# 机器学习笔记

## 前言：

吴恩达的机器学习及深度学习两门课，github上有人做了详尽的笔记

<https://github.com/fengdu78/Coursera-ML-AndrewNg-Notes>

因为有了完整的笔记，这里笔者就不打算记录完整的笔记，而是记录一些对于笔者而言重要的笔记，所以对大家而言就没有什么参考价值了。

## 一、引言

无监督学习

无监督学习中的数据没有任何标签，我们仅有一个数据集，不知如何处理。无监督学习能判断数据中的聚集簇，从而将数据分类，这称之为聚类算法（cluster）。

## 二、单变量线性回归

如果以y = ax + b这样的线性方程来拟合数据，并且以mse（均方误差）为代价函数来计算，那么一下是推导过程。

方程



代价函数

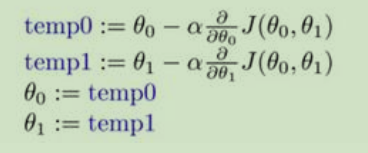


目标：代价函数最小

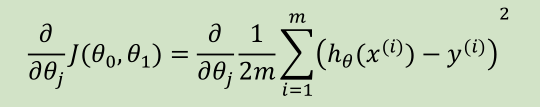
方法：通过更改theta1和theta2的值，使得代价函数降低。而最快的方法则是沿着梯度的方向改变值。Alpha是学习率，那么theta的更新函数如下所示

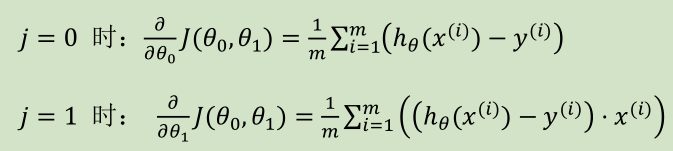


更具体一点，计算过程应该如下：

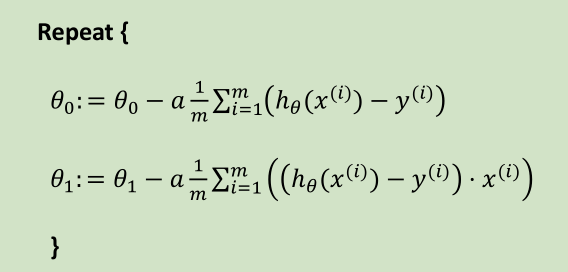


关于偏导数的公式推导：





那算法改写如下：



## 三、线性代数回顾

### 规律

矩阵乘法不满足交换律



矩阵乘法满足结合律



单位矩阵，除对角线元素为1，其余元素都为0的矩阵。一般用I或者E表示。对于单位矩阵，有

AI = IA = A

### 逆矩阵

只有方阵才有逆矩阵。

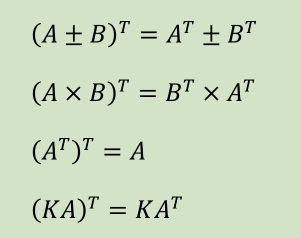
如果有逆矩阵，则

没有逆矩阵的矩阵成为奇异矩阵(Singular)或者退化矩阵(degenerate)

Numpy中求矩阵的额逆的方法

np.linalg.inv()

