Coursera

- Harmandeep Singh

ML:Linear Regression with Multiple Variables

Contents

- 1 Multiple Features
- 2 Cost function
- 3 Gradient Descent for Multiple Variables
  - 3.1 Matrix Notation
- 4 Feature Normalization
  - 4.1 Quiz question #1 on Feature Normalization (Week 2, Linear Regression with Multiple Variables)
- 5 Gradient Descent Tips
- 6 Features and Polynomial Regression
  - 6.1 Polynomial Regression
- 7 Normal Equation
  - 7.1 Normal Equation Demonstration
  - 7.2 Normal Equation Noninvertibility

Multiple Features

Linear regression with multiple variables is also known as "multivariate linear regression".

We now introduce notation for equations where we can have any number of input variables.

$$ \begin{align*} x_j^{(i)} &= \text{value of feature } j \text{ in the }i^{th}\text{ training example} \newline x^{(i)}& = \text{the column vector of all the feature inputs of the }i^{th}\text{ training example} \newline m &= \text{the number of training examples} \newline n &= \left| x^{(i)} \right| \; \text{(the number of features)} \end{align*} $$

Now define the multivariable form of the hypothesis function as follows, accomodating these multiple features:

$$ h_\theta (x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \cdots + \theta_n x_n $$

In order to have little intuition about this function, we can think about $$\theta_0$$ as the basic price of a house, $$\theta_1$$ as the price per square meter, $$\theta_2$$ as the price per floor, etc. $$x_1$$ will be the number of square meters in the house, $$x_2$$ the number of floors, etc.

Using the definition of matrix multiplication, our multivariable hypothesis function can be concisely represented as:

$$ \begin{align*} h_\theta(x) = \begin{bmatrix} \theta_0 \hspace{2em} \theta_1 \hspace{2em} ... \hspace{2em} \theta_n \end{bmatrix} \begin{bmatrix} x_0 \newline x_1 \newline \vdots \newline x_n \end{bmatrix} = \theta^T x \end{align*} $$

This is a vectorization of our hypothesis function for one training example; see the lessons on vectorization to learn more.

[**Note**: So that we can do matrix operations with theta and x, we will set $$x^{(i)}_0 = 1$$, for all values of $$i$$. This makes the two vectors 'theta' and $$x^{(i)}$$ match each other element-wise (that is, have the same number of elements: $$n + 1$$).]

Now we can collect all $$m$$ training examples each with $$n$$ features and record them in an $$n+1$$ by $$m$$ matrix. In this matrix we let the value of the subscript (feature) also represent the row number (except the initial row is the "zeroth" row), and the value of the superscript (the training example) also represent the column number, as shown here:

$$ \begin{align*} X = \begin{bmatrix} x^{(1)}_0 \hspace{2em} x^{(2)}_0\hspace{2em} ... \hspace{2em} x^{(m)}_0 \newline x^{(1)}_1 \hspace{2em} x^{(2)}_1 \hspace{2em} ... \hspace{2em} x^{(m)}_1 \newline \vdots \newline x^{(1)}_n \hspace{2em} x^{(2)}_n \hspace{2em} ... \hspace{2em} x^{(m)}_n \newline \end{bmatrix} &= \begin{bmatrix} 1 & 1 & ... & 1 \newline x^{(1)}_1 & x^{(2)}_1 &... & x^{(m)}_1 \newline \vdots \newline x^{(1)}_n & x^{(2)}_n & ... & x^{(m)}_n \newline \end{bmatrix} \end{align*} $$

Notice above that the first column is the first training example (like the vector above), the second column is the second training example, and so forth.

