Stanford Machine Learning - Week III

Eric N Johnson

July 24, 2016

1 Logistic Regression and Classification

Our target variable now is discrete. A few examples:

- Email: Spam or Not Spam?
- Online transactions: Fraudulent?

Response y is assigned into a class:

$$y \in \begin{cases} 0: & \text{“negative class”} \\ 1: & \text{“positive class”} \end{cases}$$

These are *binary classification problems*. Later we will work with *multiclass classification problems*. In general we cannot apply linear regression to classification problems.

What if $h_\theta(x) > 1$ or $h_\theta(x) < 0$? We will use *logistic regression* so that $0 \leq h_\theta(x) \leq 1$. Note that *logistic regression is a classification method*!

1.1 Hypothesis Representation

We want $0 \leq h_\theta(x) \leq 1$, so we have

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

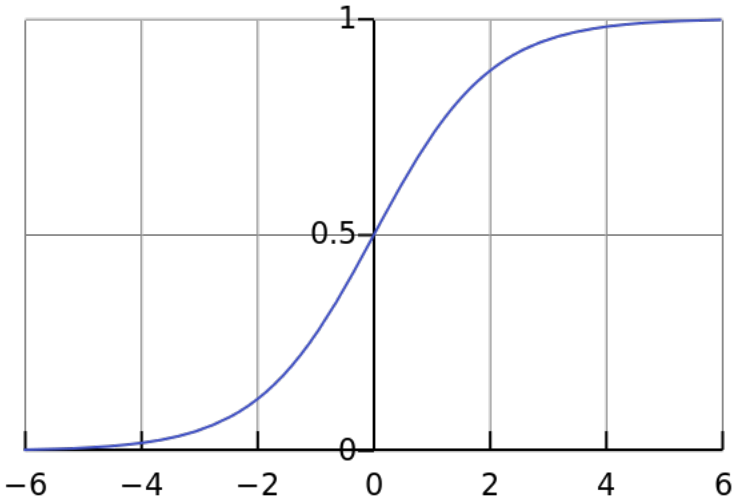
where

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

and thus

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

The *Sigmoid* or *logistic function* looks like this:



Our parameters are $\theta_0, \theta_1, ..., \theta_n$. We will develop a method for finding these parameters.

1.1.1 How do we Interpret Hypothesis Output?

We say

$$h_\theta(x) = \text{estimated probability that } y = 1 \text{ on input } x$$

So if $h_\theta(x) = 0.7$, we say that there is a 70% probability that the sample is in the positive class. In other words:

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

"Probability that $y = 1$ given x parameterized by θ".

Naturally, we have to follow axioms of probability. So we have the following relationship that always holds:

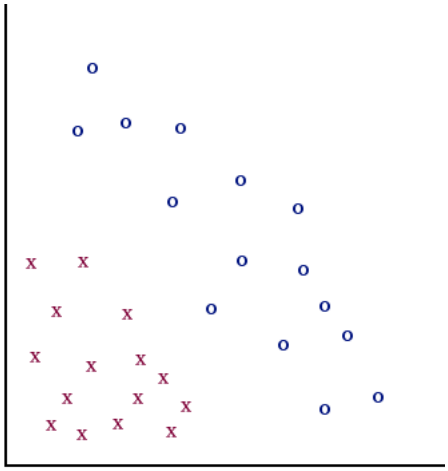
$$P(y = 1|x; \theta) + P(y = 0|x; \theta) = 1$$

1.2 The Decision Boundary

In general:

- We predict $y = 1$ if $h_\theta(x) \geq 0.5$, which corresponds to $\theta^T x \geq 0$, and
- $y = 0$ if $h_\theta(x) < 0.5$, which corresponds to $\theta^T x < 0$.

