# Regularized Linear Regression

We can apply regularization to both linear regression and logistic regression. We will approach linear regression first.

### Gradient Descent

We will modify our gradient descent function to separate out $\theta_0$ from the rest of the parameters because we do not want to penalize $\theta_0$.

$$\begin{aligned}&\text{Repeat } \{\\ &\quad \theta_0 := \theta_0 - \alpha \ \frac{1}{m} \ \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)} \\ &\quad \theta_j := \theta_j - \alpha \left[ \left( \frac{1}{m} \ \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \right) + \frac{\lambda}{m} \theta_j \right] \qquad j \in \{1, 2 ... n\} \\ &\}\end{aligned}$$

The term $\frac{\lambda}{m} \theta_j$ performs our regularization. With some manipulation our update rule can also be represented as:

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

The first term in the above equation, $1 - \alpha \, \frac{\lambda}{m}$ will always be less than 1. Intuitively you can see it as reducing the value of $\theta_j$ by some amount on every update. Notice that the second term is now exactly the same as it was before.

### Normal Equation

Now let's approach regularization using the alternate method of the non-iterative normal equation.

To add in regularization, the equation is the same as our original, except that we add another term inside the parentheses:

$$\begin{aligned}&\theta = \left( X^T X + \lambda \cdot L \right)^{-1} X^T y \\ &\text{where } \ L = \begin{bmatrix} 0 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{bmatrix}\end{aligned}$$

L is a matrix with 0 at the top left and 1's down the diagonal, with 0's everywhere else. It should have dimension (n+1)×(n+1). Intuitively, this is the identity matrix (though we are not including $x_0$), multiplied with a single real number λ.

Recall that if m ≤ n, then $X^T X$ is non-invertible. However, when we add the term λ·L, then $X^T X$ + λ·L becomes invertible.

### Octave Practice Exercises

In this section, you will implement the cost function and gradient for regularized linear regression.

Your function will take in:

| | |
|---|---|
| $\text{theta} \in \mathbb{R}^{n+1}$ | A $(n+1)$-dimensional vector representing parameters of the regularized linear regression model, including a term for the bias |
| $\text{X} \in \mathbb{R}^{m \times n+1}$ | A $m \times n+1$ matrix containing the features for each data point. |
| $\text{y} \in \mathbb{R}^{m}$ | A $m$-dimensional vector representing the y-values for each example. |
| $\text{lambda} \in \mathbb{R}$ | The regularization parameter. |

Where $m$ is the number of data points and $n > 1$ is the number of features. Your cost function should work for a range of $m$ and $n$. You may assume that the first dimension of $X$ is the intercept term (i.e. all $1$s).

Recall that the cost function is

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

**Note that in Octave, vectors are indexed from 1 onwards; that means that theta(1) in your code refers to $\theta_0$**

Remember to vectorize as much of your function as you can.

```
function cost = cost(theta, X, y, lambda)

% Your code goes here.
cost = 0;
[m, n] = size(X);
theta_1 = theta;
theta_1(1) = 0;
cost = (sum(((theta'*X')' - y).^2) + lambda*( sum(theta_1.^2) ) )/2/m;


endfunction
```

Run

```
Good job!
```

Write a function in Octave that computes the gradient for the cost function of regularized linear regression (taking in the same arguments as your cost function above). Again, remember to vectorize as much of your function as you can.

Recall that the gradient is

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

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

Tip: Remember that grad should have the same dimensions as $\theta$.

```
function grad = grad(theta, X, y, lambda)

% Your code goes here.

[m, n] = size(X);
grad = ones(length(theta), 1);
for j = 2: length(theta)
  grad(j) = sum( ((theta'*X')' - y)'* X(:,j) )/m + lambda/m*theta(j);
end
grad(1) = sum( ((theta'*X')' - y)'*X(:,1) )/m;

endfunction
```

Run

Good job!

Mark as completed