**Spark mllib 分析报告-Gradient Descent**

一．预备知识：Gradient Descent

回归与梯度下降：

回归在数学上来说是给定一个点集，能够用一条曲线去拟合之，如果这个曲线是一条直线，那就被称为线性回归，如果曲线是一条二次曲线，就被称为二次回归，回归还有很多的变种，如locally weighted回归，logistic回归，等等，这个将在后面去讲。

用一个很简单的例子来说明回归，这个例子来自很多的地方，也在很多的open source的软件中看到，比如说weka。大概就是，做一个房屋价值的评估系统，一个房屋的价值来自很多地方，比如说面积、房间的数量（几室几厅）、地 段、朝向等等，这些影响房屋价值的变量被称为特征(feature)，feature在机器学习中是一个很重要的概念，有很多的论文专门探讨这个东西。在 此处，为了简单，假设我们的房屋就是一个变量影响的，就是房屋的面积。

假设有一个房屋销售的数据如下：

面积(m^2)  销售价钱（万元）

123            250

150            320

87              160

102            220

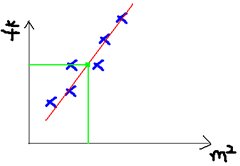
…               …

这个表类似于帝都5环左右的房屋价钱，我们可以做出一个图，x轴是房屋的面积。y轴是房屋的售价，如下：

[](http://images.cnblogs.com/cnblogs_com/LeftNotEasy/WindowsLiveWriter/1_1270E/image_2.png)

如果来了一个新的面积，假设在销售价钱的记录中没有的，我们怎么办呢？

我们可以用一条曲线去尽量准的拟合这些数据，然后如果有新的输入过来，我们可以在将曲线上这个点对应的值返回。如果用一条直线去拟合，可能是下面的样子：

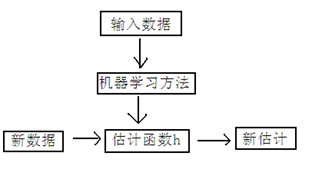
[](http://images.cnblogs.com/cnblogs_com/LeftNotEasy/WindowsLiveWriter/1_1270E/image_6.png)

绿色的点就是我们想要预测的点。

首先给出一些概念和常用的符号，在不同的机器学习书籍中可能有一定的差别。

房屋销售记录表 - 训练集(training set)或者训练数据(training data), 是我们流程中的输入数据，一般称为x房屋销售价钱 - 输出数据，一般称为y拟合的函数（或者称为假设或者模型），一般写做 y = h(x),训练数据的条目数(#training set), 一条训练数据是由一对输入数据和输出数据组成的。输入数据的维度(特征的个数，#features)，n

下面是一个典型的机器学习的过程，首先给出一个输入数据，我们的算法会通过一系列的过程得到一个估计的函数，这个函数有能力对没有见过的新数据给出一个新的估计，也被称为构建一个模型。就如同上面的线性回归函数。

[](http://images.cnblogs.com/cnblogs_com/LeftNotEasy/WindowsLiveWriter/1_1270E/image_8.png)

我们用X1，X2..Xn 去描述feature里面的分量，比如x1=房间的面积，x2=房间的朝向，等等，我们可以做出一个估计函数：

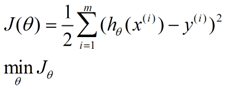
[image](http://images.cnblogs.com/cnblogs_com/LeftNotEasy/WindowsLiveWriter/1_1270E/image_12.png)

θ在这儿称为参数，在这儿的意思是调整feature中每个分量的影响力，就是到底是房屋的面积更重要还是房屋的地段更重要。为了如果我们令X0 = 1，就可以用向量的方式来表示了：

[image](http://images.cnblogs.com/cnblogs_com/LeftNotEasy/WindowsLiveWriter/1_1270E/image_14.png)

我们程序也需要一个机制去评估我们θ是否比较好，所以说需要对我们做出的h函数进行评估，一般这个函数称为损失函数（loss function）或者错误函数(error function)，描述h函数不好的程度，在下面，我们称这个函数为J函数