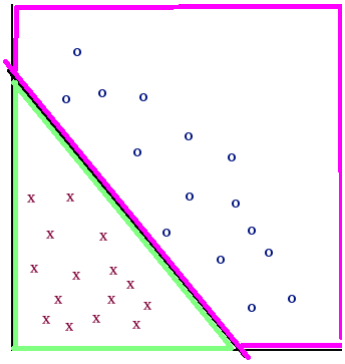
Here is an example. Suppose we have a training set that looks like this:



which we find has this hypothesis:

$$\begin{aligned} h_\theta(x) &= g(\theta^T x) \\ &= g(\theta_0 + \theta_1 x_1 + \theta_2 x_2) \\ &= g(-3 + (1)x_1 + (1)x_2) \end{aligned}$$

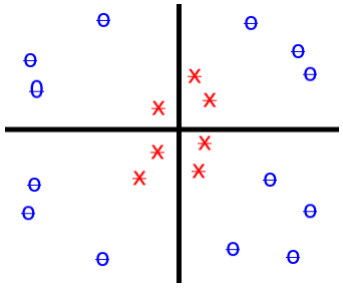
then we predict $y = 1$ if $-3 + x_1 + x_2 \geq 0$, and $y = 0$ if $-3 + x_1 + x_2 < 0$. In other words, $x_1 + x_2 \geq 3$, which we can draw on our plot of training examples.



The line that separates the two regions is called the *decision boundary*.

1.2.1 Nonlinear Decision Boundaries

We can also have something less straightforward like this:



with hypothesis

$$\begin{aligned} h_\theta(x) &= g(\theta^T x) \\ &= g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2) \end{aligned}$$

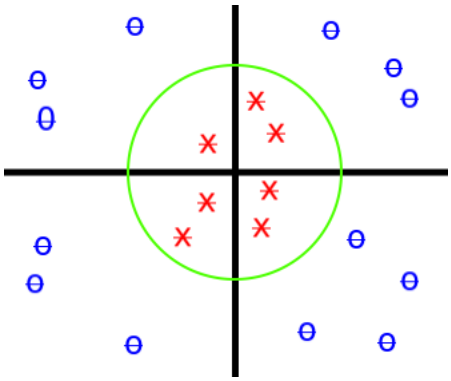
where the parameters are

$$\theta = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

which gives us this decision boundary:

- $y = 1$ if $-1 + x_1^2 + x_2^2 \geq 0$
- $y = 0$ if $-1 + x_1^2 + x_2^2 < 0$

In other words, we have the equation for a circle. If we are outside of the circle with radius 1 centered at the origin, we have $y = 1$.



Obviously these are nice examples – but we can get regions that are much more complicated.

1.3 Cost Function for Logistic Regression

How will we fit the parameters θ?

Suppose we have a training set with m training examples:

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(m)}, y^{(m)})\}$$

where

$$x \in \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{nx1},$$

$x_0 = 1$, $y \in \{0, 1\}$ and our hypothesis is

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

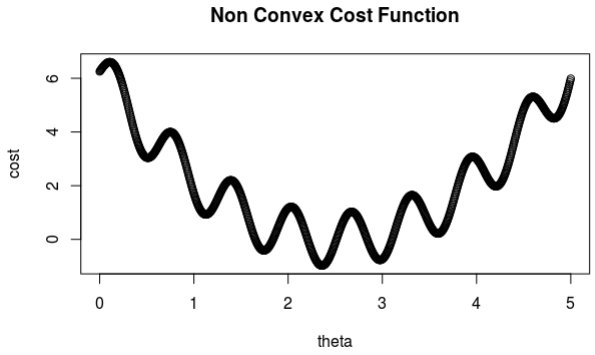
We define:

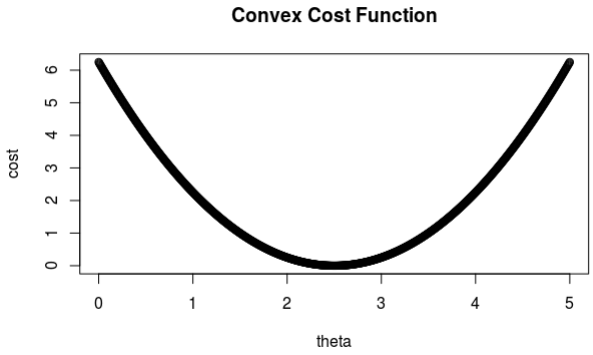
$$\text{cost}(h_\theta(x), y) = \frac{1}{2} (h_\theta(x) - y)^2$$

so cost is the sum of all of these m terms:

$$\begin{aligned} J(\theta) &= \sum_{i=1}^{m} \text{Cost}(h_\theta(x^{(i)}), y) \\ &= \frac{1}{2} \sum_{i=1}^{m} h_\theta(x^{(i)}) - y^{(i)})^2 \end{aligned}$$

However, we will end up calculating a non-convex function if we try to use this. We need, instead, to use a function with a unique minimum.





vs.

