Machine Learning Class Notes

Hugo Fragata, hugofragata@ua.pt

1 Introduction

Apontamentos da Cadeira de Opção de 4º Ano, Aprendizagem Automática, do curso de Mestrado Integrado em Engenharia de Computadores e Telemática, dado pela professora Petia Georgieva (petia@ua.pt)

Notes on Machine Learning class taught by Petia Georgieva (petia@ua.pt) at University of Aveiro in the degree Integrated Masters in Computer and Telematics Engineering.

2 Linear Regression

Defining the differences between *Regression* and *Classification* is very important in order to figure out which tool do we want to pick up from our toolbox to solve a specific problem. In *Classification* the desired output is discrete, non-continuous, a label. So it can be, for example, an integer, a boolean, and so forth. On the other hand *Regression* the output is continuous, such as a real number.

2.1 Univariate linear regression

Problem: Given the living area and the respective price of several houses how can we learn to predict the prices of other houses, as a function of their living areas?

Living Area (feet^2)	Price (1000\$s)
2014	400
1600	300
2400	369
...	...

A linear model can be defined for such a problem as such:

$$h_\theta(x) := \theta^T x = \theta_0 + \theta_1 x_1$$

from

$$\theta_0 x_0 + \theta_1 x_1 \leftrightarrow \begin{bmatrix} x_0 & x_1 \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$

Let m be the number of examples and n the number of features.
We can then define

$$x_1 = \begin{bmatrix} x_1^{(1)} \\ \\ x_1^{(2)} \\ \vdots \\ x_1^{(m)} \end{bmatrix}$$

and

$$y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

The *Cost* in machine learning usually means a function that gives us, in layman's terms, how wrong is our learned model. More precisely, it tells us the difference between our model's predictions and the actual reality. In Univariate Linear Regression a *"good" Cost*, J(θ),function to use is as such:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i)^2$$

We, to get the best parameters for our model, want to get the θ that produces the minimum value of J, or:

$$\min_\theta J(\theta)$$

To understand the *Gradient Descent* algorithm one must understand the quadratic nature of the cost function defined above.

therefore, assuming $\alpha > 0$ is the size of each step towards $J(\theta)$ minimum, we define the next model parameters to be:

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

Figure 1: Visual explanation of gradient descent.

