线性回归 （Linear Regression）

## Linear Regression

线性回归的hypothesis(假设函数)写成：

C:\Users\phenix\AppData\Local\Temp\enhtmlclip\Image.png

其中C:\Users\phenix\AppData\Local\Temp\enhtmlclip\Image(1).png表示权重，n表示变量个数（即特征点的维度）,x为特征点。C:\Users\phenix\AppData\Local\Temp\enhtmlclip\Image(1).png和x 都是向量。

  损失函数（cost function）为:

C:\Users\phenix\AppData\Local\Temp\enhtmlclip\Image(2).png

其中m为训练集的大小。

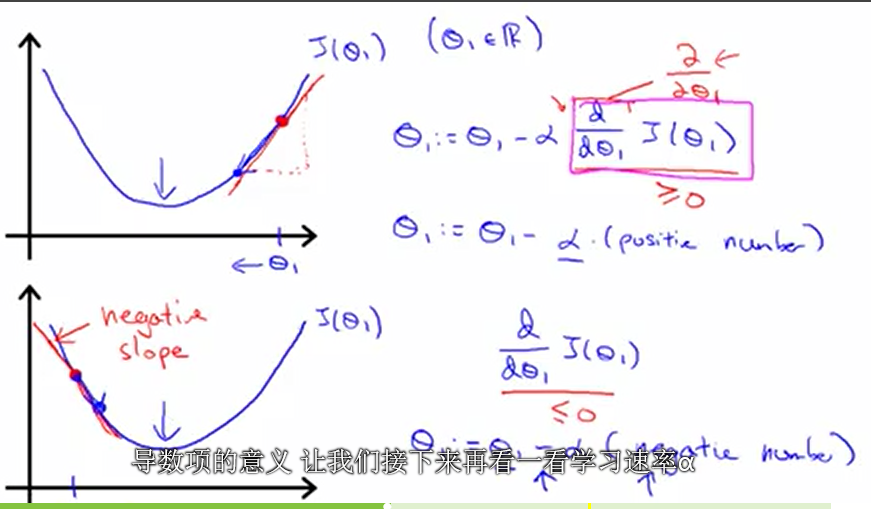
      回归最终的目标根据现有的训练机使得损失函数尽量小。一般有如下几种方法：

1. LMS(least mean squares,最小均方根) algorithm

该方法使用梯度下降法（gradient descent）,首先初始化一个值，然后迭代的更新：

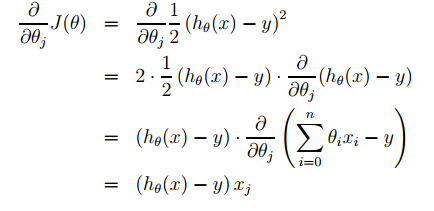
C:\Users\phenix\AppData\Local\Temp\enhtmlclip\Image(3).png

     该规则同时更新所有的C:\Users\phenix\AppData\Local\Temp\enhtmlclip\Image(4).png（j=0,...,n,）.其中C:\Users\phenix\AppData\Local\Temp\enhtmlclip\Image(5).png叫做学习速率（learning rate），控制更新幅度.下图解释了为什么可以使用梯度下降来更新theta。



      a的取值不能太大也不能太小。如果太小需要经过许多步才能收敛；太大的话会越过最低点，甚至有可能发散。

      对于求偏微分项呢，只有一个训练例子开始,有如下推导：



 这样对于单个训练例子，更新规则如下：

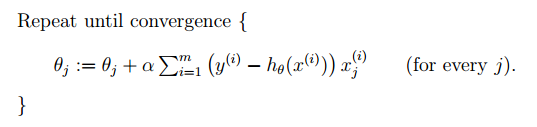
C:\Users\phenix\AppData\Local\Temp\enhtmlclip\Image(8).png

  这个更新规则就叫LMS.有几个特性：

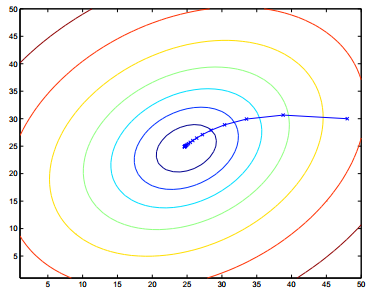
       \*误差项 C:\Users\phenix\AppData\Local\Temp\enhtmlclip\Image(9).png，表明了每次更新的幅度。如果预测结果与实际差值较大更新的幅度较大，反之，较小。