Now we can define $$h_\theta(X)$$ as a row vector that gives the value of $$h_\theta(x)$$ at each of the $$m$$ training examples:

$$ \begin{align*} h_\theta(X) = \begin{bmatrix} \theta_0 x^{(1)}_0 + \theta_1 x^{(1)}_1 + ... + \theta_n x^{(1)}_n \hspace{3em} \theta_0 x^{(2)}_0 + \theta_1 x^{(2)}_1 + ... + \theta_n x^{(2)}_n \hspace{3em} ... \hspace{3em} \theta_0 x^{(m)}_0 + \theta_1 x^{(m)}_1 + ... + \theta_n x^{(m)}_n \newline \end{bmatrix} \end{align*} $$

But again using the definition of matrix multiplication, we can represent this more concisely:

$$ \begin{align*} h_\theta(X) = \begin{bmatrix} \theta_0 \hspace{2em} \theta_1 \hspace{2em} ... \hspace{2em} \theta_n \end{bmatrix} \begin{bmatrix} x^{(1)}_0 \hspace{2em} x^{(2)}_0\hspace{2em} ... \hspace{2em} x^{(m)}_0 \newline x^{(1)}_1 \hspace{2em} x^{(2)}_1 \hspace{2em} ... \hspace{2em} x^{(m)}_1 \newline \vdots \newline x^{(1)}_n \hspace{2em} x^{(2)}_n \hspace{2em} ... \hspace{2em} x^{(m)}_n \newline \end{bmatrix}= \theta^T X \end{align*} $$

Note: this version of the hypothesis function assumes the matrix $$X$$ and vector $$\theta$$ store their values as follows:

$$ \begin{align*} X = \begin{bmatrix} x^{(1)}_0 & x^{(2)}_0 & x^{(3)}_0 \newline x^{(1)}_1 & x^{(2)}_1 & x^{(3)}_1 \end{bmatrix} & ,\theta = \begin{bmatrix} \theta_0 \newline \theta_1 \newline \end{bmatrix} \end{align*} $$

You might instead store the training examples in $$X$$ row-wise, like such:

$$ \begin{align*} X = \begin{bmatrix} x^{(1)}_0 & x^{(1)}_1 \newline x^{(2)}_0 & x^{(2)}_1 \newline x^{(3)}_0 & x^{(3)}_1 \end{bmatrix} & ,\theta = \begin{bmatrix} \theta_0 \newline \theta_1 \newline \end{bmatrix} \end{align*} $$

In which case you would calculate the hypothesis function with:

$$ h_\theta(X) = X \theta $$

However in this case, $$h_\theta(X)$$ would be a column vector, not a row vector. **For the rest of this page, and other pages of the wiki, $$X$$ will represent a matrix of training examples $$x^{(i)}$$ stored row-wise.**

Cost function

For the parameter vector $$\theta$$ (of type $$\mathbb{R}^{n+1}$$ or in $$\mathbb{R}^{(n+1) \times 1}$$), the cost function is:

$$J(\theta) = \dfrac {1}{2m} \displaystyle \sum_{i=1}^m \left (h_\theta (x^{(i)}) - y^{(i)} \right)^2$$

The vectorized version is:

$$J(\theta) = \dfrac {1}{2m} (X\theta - \vec{y})^{T} (X\theta - \vec{y})$$

Where $$\vec{y}$$ denotes the vector of all y values.

Gradient Descent for Multiple Variables

The gradient descent equation itself is generally the same form; we just have to repeat it for our 'n' features:

$$\begin{align*} & \text{repeat until convergence:} \; \lbrace \newline \; & \theta_0 := \theta_0 - \alpha \frac{1}{m} \sum\limits_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_0^{(i)}\newline \; & \theta_1 := \theta_1 - \alpha \frac{1}{m} \sum\limits_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_1^{(i)} \newline \; & \theta_2 := \theta_2 - \alpha \frac{1}{m} \sum\limits_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_2^{(i)} \newline & \cdots \newline \rbrace \end{align*}$$

In other words:

$$\begin{align*} & \text{repeat until convergence:} \; \lbrace \newline \; & \theta_j := \theta_j - \alpha \frac{1}{m} \sum\limits_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)} \; & \text{for j := 0..n} \newline \rbrace \end{align*}$$