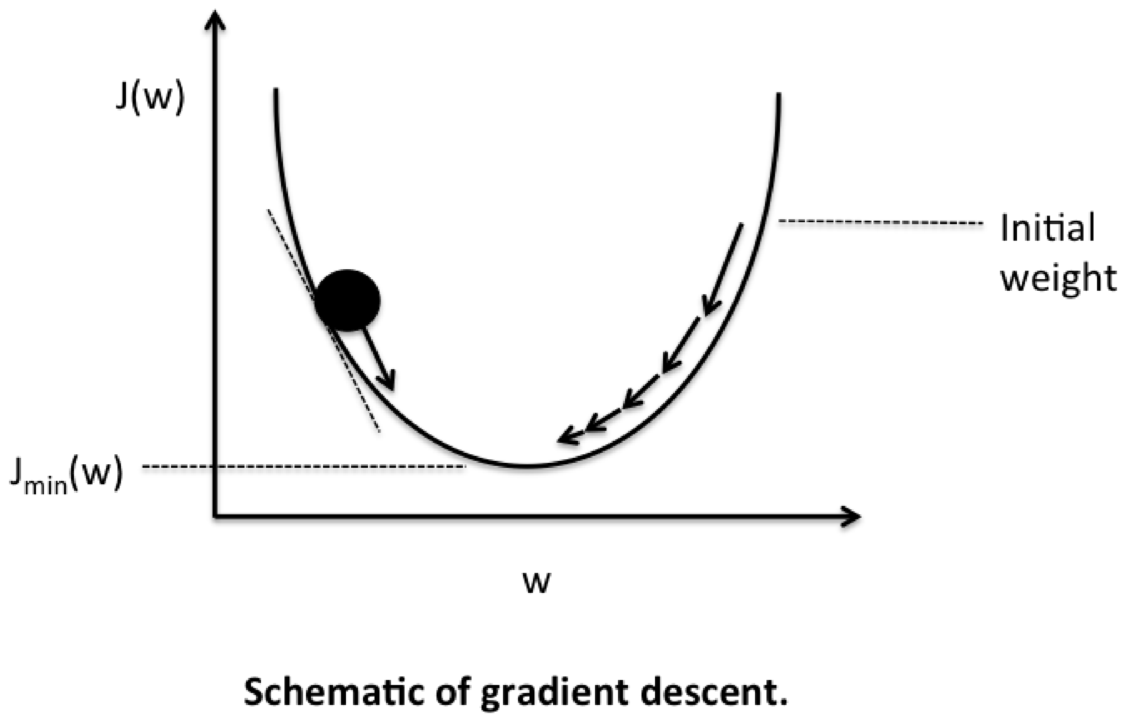


Figure 2: Visual explanation of gradient descent cost function convergence for different learning .

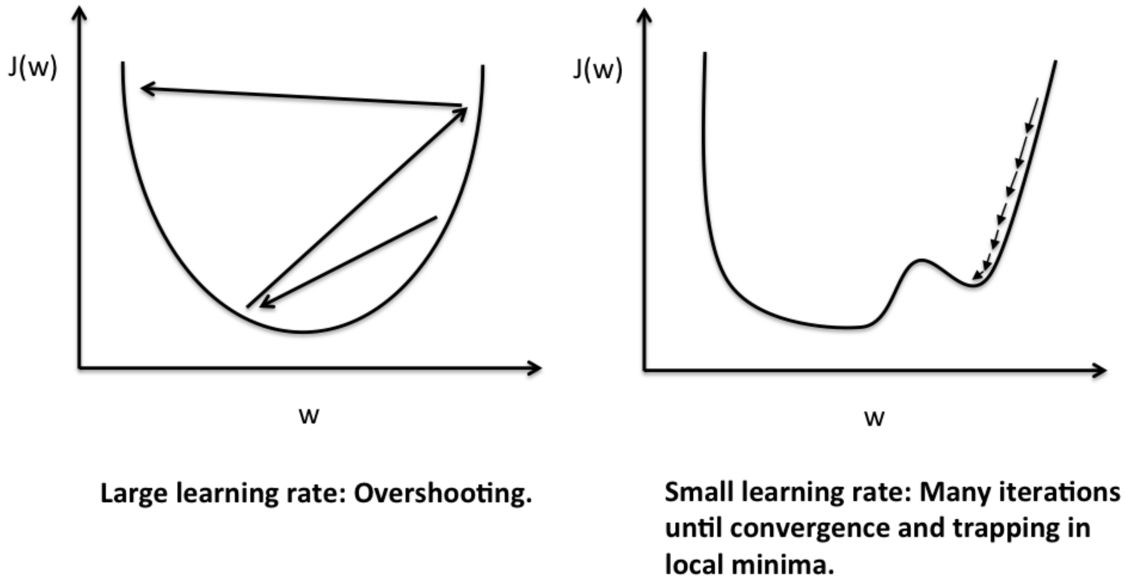
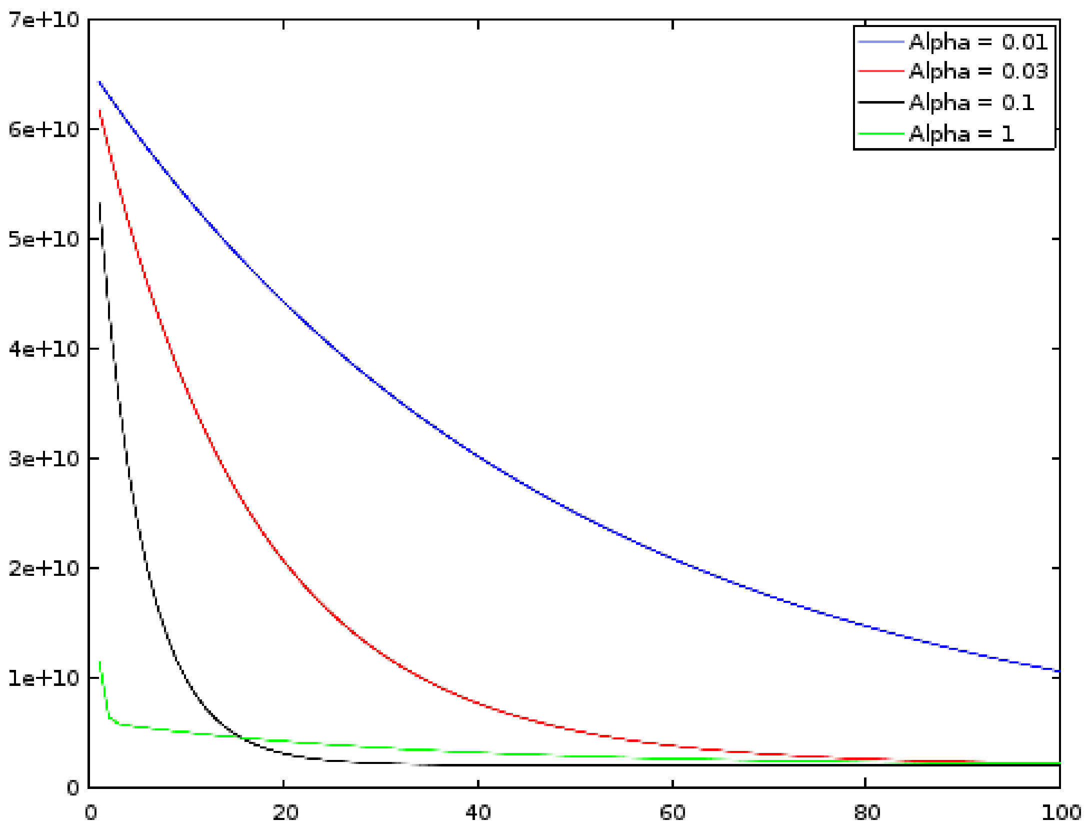