Def: Convex Function A function $f(x)$ is convex on an interval $[a, b]$ if for any two points x_1 and x_2 on $[a, b]$ and any λ where $0 < \lambda < 1$,

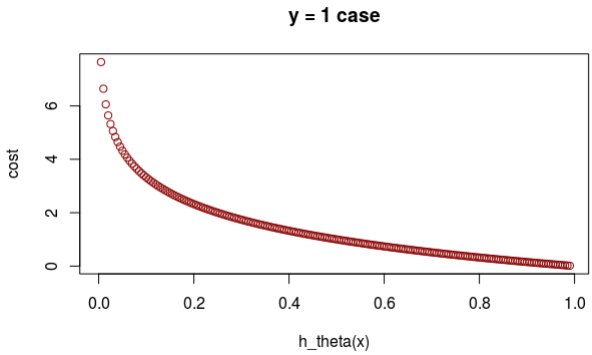
$$f(\lambda x_1 + (1 - \lambda x_2)) \leq \lambda f(x_1) + (1 - \lambda) f(x_2)$$

1.3.1 Convex Cost Function for Logistic Regression

Define

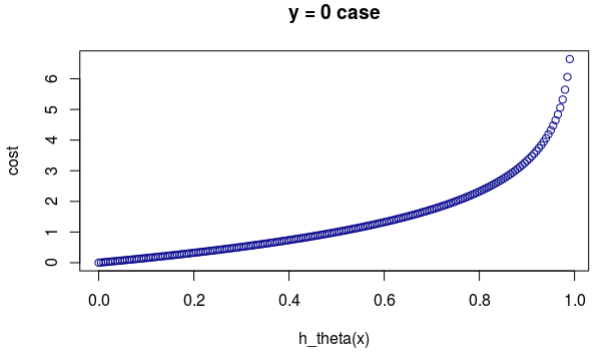
$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

It is helpful to plot this. Let $y = 1$ and $y = 0$, respectively:



The cost is zero if $y = 1$, $h_\theta(x) = 1$. But as $h_\theta(x) \to 0$, cost $\to \infty$. This captures the intuition that if $h_\theta(x) = 0$, we will penalize the learning system with a very large cost. In other words, we predict $P(y = 1|x; \theta) = 0)$ or that the probability that $y = 1$ is zero.

With $y = 0$, we do the opposite and get a plot that looks like this:



Again, as $h_\theta(x) \to 1$, the cost $\to \infty$. This occurs when we predict that $P(y = 0|x; \theta) = 0$

1.4 Logistic Regression Cost Function (Simplified)

From above we have that

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

and so

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

Note: $y = 0 \vee y = 1$ always. This fact allows us to write a simpler version:

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

Thus we may write our cost function as follows:

$$\begin{aligned} J(\theta) &= \frac{1}{m} \sum_{i=1}^{m} \text{Cost}(h_\theta(x^{(i)}) - y^{(i)}) \\ &= -\frac{1}{m} \left[\sum_{i=1}^{m} y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right] \end{aligned}$$

Note: This comes from the 'principle of maximum likelihood' in statistics.

To fit parameters θ, we find:

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

to make a new prediction given x:

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

and predict

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

We still have to actually find this minimum. Here is a Gradient Descent method to do so:

$$\begin{aligned} &\text{Repeat } \{ \\ &\quad \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \\ &\quad := \theta_j - \alpha \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)} \\ &\} \end{aligned}$$

and simultaneously update all θ_j. Note that this algorithm is identical for linear and logistic regression. They are not the same algorithm though because the hypothesis is different in each case. We also may apply the same process (plotting the descending cost) for making sure the algorithm is running correctly. It is faster to use a vectorized implementation of this algorithm:

$$\theta := \theta - \alpha \frac{1}{m} \sum_{i=1}^{m} \left[(h_\theta(x^{(i)}) - h^{(i)}) \cdot x^{(i)} \right]$$

1.4.1 Advanced Optimization

We will use some advanced optimization algorithms to run very quickly.