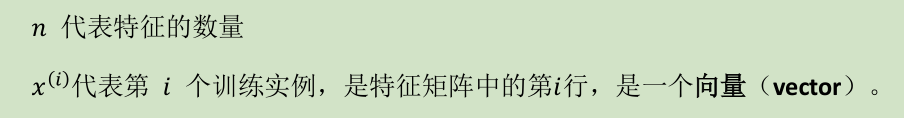
### 矩阵转置

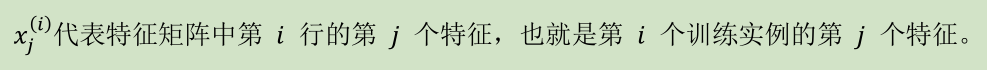


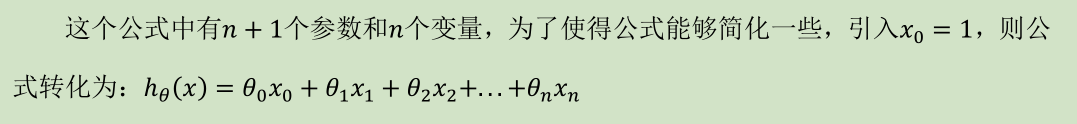
## 四、多变量线性回归

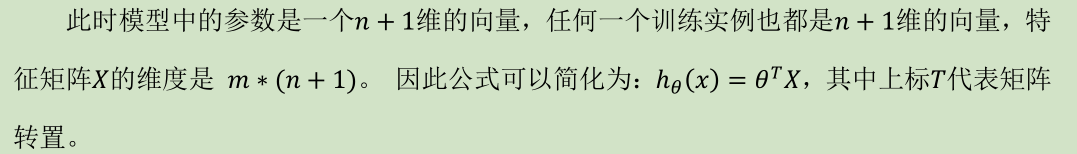
### 符号含义

有了多维特征后，公式推导的符号含义：









### 梯度下降

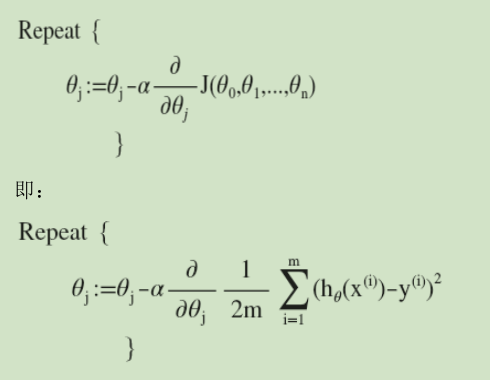
#### 公式



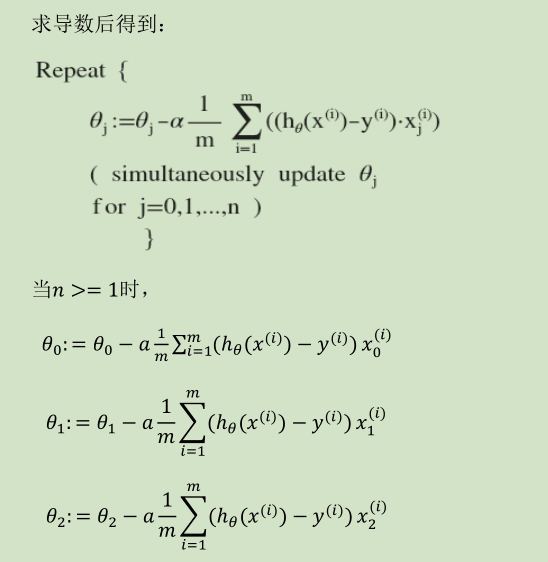
#### 代价函数：



梯度下降算法为：



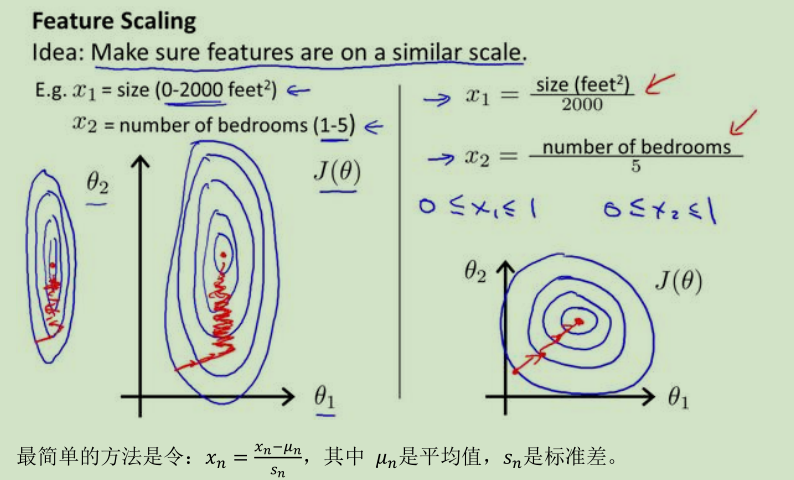
计算后得到



### 特征缩放

在我们面对多维特征问题的时候，我们要保证这些特征都具有相近的尺度，这将帮助梯度下降算法更快地收敛。

以房价问题为例，假设我们使用两个特征，房屋的尺寸和房间的数量，尺寸的值为 0-2000 平方英尺，而房间数量的值则是 0-5，以两个参数分别为横纵坐标，绘制代价函数的等高线图能，看出图像会显得很扁，梯度下降算法需要非常多次的迭代才能收敛。