J(w)
w
Large learning rate: Overshooting.
J(w)
w
Small learning rate: Many iterations until convergence and trapping in local minima.

Figure 3: Graphing the cost function convergence for different α learning rates. Too small of a learning rate takes much more iterations to converge.



How do we compute the gradient of the cost function $\frac{\partial J(\theta)}{\partial \theta_j}$? Let's start by assuming the number of examples, m, is just 1.

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{\partial}{\partial \theta_j}\frac{1}{2}(h_\theta(x) - y)^2$$

$$= 2\frac{1}{2}(h_\theta(x) - y)\frac{\partial}{\partial \theta_j}(h_\theta(x) - y)$$

$$= (h_\theta(x) - y)\frac{\partial}{\partial \theta_j}(\sum_{i=0}^{n}\theta_i x_i - y)$$

$$= (h_\theta(x) - y)x_j$$

We then can easily see, that for $m > 1$, we get a gradient as:

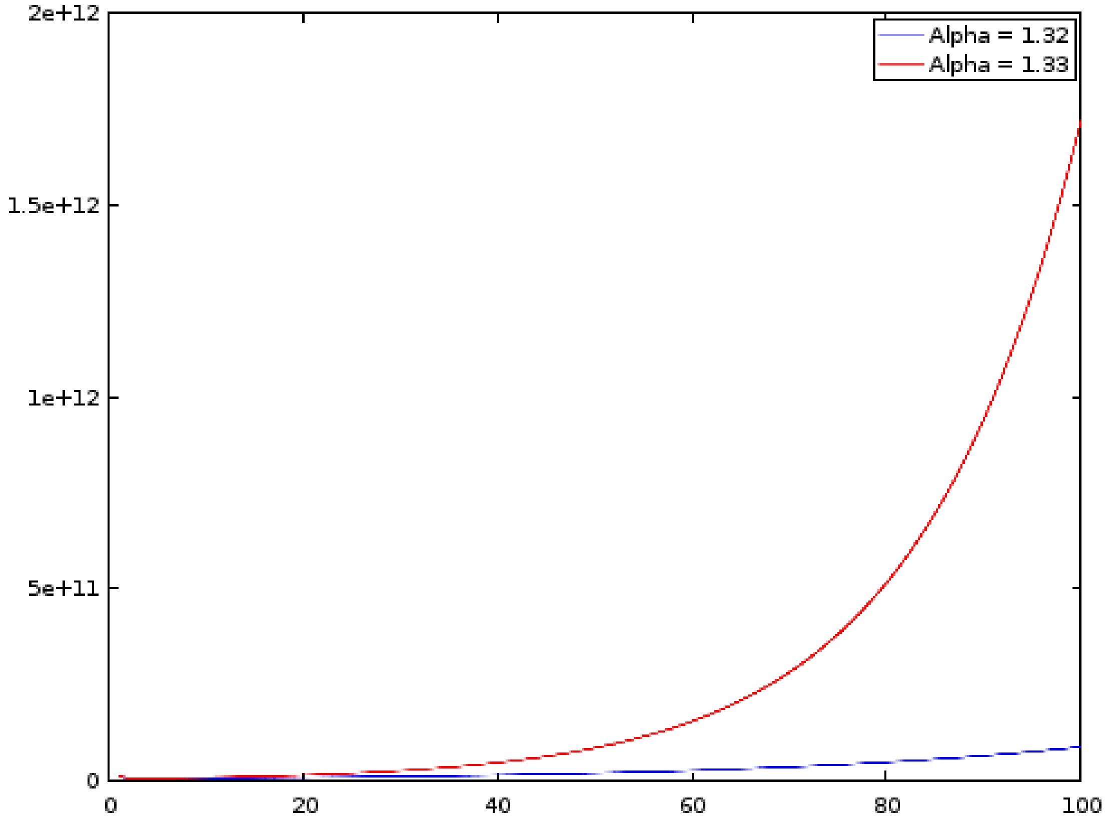
$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^i) - y^i)x_j^i$$