- Optimization algorithm: We have a cost function $J(\theta)$. We want to minimze this cost. Given θ, we need code to compute
  - $J(\theta)$
  - $\frac{\partial}{\partial \theta_j} J(\theta)$ for $j = 0, 1, ..., n$
  - Gradient Descent:

$$\begin{aligned} &\text{Repeat} \{ \\ &\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \\ &\} \end{aligned}$$

- Conjugate Gradient
- BFGS
- L-BFGS

There are some advantages of using these advanced algorithms. They pick α automatically. They are faster than Gradient Descent. But they are also more complex.

Example:

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

$\min_\theta J(\theta)$

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

$$\frac{\partial}{\partial \theta_1} J(\theta) = 2(\theta_1 - 5)$$
$$\frac{\partial}{\partial \theta_2} J(\theta) = 2(\theta_2 - 5)$$

(Clearly the minumum is at $\theta_1 = \theta_2 = 5$). Here is an implementation in Octave:

```
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);
```

then call one of the advanced algorithms:

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

This calls something called 'fminunc' or 'fmin unconstrained'. See 'help fminunc'. Evaluating this code in Octave returns:

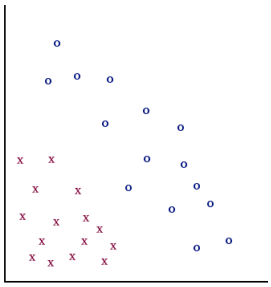
```
optTheta =

   5.0000
   5.0000

functionVal =    1.5777e-30
exitFlag =  1
```

2 Multiclass Classification

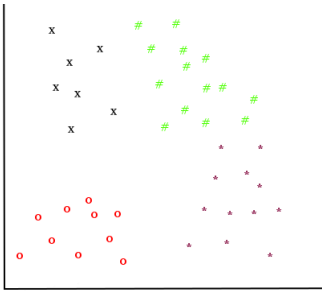
Examples:

- Email foldering: Work, friends, family, hobby
- Photos: Pets, people, lightning...
- Medical diagnosis: Not ill, flu, cold
- Weather: Sunny, cloudy, rainy, snow.

Previously we had binary classification problems that looked like



but now we will have things like this



2.1 One-Vs.-All Classification

AKA 'One-Vs.-Rest' classification. We create a training set so that one of the classes (say $h_\theta^{(1)}(x)$) fits into the *positive class* and all of the others fit into the *negative class*. This gives us a decision boundary to determine that class. Then we repeat this to find decision boundaries for all of the classes $h_\theta^{(2)}(x), h_\theta^{(3)}(x)$, etc.

In other words we train a logistic regression classifier $h_\theta^{(i)}(x)$ for each class i to predict the probability that $y = i$. Then on a new input x we pick the class i that maximizes

$$\max_i h_\theta^{(i)}(x).$$

We may use this method to use Logistic Classification on multiple classification problems.

3 The Problem of Overfitting

- **Underfit** models have **high bias**.
- **Overfit** models or models with **high variance** do a good job fitting a training set but we won't do as well on new data. These models "try too hard" to fit training data while they fail to generalize.

Balancing **bias** and **variance** is key to building a good model.

How do we address overfitting? We will go into this into more detail. If we believe overfitting is occuring, how may we address it?

1. Reduce the number of features $x_1, x_2, ...$
   - Manually select which features to keep.
   - Use a model-selection algorithm to decide which features to keep.
2. Use **regularization**.
   - Keep all of the features but reduce the magnitude / values of parameters θ_j. This works well if we have a lot of features which all contribute a little bit to predicting y.

3.1 Cost Function: Regularization

Suppose we have a model that overfits data such as

$$\theta_0 + \theta_1 x + \theta^2 x + \theta_3 x^3 + \theta_4 x^4$$

We may penalize the model for this by making θ_3 and θ_4 very small:

$$\min_\theta \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)} - y^{(i)} \right)^2 + 1000\ \theta_3^2 + 1000\ \theta_4^2$$

If we have $\theta_3 \approx 0$ and $\theta_4 \approx 0$, we will get a model that is closer to quadratic.

