}(x) = P(y=1|x;\theta) \cdots h_\theta^{(n)}(x) = P(y=n|x;\theta) \mathrm{prediction} = \max_i(h_\theta^{(i)}(x$$

We are basically choosing one class and then lumping all the others into a single second class. We do this repeatedly, applying binary logistic regression to each case, and then use the hypothesis that returned the highest value as our prediction.

The following image shows how one could classify 3 classes:



To summarize:

Train a logistic regression classifier h_\theta(x)$h_\theta(x)$ for each class to predict the probability that y = i.

To make a prediction on a new x, pick the class that maximizes h_\theta (x) $h_\theta(x)$