Figures 1 and 2 were taken from *https://sebastianraschka.com.*

2.2 Multivariate Linear Regression

Let's suppose that instead of having only one feature in the data, as we have in Univariate Linear Regression, we have several Multivariate ($m > 1$).

Figure 4: Graphing the cost function convergence for different α learning rates. Too big of a learning rate results in overshooting and non-convergence.



Living Area (feet^2)	#bedrooms	Price (1000$s)
2014	2	400
1600	3	300
2400	1	369
$\vdots$	$\vdots$	$\vdots$

Having more than 1 feature we need to redefine our model as such:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

Let's redefine X, y, theta, and error, all as vectors. Note that X is a vector of examples with each example being a vector of features instances.

$$X = \begin{bmatrix} (x^1)^T \\ (x^2)^T \\ \vdots \\ (x^m)^T \end{bmatrix}$$

,

$$y = \begin{bmatrix} y^1 \\ y^2 \\ \vdots \\ y^m \end{bmatrix}$$

,

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

,

$$X\theta - y = \begin{bmatrix} (x^1)^T\theta \\ (x^2)^T\theta \\ \vdots \\ (x^m)^T\theta \end{bmatrix} - \begin{bmatrix} y^1 \\ y^2 \\ \vdots \\ y^m \end{bmatrix} = \begin{bmatrix} h_\theta(x^1) - y^1 \\ h_\theta(x^2) - y^2 \\ \vdots \\ h_\theta(x^m) - y^m \end{bmatrix}$$

So, assuming we have the same Cost function $J(\theta)$ as we did in Univariate, we can then define a vector of gradients of the cost function:

$$\begin{bmatrix} \frac{\partial J}{\partial \theta_0} \\ \frac{\partial J}{\partial \theta_1} \\ \vdots \\ \frac{\partial J}{\partial \theta_n} \end{bmatrix} = \frac{1}{m} \begin{bmatrix} \sum_{i=1}^m((h_\theta(x^i) - y^i)x_0^i) \\ \sum_{i=1}^m((h_\theta(x^i) - y^i)x_1^i) \\ \vdots \\ \sum_{i=1}^m((h_\theta(x^i) - y^i)x_n^i) \end{bmatrix}$$