Matrix Notation

The Gradient Descent rule can be expressed as:

$$ \large \theta := \theta - \alpha \nabla J(\theta) $$

Where $$\nabla J(\theta)$$ is a column vector of the form:

$$\large \nabla J(\theta) = \begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_0} \newline \frac{\partial J(\theta)}{\partial \theta_1} \newline \vdots \newline \frac{\partial J(\theta)}{\partial \theta_n} \end{bmatrix} $$

The j-th component of the gradient is the summation of the product of two terms:

$$\begin{align*} \; &\frac{\partial J(\theta)}{\partial \theta_j} &=& \frac{1}{m} \sum\limits_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x_j^{(i)} \newline \; & &=& \frac{1}{m} \sum\limits_{i=1}^{m} x_j^{(i)} \cdot \left(h_\theta(x^{(i)}) - y^{(i)} \right) \end{align*}$$

Sometimes, the summation of the product of two terms can be expressed as the product of two vectors.

Here, the term $$x_j^{(i)}$$ represents the $$m$$ elements of the $$j$$-th column $$\vec{x_j}$$ ($$j$$-th feature $$\vec{x_j}$$) of the training set $$X$$.

The other term $$\left(h_\theta(x^{(i)}) - y^{(i)} \right)$$ is the vector of the deviations between the predictions $$h_\theta(x^{(i)})$$ and the true values $$y^{(i)}$$. Re-writing $$\frac{\partial J(\theta)}{\partial \theta_j}$$, we have:

$$\begin{align*} \; &\frac{\partial J(\theta)}{\partial \theta_j} &=& \frac1m \vec{x_j}^{T} (X\theta - \vec{y}) \newline \newline \newline \; &\nabla J(\theta) & = & \frac 1m X^{T} (X\theta - \vec{y}) \newline \end{align*}$$

Finally, the matrix notation (vectorized) of the Gradient Descent rule is:

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

Feature Normalization

We can speed up gradient descent by having each of our input values in roughly the same range. This is because $$\theta$$ will descend quickly on small ranges and slowly on large ranges, and so will oscillate inefficiently down to the optimum when the variables are very uneven.

The way to prevent this is to modify the ranges of our input variables so that they are all roughly the same. Ideally:

$$ -1 \le x_i \le 1 $$

or

$$ -0.5 \le x_i \le 0.5 $$

These aren't exact requirements; we are only trying to speed things up. The goal is to get all input variables into roughly one of these ranges, give or take a few.

Two techniques to help with this are **feature scaling** and **mean normalization**. Feature scaling involves dividing the input values by the range (i.e. the maximum value minus the minimum value) of the input variable, resulting in a new range of just 1. Mean normalization involves subtracting the average value

for an input variable from the values for that input variable, resulting in a new average value for the input variable of just zero. To implement both of these techniques, adjust your input values as shown in this formula:

$$ x_i := \dfrac{x_i - \mu_i}{s_i} $$

Where $$\mu_i$$ is the **average** of all the values and $$s_i$$ is the maximum of the range of values *minus* the minimum or $$s_i$$ is the standard deviation.

Example: $$x_i$$ is housing prices in range 100-2000. Then, $$x_i := \dfrac{price-1000}{1900}$$, where 1000 is the average price and 1900 is the maximum (2000) minus the minimum (100).

Quiz question #1 on Feature Normalization (Week 2, Linear Regression with Multiple Variables)

Your answer should be rounded to exactly two decimal places. Use a '.' for the decimal point, not a ','. The tricky part of this question is figuring out which feature of which training example you are asked to normalize. Note that the mobile app doesn't allow entering a negative number (Jan 2016), so you will need to use a browser to submit this quiz if your solution requires a negative number.

Gradient Descent Tips

Debugging gradient descent. Make a plot with *number of iterations* on the x-axis. Now plot the cost function, $$J(\theta)$$ over the number of iterations of gradient descent. If $$J(\theta)$$ ever increases, then you probably need to decrease $$\alpha$$.