       对于训练集大小多于1的情况更新规则有如下两种：

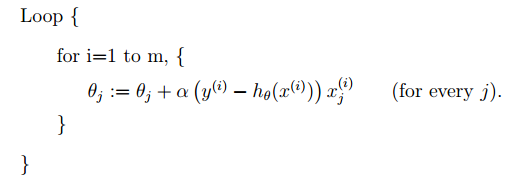
       a. batch gradient descent (批梯度下降)



      这种方式对于Convex function总能够达到全局优化。二次函数（quadratic function）的批梯度下降的J变化过程如下图:



  b. stochastic gradient descent (随机梯度下降)，又叫incremental gradient descent(增量梯度下降)



      该方法遍历训练集，每遇到一个训练例子就更新。与batch gradient descent 相比：

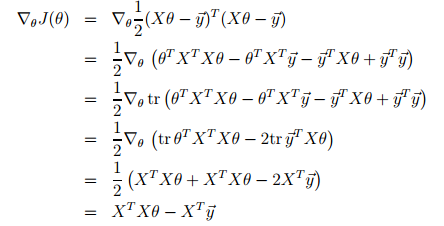
        \* 速度快（快速收敛）；

        \* 不会收敛到最优值，但离最优值很接近

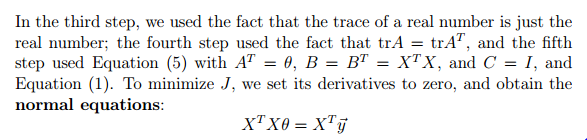
        总体来说，比batch gradient descent 好。

1. The normal equations.(正规方程)

最小化J的第二种方法用正规方程。函数J对于C:\Users\phenix\AppData\Local\Temp\enhtmlclip\f8f4621eb6028a98a985bc548e1a5e45[1].png求导，矩阵求导推导公式如下：



   推导说明：



   最后可得：

C:\Users\phenix\AppData\Local\Temp\enhtmlclip\Image(15).png

两种方法的比较：

|  |  |
| --- | --- |
| Gradient Descent | Normal Equation |
| \* Need to choose a | \* no need to choose a |
| \* Needs many iterations | \* don't need to iterate |
| \* works well even when n is large | \* need to compute C:\Users\phenix\AppData\Local\Temp\enhtmlclip\Image(16).png O(n^3) |
|  | \* slow if n is very large |

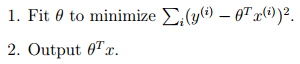
what if X'X is non-invertiable?

 ~~Redundant features(linearly dependent).

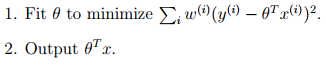
 ~~ Too many features (e.g. m<=n)

    --Delete some features ,or use regularization.

## Locally weighted linear regression

原始的线性回归算法在预测一个点时的做法如下

   而LWR 算法如下：



  其中权重w(i)可由C:\Users\phenix\AppData\Local\Temp\enhtmlclip\Image(19).png计算得到。

   包含的含义是，对于接近需要预测的x点的训练点x(i)值的权重更大，远离则权重小。

   The parameter τ controls how quickly the weight of a training example falls off with distance of its x(i) from the query point x; τ is called the bandwidth parameter。

    LWR是一种non-parametric算法，一种non-parametric 是指随着训练集的增加，需要保留的信息也随之线性增加；

而之前的linear regression algorithm 是parametric 算法，是指一旦 被评估出来，训练集就不在需要；

**Reference:**

1. Machine Learning Course- Stanford Unversity
2. Machine Learning For Computer Vision – Iasonas Kokkinos
3. [Linear Regression Implementation code.](https://github.com/PhenixI/machine-learning/tree/master/machine-learning-ex1/machine-learning-ex1)

***Q: Why is the cost function about the sum of squares, rather than the sum of cubes?***

A: The sum of squares isn’t the only possible cost function, but it has many nice properties. Squaring the error means that an overestimate is "punished" just the same as an underestimate: an error of -1 is treated just like +1, and they two equal but opposite errors can’t cancel each other. If we cube the error, we lose this property. Big errors are punished more than small ones, so an error of 2 becomes 4.

***Q: Why does 1/(2 \* m) make math easier?***

A: When we differentiate the cost to calculate the gradient, we get an factor of 2 due to the exponent inside the sum. The two factors will cancel out, giving a slightly simpler formula