解决的方法是尝试将所有特征的尺度都缩放到一个大致相等的区间之间。如图：

### 学习率

学习率也考虑多测试几个值，Andrew推荐的序列是以3倍递增：

0.001，0.003，0.01，0.03，0.1……

### 多项式回归模型

线性模型有时候并不能满足我们的需要，这时候可以考虑别的曲线来拟合数据，比如二次方模型、三次方模型v





或者：





多项式回归模型的话，特征缩放非常有必要

### 正规方程

梯度下降算法的原理是找到损失函数的最低点；从数学上讲，这个点就是该点梯度为0的地方，而正规方程就是一步到位，求出损失函数梯度为0 的点。

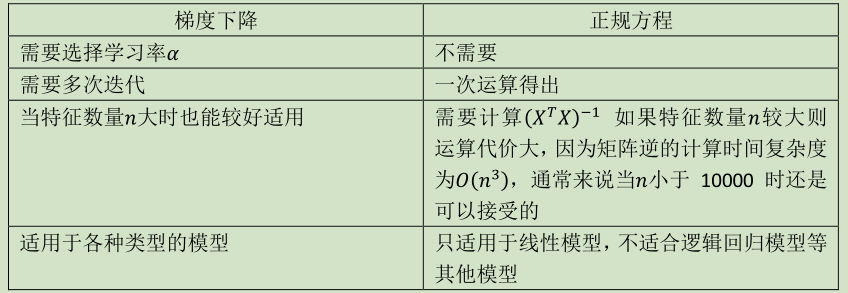
求解公式为：

θ = (XTX)-1XTy

矩阵不可逆通常有2种情况引起：

* 使用了线性相关的特征，比如同时使用m和feet
* 使用了太多特征，比如想用10个数据算出100个特征。

#### 比较



#### 结论

只要特征变量的数目并不大，标准方程是一个很好的计算参数的替代方法。

具体地说，只要特征变量数量小于一万，我通常使用标准方程法，而不使用梯度下降法。

随着我们要讲的学习算法越来越复杂，例如，当我们讲到分类算法，像逻辑回归算法只要特征变量的数目并不大，标准方程是一个很好的计算参数𝜄的替代方法。对于那些更复杂的学习算法，

我们将不得不仍然使用梯度下降法。

正规方程的 python 实现：

import numpy as np

def normalEqn(X, y):

theta = np.linalg.inv([X.T@X)@X.T@y](mailto:X.T@X)@X.T@y) #X.T@X 等价于 X.T.dot(X)

return theta

## 五、逻辑回归

### 假说表示

逻辑回归模型的假设是





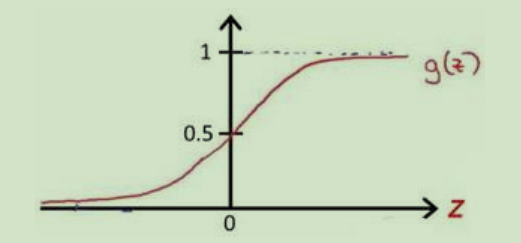
Python代码实现为

import numpy as np

def sigmoid(z):

return 1 / (1 + np.exp(-z))

函数图像



ℎ𝜃 (x)的作用是，对于给定的输入变量，根据选择的参数计算输出变量=1 的可能性

（estimated probablity）即ℎ𝜃 (x) = P(y = 1|x: 𝜃)

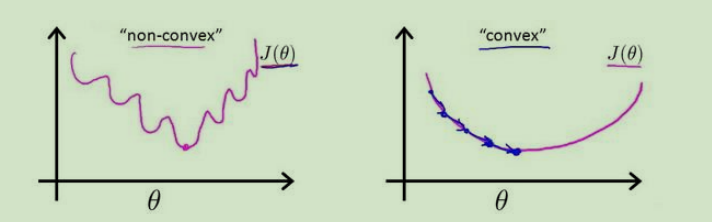
例如，如果对于给定的𝑦，通过已经确定的参数计算得出ℎ 𝜃 (x) = 0.7，则表示有 70%的