在这儿我们可以做出下面的一个错误函数：

[](http://images.cnblogs.com/cnblogs_com/LeftNotEasy/WindowsLiveWriter/1_1270E/image_16.png)

这个错误估计函数是去对x(i)的估计值与真实值y(i)差的平方和作为错误估计函数，前面乘上的1/2是为了在求导的时候，这个系数就不见了。

如何调整θ以使得J(θ)取得最小值有很多方法：

最小二乘法(min square)，是一种完全是数学描述的方法。

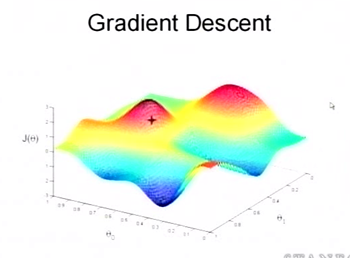
梯度下降法。

梯度下降法是按下面的流程进行的：

1）首先对θ赋值，这个值可以是随机的，也可以让θ是一个全零的向量。

2）改变θ的值，使得J(θ)按梯度下降的方向进行减少。

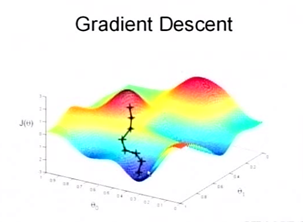
为了更清楚，给出下面的图：

[](http://images.cnblogs.com/cnblogs_com/LeftNotEasy/WindowsLiveWriter/1_1270E/image_20.png)

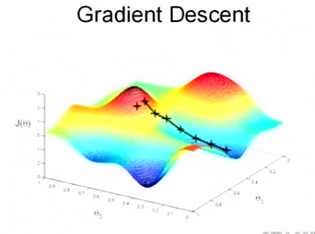
这是一个表示参数θ与误差函数J(θ)的关系图，红色的部分是表示J(θ)有着比较高的取值，我们需要的是，能够让J(θ)的值尽量的低。也就是深蓝色的部分。θ0，θ1表示θ向量的两个维度。

在上面提到梯度下降法的第一步是给θ给一个初值，假设随机给的初值是在图上的十字点。

然后我们将θ按照梯度下降的方向进行调整，就会使得J(θ)往更低的方向进行变化，如图所示，算法的结束将是在θ下降到无法继续下降为止。

[](http://images.cnblogs.com/cnblogs_com/LeftNotEasy/WindowsLiveWriter/1_1270E/image_22.png)

当然，可能梯度下降的最终点并非是全局最小点，可能是一个局部最小点，可能是下面的情况：

[](http://images.cnblogs.com/cnblogs_com/LeftNotEasy/WindowsLiveWriter/1_1270E/image_24.png)

上面这张图就是描述的一个局部最小点，这是我们重新选择了一个初始点得到的，看来我们这个算法将会在很大的程度上被初始点的选择影响而陷入局部最小点

下面我将用一个例子描述一下梯度减少的过程，对于我们的函数J(θ)求偏导J：（求导的过程如果不明白，可以温习一下微积分）

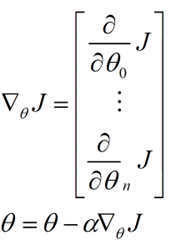
[image](http://images.cnblogs.com/cnblogs_com/LeftNotEasy/WindowsLiveWriter/1_1270E/image_26.png)

下面是更新的过程，也就是θi会向着梯度最小的方向进行减少。θi表示更新之前的值，-后面的部分表示按梯度方向减少的量，α表示步长，也就是每次按照梯度减少的方向变化多少。

[image](http://images.cnblogs.com/cnblogs_com/LeftNotEasy/WindowsLiveWriter/1_1270E/image_30.png)

一个很重要的地方值得注意的是，梯度是有方向的，对于一个向量θ，每一维分量θi都可以求出一个梯度的方向，我们就可以找到一个整体的方向，在变化的时候，我们就朝着下降最多的方向进行变化就可以达到一个最小点，不管它是局部的还是全局的。

用更简单的数学语言进行描述步骤2）是这样的：

[](http://images.cnblogs.com/cnblogs_com/LeftNotEasy/WindowsLiveWriter/1_1270E/image_32.png)

