20220512-机器学习

1.学习内容

1.1机器学习

简单线性回归学习的实现

C++ 复制代码

```cpp
#include <stdlib.h>
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <regex>
#include <math.h>
//生成数据并写入文件

double randomData(int & x)
{
    double k = (-1) + 2 * rand() / double(RAND_MAX);
    double y=x * 10 + 4+k;
    return y;
}

void writeData(int num,std::string filename)
{
    std::ofstream myFile;
    myFile.open(filename);
    for (int i = 0; i < num; i++)
    {
        myFile << i << "\t" << randomData(i) << std::endl;
    }
    myFile.close();
    return;
}


//读取文件并存储数据

std::vector<std::vector<double>> readData(std::string filename)
{
    std::vector<double> temp_line;
    std::vector<std::vector<double>> myVec;
    std::string line;
    std::regex pat_regex("(\\d+(\\.\\d+)?)");
    std::ifstream fp(filename);
    if (!fp.is_open()) {
        std::cout << "could not open file" << std::endl;
        exit(-1);
    }

    while (std::getline(fp, line))
    {
```

```
        for (std::sregex_iterator it(line.begin(), line.end(), pat_regex), end_it; it != end_it; ++it)
        {
            temp_line.push_back(std::stod(it->str()));
        }
        myVec.push_back(temp_line);
        temp_line.clear();
    }
    return myVec;
}


//模型(公式)
double total_loss(double weight_, double bias_, std::vector<std::vector<double>> srcData)
{
    int num_x = srcData[0].size();
    int num_y = srcData.size() / num_x;
    double loss = 0.0;
    for (int i = 0; i < num_y; i++)
    {
        loss += 0.5 * std::pow(weight_ * srcData[i][0] + bias_ - srcData[i][1], 2)* (double(1) / num_y);
    }
    return loss;
}

std::vector<double> SGD(std::vector<std::vector<double>> srcData,double lr)
{
    std::vector<double> params;
    double weight= rand();
    double bias=rand();
    double delta_weight;
    double delta_bias;

    int numbers = srcData.size() / srcData[0].size();
    for (int epoch= 0; epoch < 10000; epoch++)
    {
        delta_weight = 0.0;
        delta_bias = 0.0;

        for (int k = 0; k < numbers;k++)
        {
            delta_weight += (weight * srcData[k][0] + bias - srcData[k][1]) * srcData[k][0];
            delta_bias += weight * srcData[k][0] + bias-srcData[k][1];
        }
```

```
        weight = weight - lr * delta_weight * (double(1) / numbers);
        bias = bias - lr * delta_bias * (double(1) / numbers);

    }
    double loss = total_loss(weight, bias, srcData);
    params.push_back(weight);
    params.push_back(bias);
    params.push_back(loss);
    return params;
}

//主程序
int main()
{
    srand((unsigned)time(NULL));
    writeData(20, "dataset.txt");
    std::vector<std::vector<double>> allData = readData("dataset.txt");
    std::vector<double> learnedParam=SGD(allData,0.03);
    std::cout << "weight: " << learnedParam[0] << ", " << "bias: " << learnedParam[1] <<", "<<"loss" <<learnedParam[2] << std::endl;
}
```

利用new声明二维和三维数组

C++ 复制代码

```cpp
#include <iostream>

int main()
{
    //二维数组
    int** matrix = new int*[4];
    for (int i = 0; i < 4; i++)
    {
        matrix[i] = new int[4];
        for (int j = 0; j < 4; j++)
        {
            matrix[i][j] = j;
        }
    }
    std::cout << matrix[0][2] << std::endl;

    for (int i = 0; i < 4; i++)
    {
        delete[] matrix[i];
    }
    delete[] matrix;

    //三维数组
    int*** matrix = new int** [4];
    for (int i = 0; i < 4; i++)
    {
        matrix[i] = new int* [4];
        for (int j = 0; j < 4; j++)
        {
            matrix[i][j] = new int[4];
            for (int k = 0; k < 4; k++)
            {
                matrix[i][j][k] = k;
            }
        }
    }

    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            delete[] matrix[i][j];
        }
        delete[] matrix[i];
    }
```

```
        delete[] matrix;
    }
```

2.结果描述

今天主要用实现了一个简单的线性回归学习示例，涉及的内容包括txt文件读写、vector嵌套、随机数生成、一元线性回归梯度计算等。总体而言不复杂，但也算是迈出了自己完整写一个C++示例的第一步。在编写的过程中主要遇到了以下问题：

- 随机数生成（解决方案：利用标准库中的rand函数；结合RAND_MAX生成位于0-1之间的随机数
- 文件读写（解决方案：ofstream用于文件写入，ifstream用于文件读取；利用正则表达式识别一个txt矩阵中的元素并写入vector容器。这部分代码主要参考的网上的示例，关于正则表达式还有待学习）
- 一元线性回归的梯度计算（解决方案：简单求导即可）

$$h_\theta(x) = \theta_0 + \theta_1 x$$

这个方程对应的图像是一条直线，称作回归线。其中，θ_1为回归线的斜率，θ_0为回归线的截距。

用梯度下降法来求解线性回归

$$
\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \begin{cases} j = 0 : \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \\ j = 1 : \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)} \end{cases}
$$

$$
\begin{aligned}
&\text{repeat until convergence } \{ \\
&\quad \theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \\
&\quad \theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)} \\
&\}
\end{aligned}
$$

在开展优化的过程中发现，学习率对于最终能否达到或逼近最优点具有十分重大的影响。当把学习率设定为0.3时，权重和bias一下子就暴增到系统无法处理的水平；而当设置为0.03时，则学习的结果往往较好。此外，参数的初始化对于最终训练集的loss也有影响。