一、线性回归

线性回归一般用来做连续值的预测，预测的结果为一个连续值。因训练时学习样本不仅要提供学习的特征向量X，而且还要提供样本的实际结果（标记label），所以它是一种有监督学习。其中 X={x0,x1,...,xn}X={x0,x1,...,xn}。

线性回归需要学习得到的是一个映射关系 f：X→yf：X→y，即当给定新的待预测样本时，我们可以通过这个映射关系得到一个测试样本 XX 的预测值 yy 。

在线性回归中，假定输入X和输出y之间具有线性相关的关系。

例如当特征向量 XX 中只有一个特征时，需要学习到的函数应该是一个一元线性函数 y=a+bxy=a+bx。

当情况复杂时，考虑 XX 存在n个特征的情形下，我们往往需要得到更多地系数。我们将 XX 到 yy 的映射函数记作函数hθ(X)hθ(X)：

hθ(X)=∑i=0nθixi=θTX

hθ(X)=∑i=0nθixi=θTX

其中，为了在映射函数hΘ(X)中保留常数项，令 x0x0 为1，所以特征向量X={1,x1,x2,...,xn}X={1,x1,x2,...,xn}，特征系数向量 θ={θ0,θ1,θ2,...,θn}θ={θ0,θ1,θ2,...,θn} 。

当给定一个训练集数据的情况，可以通过某个算法，学习出来一个线性的映射函数 hθ(X)hθ(X) 来求得预测值 yy。

二、损失函数

在需要通过学习得到的映射函数 hθ(X)hθ(X) 中，需要通过训练集得到特征系数向量 θ={θ0,θ1,θ2,...,θn}θ={θ0,θ1,θ2,...,θn} 。

根据特征向量系数 θθ，可有损失函数 J(θ)J(θ) 如下 :

J(θ)=12m∑i=1m(hθ(X(i))−y(i))2

J(θ)=12m∑i=1m(hθ(X(i))−y(i))2

其中 hθ(X)hθ(X) 为需要学习到的函数，mm 为训练集样本的个数，XiXi 表示训练集中第 ii 个样本的特征向量，yiyi 表示第 ii 个样本中的标签。

为了得到预测值 hθ(Xi)hθ(Xi) 和 yiyi 的绝对值，在公式上使用了平方数。为了平均每个样本的损失，在公式上对损失和进行除以 mm 操作，，再除以 22 是为了之后的求导计算。

三、梯度下降算法求解

批量随机梯度下降BGD

在上面，找到了一个特征系数向量 θθ 好坏的损失函数 J(θ)J(θ)。为了迎合这样的评判标准得到较好的 θθ，需对损失函数值进行最小化，即让损失函数在样本中的损失最小。

对于损失函数 J(θ)J(θ)，可以发现 θθ 是一个关于 \theta 的凸函数。

梯度下降就是一个不断地最小化损失函数的过程。从图像上来看，先初始化 θiθi 为某个值，然后让 θiθi 沿着J J(θ)J(θ) 在 θiθi 的偏导方向不断地走，直达走到底部收敛为止，最后就可以得到 J(θ)J(θ) 最小时的那个 θiθi 的值。

这个不断迭代的过程犹如一个不断下山的过程，我们可以得到图中关于 θiθi 的迭代函数，其中 αα 为每次下山的步长。

当 θiθi 小于最低处的值的时，对其的偏导为负，在迭代过程中， θiθi 不断地增大以逼近最低处的值；当 θiθi 大于最低处的值时，对其的偏导为正， θiθi 会不断地做减法以逼近最低处的值。所以当步长 αα 较小时， θiθi 会收敛于最低处的值。通常，我们将 αα 叫做学习率 （learning rate）。

对于特征向量系数 θθ 中的每个元素 θiθi，可以通过上面的迭代公式，它们都会往各自偏导的方向“下山”，“下山”的方向（梯度）为偏导的方向，按照这样的下山方向，下山的速度会更快一点。梯度下降算法就是这么的一个“下山”的过程。

对下山迭代公式进行展开，可得到具体求得 θiθi 的公式：

其中，x(i)0x0(i) 为第 ii 个训练样本的第 00 个特征值，为保持 hθ(X)hθ(X) 的常数项，通常令 x(i)0x0(i) 为1。hθ(X)hθ(X) 为在迭代过程中通过 θθ 得到的预测函数，它的表达式会跟随着 θθ 的变化而不断地变化。

当学习率 αα 很小时，我们往往需要迭代更多次才可以“下山”达到山底，θθ 收敛很慢；当学习率a很大时，“下山”步子迈大后，往往会在山底的两边跳跃，可能无法到达山底，θθ 会出现震荡，导致无法收敛。

一般情况下， αα 可设为0.01，0.005等，也可以在迭代的过程中不断自适应地修改 αα 的值，例如可以在迭代过程中不断地减小 αα 的值。

代码实现：

#include <stdio.h>

#include <stdlib.h>

int main(int argc, char \*argv[])

{

double matrix[4][2]={{1,1},{4,1},{5,1},{8,1}}; //样本

double result[4]={5,11,13,19.5}; //期望值

double err\_sum[4] = {0,0,0,0}; //各个样本的误差

double theta[2] = {1,6}; //Θ,初始值随机

double err\_square\_total = 0.0; //方差和

double learning\_rate = 0.01; //学习率

int ite\_num; //迭代次数

for(ite\_num = 0; ite\_num <= 10000; ite\_num++)

{

int i,j,k;

err\_square\_total = 0.0;

for(i = 0; i < 4; i++)

{

double h = 0;

for(j = 0; j < 2; j++)

h += theta[j]\*matrix[i][j];

err\_sum[i] = result[i] - h;

err\_square\_total += 0.5\*err\_sum[i]\*err\_sum[i];

}

if(err\_square\_total < 0.05) //0.05表示精度

break;

for(j = 0; j < 2; j++)

{

double sum = 0;

for(k = 0; k < 4; k++) //所有样本都参与计算

sum += err\_sum[k]\*matrix[k][j];

theta[j] = theta[j] + learning\_rate\*sum; //根据上面的公式计算新的Θ

}

}

printf(" @@@ Finish, ite\_number:%d\n, err\_square\_total:%lf, theta[0]:%lf, theta[1]:%lf\n", ite\_num, err\_square\_total, theta[0], theta[1]);

return 0;

}