倒三角形表示梯度，按这种方式来表示，θi就不见了，看看用好向量和矩阵，真的会大大的简化数学的描述啊。

Tips:迭代过程中，初始给定θ，对于每个样本点带入下式

[image](http://images.cnblogs.com/cnblogs_com/LeftNotEasy/WindowsLiveWriter/1_1270E/image_30.png)

[image](http://images.cnblogs.com/cnblogs_com/LeftNotEasy/WindowsLiveWriter/1_1270E/image_12.png)

迭代更新 θ

1. 算法并行化

梯度下降本质上来讲是可以并行计算的，集群中每个Executor 持有一部分训练数据，各个Executor 计算完梯度，统一汇总到Driver 进行Gradient 的更新。

算法流程：Spark 实现

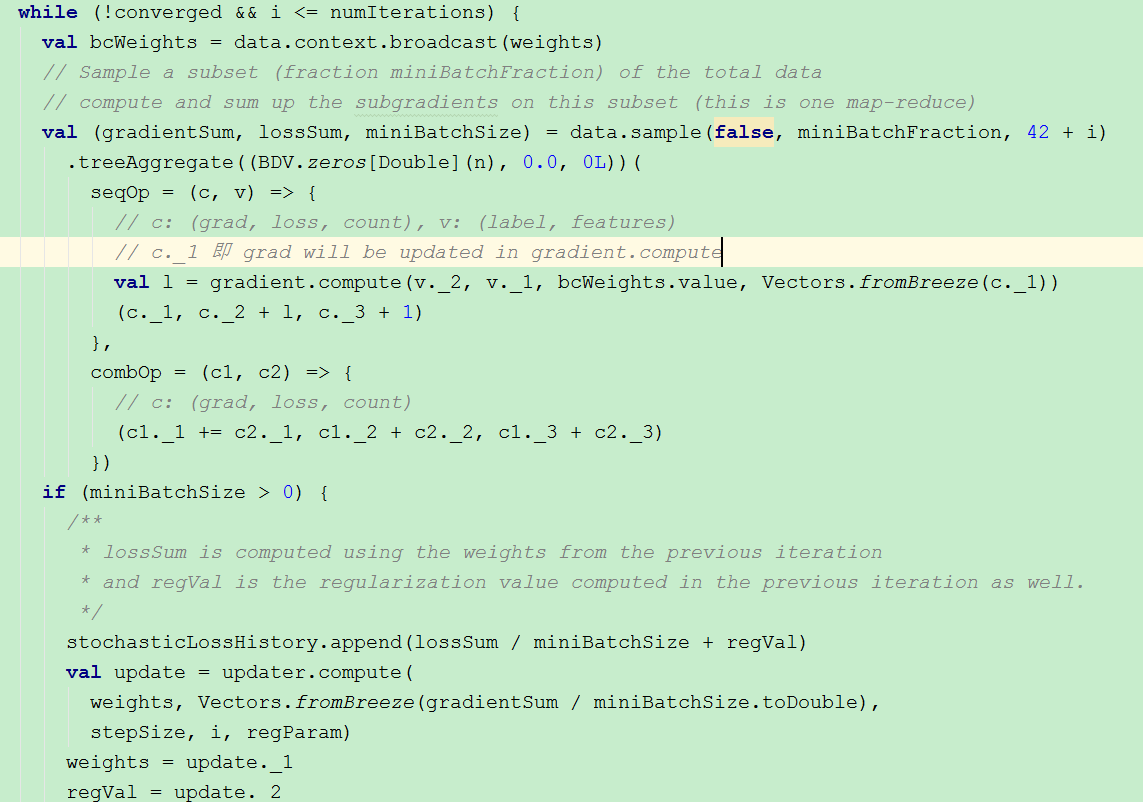
* 初始化初始的梯度，可以是0。
* 迭代下面过程：
* Driver Broadcast：Weight
* Executor 计算 local Gradient
* 使用Treeaggregate 对 executor’ local gradient进行聚合操作得到：Gradient Sum