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

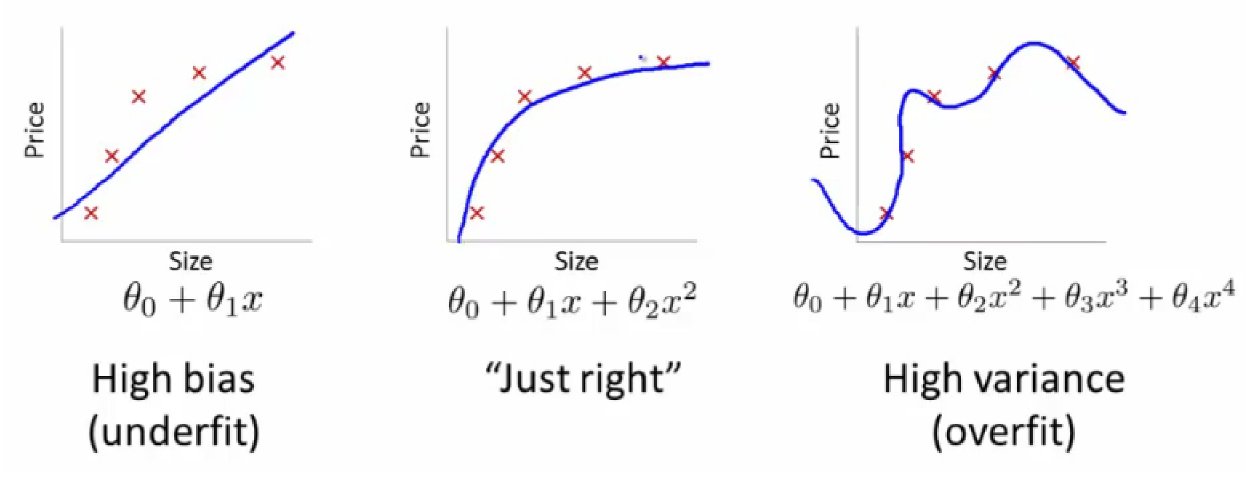
The θ is updated at each learning iteration by the same gradient descent algorithm, but with a different cost gradient:

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

and, of course, we get:

$$\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^i) - y^i)x_0^i$$

Figure 5: Comparison of degrees of complexity of fit between a number of features.



$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i) x_1^i$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i) x_2^i$$

2.3 Transforming a Univariate Regression into Multivariate

A very useful method to get a better suited model to fit the data when it only has one feature is to create *artificial* features by powering it, basically turning a model that is a line (*first order polynomial*), to a parabola (*second order polynomial*) and so forth. Treating the powers of x as features on their own right, we see that we actually created a multivariate model. Formalizing such models would be:

$$h_\theta(x) = \theta_0 + x\theta_1$$

first order polynomial

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x^2$$

second order polynomial

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x^2 + \cdots + \theta_{16} x^{16}$$

16th order polynomial

2.4 Over-Fitting Problem & Regularized Linear Regression

Let's say we have a problem as described in 2.1 and we increase the number of features by using the method explained in 2.3, let's graph the data and the different models fit in Figure 5.

Although the *fourth order polynomial* fits quite nicely on the given data, it doesn't generalize very well on new data. This problem is called *Over-fitting.*

There are two solutions for this problem:

1. Reducing the number of features
2. Data regularization

We'll look into the Regularization solution. Basically we modify the cost function in order to prevent overfit. We do this by adding a term to $J(\theta)$ called the *Regularization term* $\lambda \sum_{j=1}^{n} \theta_j^2$, in contrast to the already studied *bias term.* The *Regularization term* is the average of the model's parameters times a coefficient λ. Therefore the regularized cost function looks like this:

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^i)^2 + \lambda\sum_{j=1}^{n}\theta_j^2$$

Now, we have to compute the gradient of this regularized cost function. Since we only added one term to the cost function and the already computed the unregularized cost function we need only to compute the gradient of the regularization term and then add it to the other gradient. So,

$$\frac{\partial}{\partial\theta_j}\lambda\sum_{j=1}^{n}\theta_j^2 = \frac{\lambda}{m}\theta_j$$

therefore, the complete gradient of the regularized cost function:

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

3 Logistic Regression

Unlike *Linear Regression, Logistic Regression* is a *Classification* tool. In *Classification* the desired output is discrete, non-continuous, a label. For example: Is it spam (yes/no), Is it a tumor (yes/no), and so forth. There are different way to classify an example of data: Binary classification, Threshold classification and Multiclass Classification.

Binary classification is when our model outputs a binary value, 0/1 True/False and the like. Threshold classification is when the model outputs a real number within a certain range, let's say in the interval $[0, 1]$, and then if it is above a certain *threshold*, then it outputs 1 and if its less outputs a 0. In Multiclass classification we have a limited set of possible outputs, for example $\{0, 1, 2, 3\}$. This will be addressed in another chapter. We'll look into a problem defined as such: Training set:

$$\{(x^1, y^1), (x^2, y^2), \ldots, (x^m, y^m)\}, y \in \{0, 1\}$$

For m examples, each one has n features:

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}$$

3.1 Sigmoid Hypothesis

The sigmoid function, $s(z)$, is a function where $z \in \mathbb{R}$, $y = \{0, 1\}$. Can be graphed as 6 and defined as such:

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

Our model $h_\theta(x)$ is then defined using the sigmoid function, because we need outputs in such a range:

$$\theta^T x = \theta_0 + \sum_{j=1}^{n}\theta_j x_j$$

$$h_\theta(x) := s(\theta_T x)$$

3.2 Logistic Regression Cost Function

Why can't we use the same cost function $J(\theta)$ that we used in linear regression? Because, since the model $h_\theta(x)$ is *sigmoidal*, the cost is highly convex, with a many local minims, which makes it hard to converge on the global minimum.