几率𝑧为正向类，相应地𝑧为负向类的几率为 1-0.7=0.3

### 代价函数

对于线性回归模型，我们定义的代价函数是所有模型误差的平方和。理论上来说，我们也可以对逻辑回归模型沿用这个定义，但是问题在于，当我们将

带入到这样定义了的代价函数中时，我们得到的代价函数将是一个非凸函数（non-convexfunction）



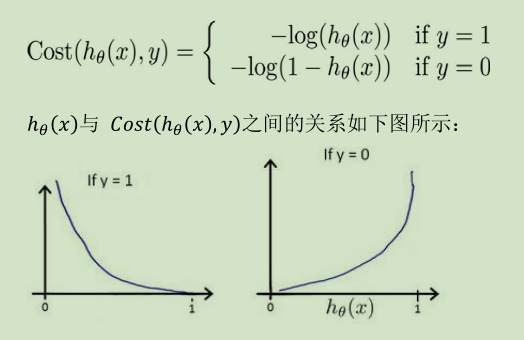
这意味着我们的代价函数有许多局部最小值，这将影响梯度下降算法寻找全局最小值。

最小平方函数不具有单调性，而对数函数具有单调性，能得到凸函数的代价函数，这就是代价函数采取对数函数而不是最小平方法的原因。

重新定义逻辑回归的代价函数为



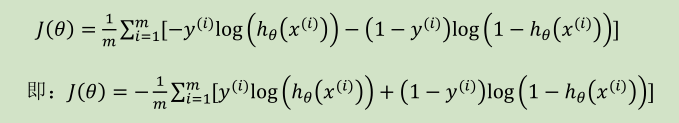
其中



代价函数简化之后如下



带入代价函数中



Python代码实现

import numpy as np

def cost(theta, X, y):

theta = np.matrix(theta)

X = np.matrix(X)

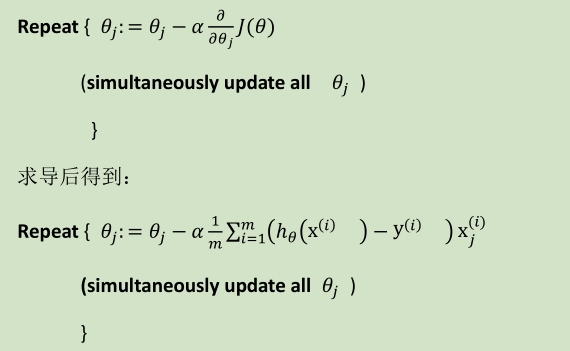
y = np.matrix(y)

first = np.multiply(-y, np.log(sigmoid(X\* theta.T)))

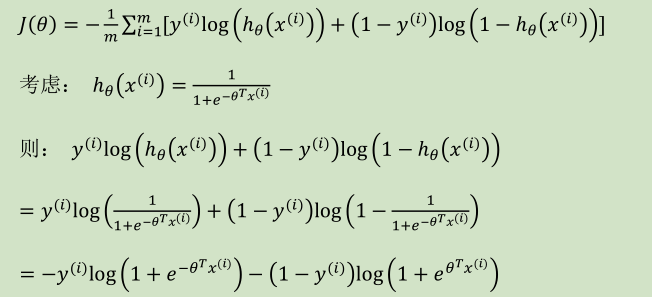
second = np.multiply((1 - y), np.log(1 - sigmoid(X\* theta.T)))

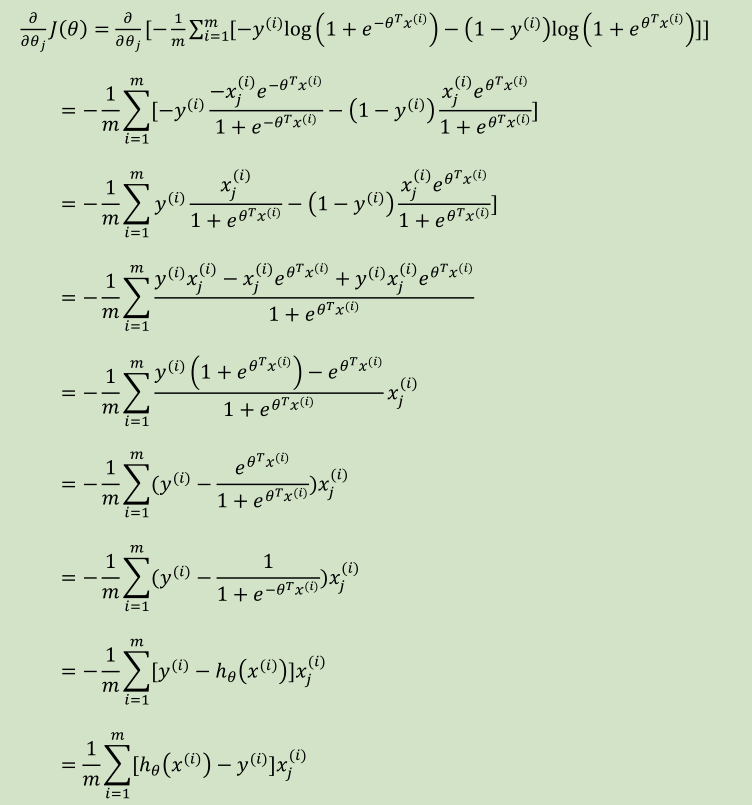
return np.sum(first - second) / (len(X))