Automatic convergence test. Declare convergence if $$J(\theta)$$ decreases by less than $$E$$ in one iteration, where $$E$$ is some small value such as $$10^{-3}$$. However in practice it's difficult to choose this threshold value.

It has been proven that if learning rate $$\alpha$$ is sufficiently small, then $$J(\theta)$$ will decrease on every iteration. Andrew Ng recommends decreasing $$\alpha$$ by multiples of 3.

Features and Polynomial Regression

We can improve our features and the form of our hypothesis function in a couple different ways.

We can **combine** multiple features into one. For example, we can combine $$x_1$$ and $$x_2$$ into a new feature $$x_3$$ by taking $$x_1 \cdot x_2$$.

Polynomial Regression

Our hypothesis function need not be linear (a straight line) if that does not fit the data well.

We can **change the behavior or curve** of our hypothesis function by making it a quadratic, cubic or square root function (or any other form).

For example, if our hypothesis function is $$h_\theta(x) = \theta_0 + \theta_1 x_1$$ then we can simply **duplicate** the instances of $$x_1$$ to get the quadratic function $$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2$$ or the cubic function $$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^3$$

In the cubic version, we have created new features $$x_2$$ and $$x_3$$ where $$x_2 = x_1^2$$ and $$x_3 = x_1^3$$.

To make it a square root function, we could do: $$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 \sqrt{x_1}$$

One important thing to keep in mind is, if you choose your features this way then feature scaling becomes very important.

eg. if $$x_1$$ has range 1 - 1000 then range of $$x_1^2$$ becomes 1 - 1000000 and that of $$x_1^3$$ becomes 1 - 1000000000

Normal Equation

The "normal equation" is a version of finding the optimum **without iteration.**

The proof (http://en.wikipedia.org/wiki/Linear_least_squares_(mathematics)) for this equation requires knowledge of linear algebra and is fairly involved, so you do not need to worry about the details.

$$ \large \theta = (X^T X)^{-1}X^T y $$

There is **no need** to do feature scaling with the normal equation.

The following is a comparison of gradient descent and the normal equation:

Gradient Descent	Normal Equation
Need to choose alpha	No need to choose alpha
Needs many iterations	No need to iterate
Works well when n is large	Slow if n is very large

With the normal equation, computing the inversion has complexity $$\mathcal{O}(n^3)$$. So if we have a very large number of features, the normal equation will be slow. In practice, according to A. Ng, when n exceeds 10,000 it might be a good time to go from a normal solution to an iterative process.

Normal Equation Demonstration

*$$\theta$$ is a $$(n+1)$$x$$1$$ matrix

*$$X$$ is a $$m$$x$$(n+1)$$ matrix so $$X^T$$ is a $$(n+1)$$x$$m$$ matrix

*$$\vec{y}$$ is a $$m$$x$$1$$ vector

$$X \theta= \vec{y}$$ *The point is to inverse $$X$$, but as $$X$$ is not a square matrix we need to use $$X^T$$ to have a square matrix*

$$ (X^T)X \theta = (X^T)\vec{y} $$

Associative matrix multiplication $$ (X^TX) \theta = X^T\vec{y} $$

Assuming $$ (X^T X)$$ invertible $$ \theta = (X^T X)^{-1}X^T \vec{y} $$

Normal Equation Noninvertibility

When implementing the normal equation in octave we want to use the 'pinv' function rather than 'inv.'

$$X^T X$$ may be **noninvertible**. The common causes are:

- Redundant features, where two features are very closely related (i.e. they are linearly dependent)
- Too many features (e.g. $$m \leq n$$). In this case, delete some features or use "regularization" (to be explained in a later lesson).

Solutions to the above problems include deleting a feature that is linearly dependent with another or deleting one or more features when there are too many features.

Next: Octave Tutorial Back to Index: Main

Category: ML:Lecture Notes