人工智能第六次实验报告

多元线性回归

多元线性回归概念：

  在回归分析中，如果有两个或两个以上的自变量，就称为多元回归。社会经济现象的变化往往受到多个因素的影响，例如，家庭消费支出，除了受家庭可支配收入的影响外，还受诸如家庭所有的财富、物价水平、金融机构存款利息等多种因素的影响。因此，一般要进行多元回归分析，我们把包括两个或两个以上自变量的回归称为多元线性回归。一元线性回归是一个主要影响因素作为自变量来解释因变量的变化。事实上，一种现象常常是与多个因素相联系的，由多个自变量的最优组合共同来预测或估计因变量，比只用一个自变量进行预测或估计更有效，更符合实际。因此多元线性回归比一元线性回归的实用意义更大。

  多元线性回归与一元线性回归类似，可以用最小二乘法估计模型参数，也需对模型及模型参数进行统计检验。

  选择合适的自变量是正确进行多元回归预测的前提之一，多元回归模型自变量的选择可以利用变量之间的相关矩阵来解决。

多元线性回归服从正态分布：

多元线性回归要求服从高斯分布也就是正态分布。

代码实现：

#include <iostream>

#include<fstream>

#include<cmath>

const int sample = 28;//样本数 fish样本为44 house\_price样本为28

const int feature = 13;//特征值数 fish样本为4 house\_price样本为13

const int scaling = 10;//特征缩放大小 fish样本为100 house\_price样本为10

double a =0.001;//学习率 fish样本为0.0002 house\_price样本为0.001

#define path "D:\house\_price.txt"

using namespace std;

int main()

{

double x[sample][feature] = { 0 };

ifstream infile;

infile.open(path);

int i, j;

if (!infile)

{

cout << "cannot open file" << endl;

return -1;

}

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

{

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

{

infile >> x[i][j];

if (j == feature-1)

x[i][j] = x[i][j] /scaling;

if (j == 0)

x[i][j] = 1;

cout << x[i][j] << " ";

}

cout << endl;

}

infile.close();

double J=0;//代价函数

double Y[sample] = { 0 };//预测函数

double temp[feature - 1] = { 0 }, t = 1, p[feature] = { 1 };//置换变量

double w[feature-1] = {0};//系数数组

double hype;//验证值

double A=0;//验证值与真实值的平均差异

int cnt = 0;//计数变量

double sum = 0, s[feature - 1] = { 0 };//求和变量

cout << endl;

cout <<"输出验证结果："<< endl;

for (int m = 0; m < sample; m++)

{

for (int n = 0; n < feature; n++)

{

p[n] = x[m][n];

x[m][n] = 0;

//抽取一个为测试集

while (fabs(J - t) > 0.001)//当代价函数变化极小时退出循环

{

t = J;//等于上一个代价函数的值

J = 0;//代价函数归零

for (i = 0; i < sample; i++)//求每个样本的预测函数

{

sum = 0;

for (j = 1; j < feature - 1; j++)//除去序值

{

sum += w[j] \* x[i][j];

}

Y[i] = sum;

}

for (j = 0; j < feature - 1; j++)//梯度下降法

{

sum = 0;

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

{

sum += (1.0 / (sample))\*(Y[i] - x[i][feature - 1])\*x[i][j];

cnt++;

}

temp[j] = w[j] - a \* sum;

}

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

{

w[i] = temp[i];

}

for (i = 0; i < sample; i++)//求代价函数

{

for (j = 1; j < feature - 1; j++)

{

J += (1.0 / 2.0 / (double)(sample))\*pow((Y[i] - x[i][feature - 1]),2);

}

}

}

}

sum = 0;

for (i = 0; i < feature-1; i++)//开始使用留一法进行测试

{

x[m][i] = p[i];

sum += w[i] \* x[m][i];

}

hype = sum;

cout << "第" << m+1<<"个测试结果：" << endl;

cout << "验证值为" << hype << endl;

cout << "真实值为" << p[3] << endl;

cout << "验证值与真实值的差为" << fabs(p[3]-hype) << endl<<endl;

A += fabs(p[3] - hype);

//求系数均值

for (i = 0; i < feature - 1; i++)//求平均系数值

s[i]+= w[i];

}

cout << endl;

cout << "线性回归系数值：" << endl;

for (i = 0; i < feature - 1; i++)

{

w[i] =s[i] / sample;

cout << "第" << i + 1 << "个系数为" << w[i] << endl;

}

cout <<endl<< "代价函数的值为"<< J << endl;

cout <<endl<<"梯度下降法一共运行了"<< cnt<<"次"<<endl << endl;

cout << "验证值与真实值的平均差为" << A/(sample)<< endl<<endl;

cout << "该线性数据集学习率为" << a << endl;

cout << "该线性数据集样本数为" << sample << endl;

cout << "该线性数据集特征数为" << feature << endl;

cout<< "更换数据集请记得调试特征缩放以及以上参数!" << endl;

cout << endl;

system("pause");

return 0;

}

运行结果：