梯度下降算法



#### 代价函数推导





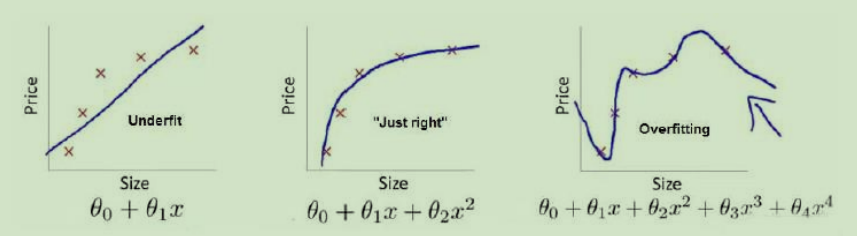
虽然得到的梯度下降算法表面上与线性回归是一样的，但这里的



所以，实际上是不一样的。

此外，进行特征缩放是很有必要的

## 六、正则化



出现了过拟合的问题，我们会采取两种办法：

* 丢弃无用特征， 或者使用模型选择的算法来改善。
* 正则化，保留所有参数，但是减小参数的大小。如第三个overfitting的图，减小所有多项式系数，模型表现好很多。

### 代价函数

图三的模型是，正是高次项导致了过拟合的产生，如果能让高次项的系数接近0的话，就可以很好拟合数了。所以目标就是一定程度上减小这些参数的值，这就是正则化的基本方法。

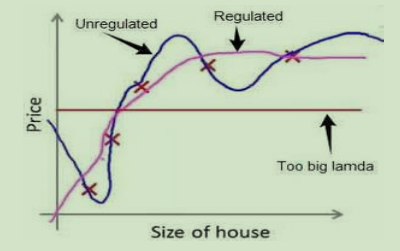
这里决定要减小后两个高阶项的系数，方法就是修改代价函数，对后两个系数进行一点惩罚。修改后的代价函数如下：



在我们不知道那些项是高阶项或者说无法指明要减小哪些参数的时候，可以对所有的参数进行惩罚：

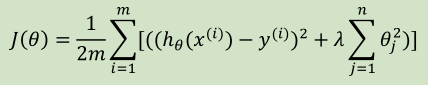


其中λ称为正则化参数。注意：θ0 是常数项，我们不对它进行惩罚。经过正则化的效果图可能如下所示

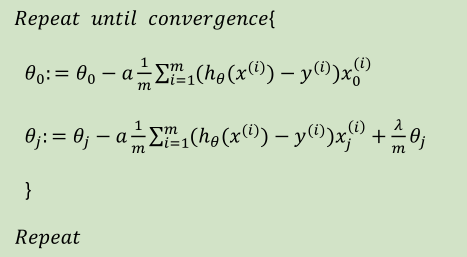


### 正则化线性回归

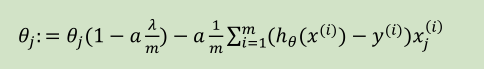
正则化线性回归的代价函数



算法

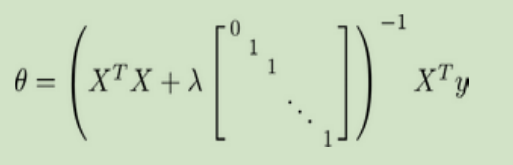


对第二式调整



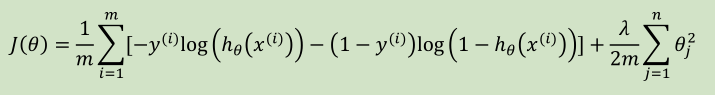
可以看出，正则化线性回归的梯度下降算法的变化在于， 每次都在原有更新规则下令θ减小一个额外的值。

### 正规方程正则化



图中矩阵尺寸为(n+1)\*(n+1)

### 逻辑回归模型正则化



Python代码

import numpy as np

def costReg(theta, X, y, learningRate):

theta = np.matrix(theta)

X = np.matrix(X)

y = np.matrix(y)

first = np.multiply(-y, np.log(sigmoid(X\*theta.T)))

second = np.multiply((1 - y), np.log(1 - sigmoid(X\*theta.T)))

reg = (learningRate / (2 \* len(X))\* np.sum(np.power(theta[:,1:the

ta.shape[1]],2))

return np.sum(first - second) / (len(X)) + reg