# Machine Learning---LMS 算法

## 引言

简单的感知器学习算法（《Machine Learning---感知器学习算法》）会将真个集合正确分类后，才会停止，显然当测试数据多的时候，这种算法会变得迟钝。所以这里，引入一个理念，最小均方算法（Least Mean Square）。

## LMS算法基本介绍

### 1.历史

LMS算法首先由Bernard Widrow和Marcian E. Hoff提出，被用于分类计算。大大降低了分类算法的复杂度。LMS算法是一种梯度下降法(Gradient Descent)。

对于LMS的数学证明，这里暂时不做介绍。

所以下面提到的公式，也只做简单性说明，请见谅。

### 2.均方差

均方差（Mean Square Error）这个概念我就用下面这个公式进行介绍。

公式（1）

上面的公式1中的R表示正确的预期结果，C表示当前计算结果。这个便是LMS算法中终止算法的核心公式。

对于如何得到“当前计算结果C”，按照下面这个公式进行计算

公式（2）

对于该公式，笔者在《Machine Learning---感知器学习算法》中有介绍。这里就只做简单解释：i表示输入值，W表示输入端所对应的权值，对这两个值进行乘法运算后，并求和。对于求和的结果可以进行一定处理，比如大于0的O便为1；否则就为-1。

### 3.权值调整公式

用于调整输入端的权值。

公式（3）

在算法运行时，不断利用公式2进行输入端的权值调整，使权值越来越接近正确值。其中w便是输入端所对应的权值，I便是输入值，便是学习参数，一般为小于1的正数。

### 4.算法流程

下面介绍一下LMS算法的基本流程。

1. 初始化工作，为各个输入端的权值覆上随机初始值；
2. 随机挑选一组训练数据，进行计算得出计算结构C；
3. 利用公式3对每一个输入端的权值进行调整；
4. 利用公式1计算出均方差MSE；
5. 对均方差进行判断，如果大于某一个给定值，回到步骤2，继续算法；如果小于给定值，就输出正确权值，并结束算法。

## 算法实现

以下就给出一段LMS算法的代码。

const unsigned int nTests = 4;

const unsigned int nInputs = 2;

const double rho = 0.005;

struct lms\_testdata

{

double inputs[nInputs];

double output;

};

double compute\_output(const double \* inputs,double \* weights)

{

double sum = 0.0;

for (int i = 0 ; i < nInputs; ++i)

{

sum += weights[i]\*inputs[i];

}

//bias

sum += weights[nInputs]\*1.0;

return sum;

}

//计算均方差

double caculate\_mse(const lms\_testdata \* testdata,double \* weights)

{

double sum = 0.0;

for (int i = 0 ; i < nTests ; ++i)

{

sum += pow(testdata[i].output - compute\_output(testdata[i].inputs,weights),2);

}

return sum/(double)nTests;

}

//对计算所得值,进行分类

int classify\_output(double output)

{

if(output > 0.0)

return 1;

else

return -1;

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

lms\_testdata testdata[nTests] = {

{-1.0,-1.0, -1.0},

{-1.0, 1.0, -1.0},

{ 1.0,-1.0, -1.0},

{ 1.0, 1.0, 1.0}

};

double weights[nInputs + 1] = {0.0};

while(caculate\_mse(testdata,weights) > 0.26)//计算均方差,如果大于给定值,算法继续

{

int iTest = rand()%nTests;//随机选择一组数据

double output = compute\_output(testdata[iTest].inputs,weights);

double err = testdata[iTest].output - output;

//调整输入端的权值

for (int i = 0 ; i < nInputs ; ++i)

{

weights[i] = weights[i] + rho \* err \* testdata[iTest].inputs[i];

}

weights[nInputs] = weights[nInputs] + rho \* err;

cout<<"mse:"<<caculate\_mse(testdata,weights)<<endl;

}

for(int w = 0 ; w < nInputs + 1 ; ++w)

{

cout<<"weight"<<w<<":"<<weights[w]<<endl;

}

cout<<"\n";

for (int i = 0 ;i < nTests ; ++i)

{

cout<<"right result：êo" <<testdata[i].output<<"\t";

cout<<"caculate result:" << classify\_output(compute\_output(testdata[i].inputs,weights))<<endl;

}

//

char temp ;

cin>>temp;

return 0;

}

## 总结

LMS算法的数学方面的说明比较麻烦,所以笔者想之后单独写一篇。

如果有兴趣的可以去看维基百科关于LMS算法的说明，这篇暂时只做编程上的简单介绍。

由于笔者不是专门研究人工智能方面，所以在写这些文章的时候，肯定会有一些错误，也请谅解，上面介绍中有什么错误或者不当地方，敬请指出，不甚欢迎。

如果有兴趣的可以留言，一起交流一下算法学习的心得。

声明：本文章是笔者整理资料所得原创文章，如转载需注明出处，谢谢。