Small values for parameters $\theta_0, \theta_1, ..., \theta_n$

- Simpler hypothesis
- Less prone to overfitting

Here is an example:

- Suppose our housing example has features $x_1, x_2, ..., x_{100}$
- and parameters $\theta_0, \theta_1, ..., \theta_{100}$

We may then modify our cost function to add a regularization term

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

Note: We do not regularize θ_0 by convention. The regularization term is

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

where λ is called *regularization parameter*. The '$1000\ \theta_3^2$' term in the expression we started with is the regularization parameter for that expression. It keeps the parameters small while still allowing the model to fit the training data well.

If λ is set too large we will penalize these terms too much. We will then get a model that underfits training data - or has too much bias toward an oversimplified model.

3.2 Regularized Linear Regression

We have two models so far for linear regression:

- Gradient Descent
- Normal Equations

Here is a regularized linear model:

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

where we seek $\min_\theta J(\theta)$.

- Gradient Descent with Regularization:

$$\begin{aligned}
&\text{Repeat}\{\\
&\quad \theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)} - y^{(i)}\right)x_0^{(i)}\\
&\quad \theta_j := \theta_j - \alpha\left[\frac{1}{m}\sum_{j=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)} + \frac{\lambda}{m}\theta_j\right]\\
&\quad\quad := \theta_j(1-\alpha)\frac{\lambda}{m} - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)}\\
&\}
\end{aligned}$$

  Note that this second term is the partial derivative of $J(\theta)$ for $j = 1, ..., n$. The first term is the partial derivative of $J(\theta)$ with respect to θ_0. The term

$$1 - \alpha\frac{\lambda}{m} < 1$$

  has the effect of shrinking θ_j. This occurs every step - which keeps it from getting too large and overfitting our training data.

- Normal Equation with Regularization
  To use the normal equations we have

$$X = \begin{bmatrix}(x^{(1)})^T\\(x^{(2)})^T\\\vdots\\(x^{(m)})^T\end{bmatrix} \in \mathbb{R}^{m\times(n+1)}$$

  and

$$y = \begin{bmatrix}y^{(1)}\\y^{(2)}\\\vdots\\y^{(m)}\end{bmatrix} \in \mathbb{R}^{m\times 1}$$

  to minimize our cost function $J(\theta)$, we say

$$\theta = (X^TX)^{-1}X^Ty$$

  which regularized becomes

$$\theta = \left(X^TX + \lambda\begin{bmatrix}0&0&0&\cdots&0\\0&1&0&\cdots&0\\0&0&1&\cdots&0\\\vdots&\vdots&\vdots&\ddots&\vdots\\0&0&0&0&1\end{bmatrix}\right)^{-1}X^Ty$$

For example, if $n = 2$, we have the regularization matrix

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

What about non-invertibility of the terms?

- Suppose $m \leq n$... then $X^T X$ will be singular. But our $X^T X + \lambda R$ term (where R is the regularization matrix) will necessarily be invertible! We will not prove that here - but that is a good thing to remember.

- Logistic Regression with Regularization

Recall our cost function for Logistic Regression

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

Now we add a regularization term

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

for $\theta_1, \theta_2, ..., \theta_n$ to obtain a regularized logistic regression:

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

Here is our new regularized algorithm:

Repeat{

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

$$\theta_1 := \theta_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right) x_j^{(i)} + \frac{\lambda}{m}\theta_j\right]$$

}

for $j = 1, 2, ..., n$, where

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

We can plot this function as the number of iterations increases and see the cost decrease

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

3.3 Advanced Optimization for Regularized Logistic Regression

We can implement this in Octave using the following:

```
function [jVal, gradient] = costFunction(theta)
jVal = code to compute J(theta)
gradient(1) = code to compute \frac{\partial}{\partial \theta_0} J(\theta)
gradient(2) = code to compute \frac{\partial}{\partial \theta_1} J(\theta)
...
gradient(n+1) = code to compute \frac{\partial}{\partial \theta_n} J(\theta)
```

which can be passed to 'fminunc' to solve for θ.