Weight = Weight - α\*Gradient

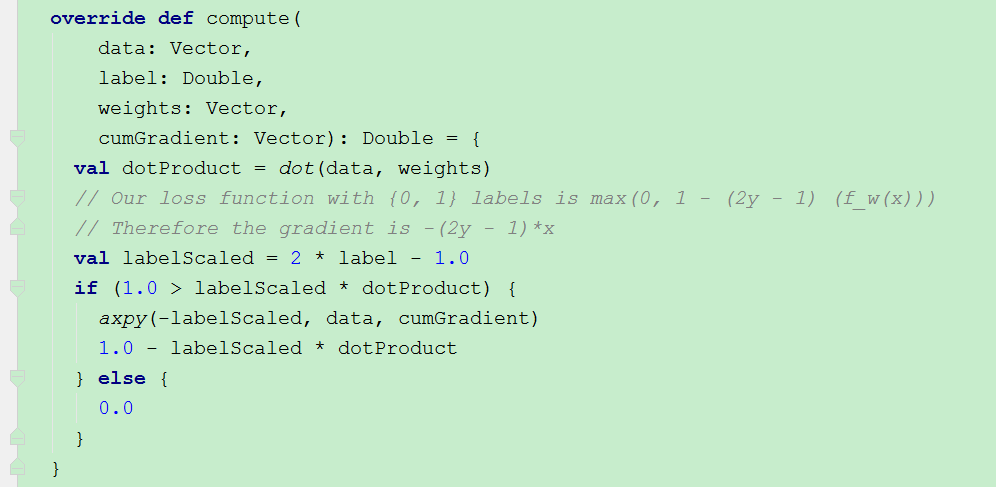
* 算法终止条件：算法达到迭代次数或者weight 收敛

Code 分析：

* Basic logic



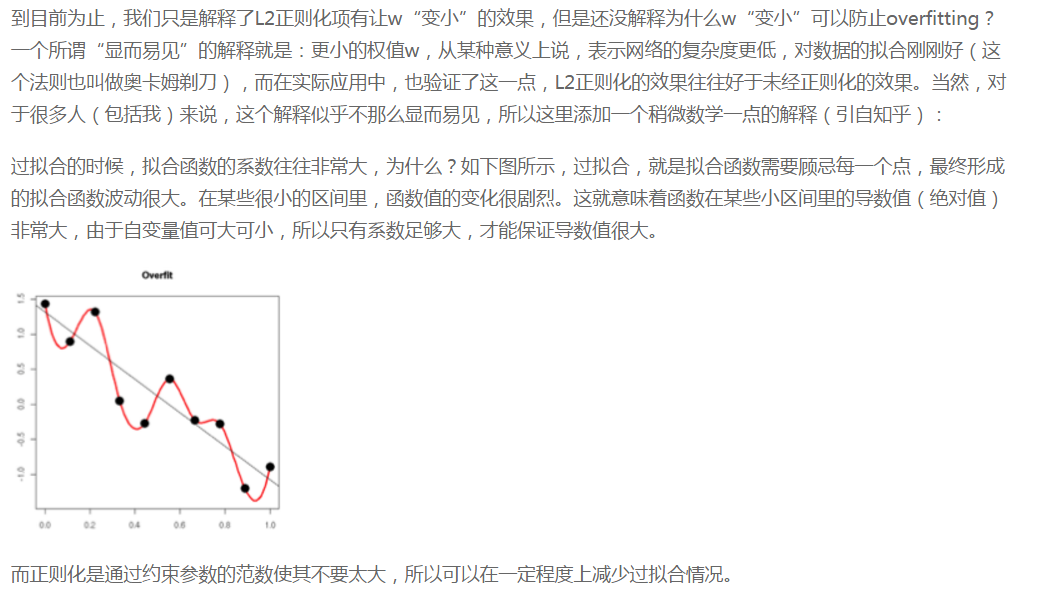
* Gradient compute:以HingeGradient 为例



axpy(a,x,y) // y += ax

1. 正则化

通常，加入L1/L2 正则化往往能够收到更好地效果，可以有效防止过拟合。具体解释见下面截图：

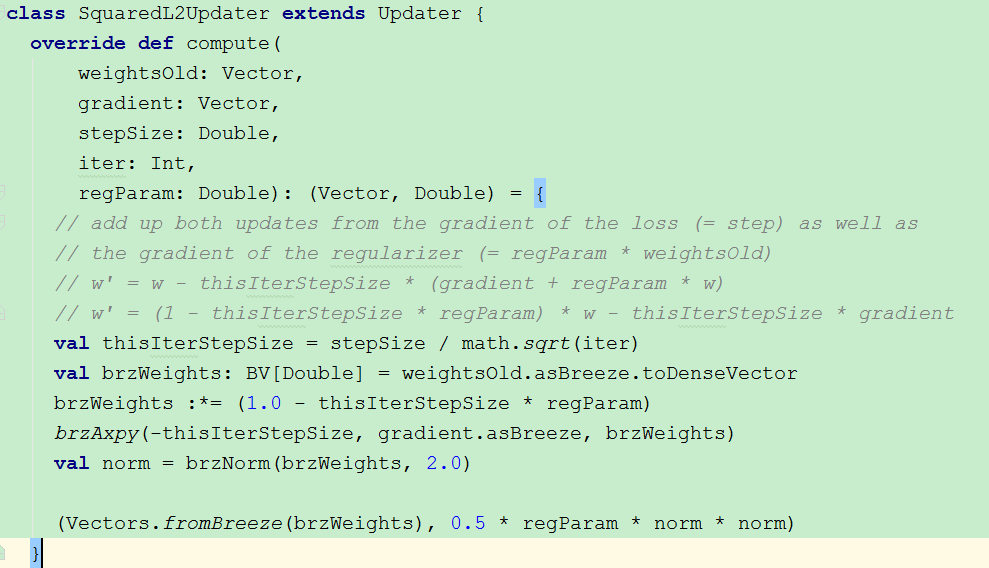


对于L2正则化 R(w) = 1/2 |w|^2 求导 =》 w,加入regParam 是为了权衡gradient 和L2正则化。