Therefore, a new cost function is proposed for Logistic Regression:

Figure 6: Graphing $s(z)$

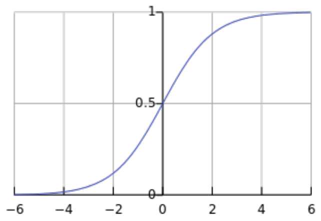
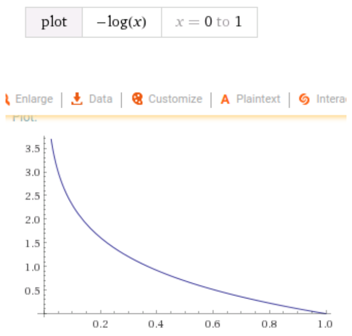


Figure 7: $-log(h_\theta(x))$



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

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

which can be written as a single function as:

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

But why did we pick such a $Cost(h_\theta(x), y)$ function? Let's look into what happens for the case where $y = 1$. Because $y = 1 \Rightarrow Cost = -log(h_\theta(x))$

$$h_\theta(x) = 1 \rightarrow log(1) = 0$$

$$h_\theta(x) = 0 \rightarrow log(0) = -\infty$$

So when $y = 1$, $Cost(h_\theta(x)) = 1 \Rightarrow -0$ and $Cost(h_\theta(x)) = 0 \Rightarrow +\infty$. The inverse will happen when $y = 0$, if the model predicts a 1 then cost tends to $+\infty$ and when it predicts a 0 cost tends to 0.

Even though the cost function had to be adjusted from the Linear Regression to achieve results in Logistic Regression the gradient descent algorithm stays the same. We have the same cost function gradient and the same gradient descent parameter θ_j update.

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

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

Figure 8: $-log(1 - h_\theta(x))$

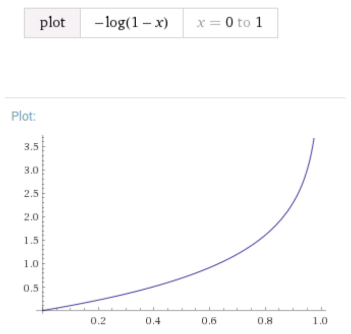
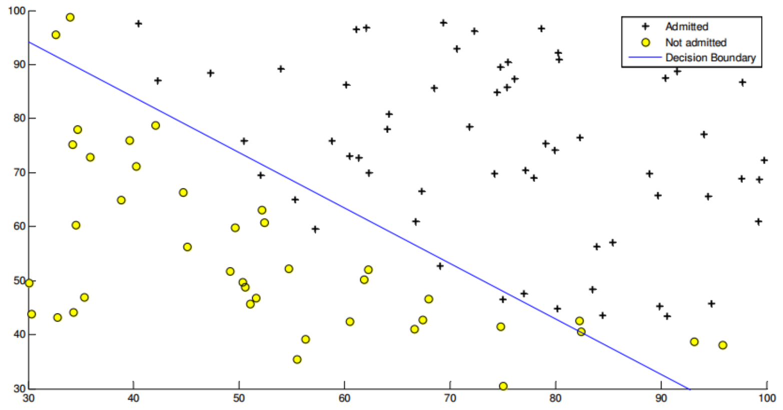


Figure 9: Linear Decision Boundary in 2-Dimensions



3.3 Decision Boundary

A decision boundary in logistic regression with two possible output classes is simply put, the plane (if we're dealing in 3 dimension/features) or the line (2 dimensions/features) that separates one class from another. More formally, and generalizing for n features, is the hyper-surface that partitions the underlying vector space into two sets, one for each class.

Graphically, can be seen, for 2-dimensions in 9. Note that the decision boundary is what our learning model produces, and is not necessarily a line as we'll see in other examples.

3.4 Non Linearly Separable Data

todo

3.5 Multiclass Logistic Regression

todo

4 Neural Networks

todo

5 Support Vector Machines

todo