那么：Weight 更新公式就变成了：W2 = w1 - stepsize\*(gradient + regParam\*w)

化简 W2 = (1 - stepsize \* regparam) \* w - stepsize \* gradient

具体如下代码：



附录1：

Spark中目前实现了几种Gradient,本质区别是损失函数不同。

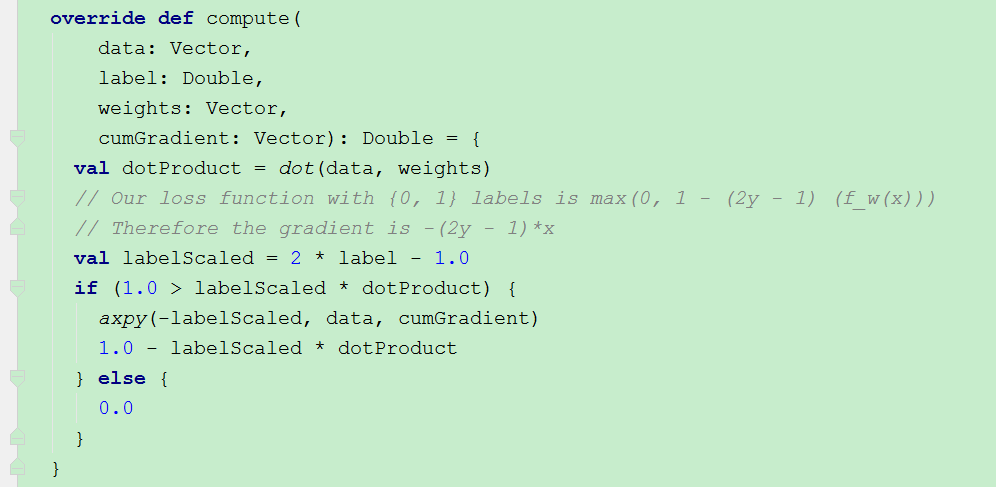
Logistic Gradient

Hinge Gradient

Least Squares Gradient

Logistic Gradient:应用于Logistic Regression

Hinge Gradient: can be used in SVM



Least Squares Gradient: can be used in linear regression

