算法描述

Logistic Regression形式为

$$P(Y=1)=\frac{1}{1+\exp w^T x} \tag{1}$$

$$P(Y=0)=\frac{\exp w^T x}{1+\exp w^T x} \tag{2}$$

则似然比为

$$\frac{P(Y=0)}{P(Y=1)}=\exp w^T x \tag{3}$$

对数似然比为

$$\ln \frac{P(Y=0)}{P(Y=1)}=w^T x \tag{4}$$

即对数似然比是关于x的线性函数，则似然函数为

$$L(w)=\prod_{i=1}^{m} P(Y=y^{(i)}|x^{(i)};w)=\prod_{i=1}^{m}(\frac{1}{1+\exp w^T x^{(i)}})^{y^{(i)}}(\frac{\exp w^T x^{(i)}}{1+\exp w^T x^{(i)}})^{1-y^{(i)}} \tag{5}$$

对数似然函数为

$$\ln L(w)=\sum_{i=1}^{m}\left(y^{(i)}\ln\frac{1}{1+\exp w^T x^{(i)}}+(1-y^{(i)})\ln\frac{\exp w^T x^{(i)}}{1+\exp w^T x^{(i)}}\right) \tag{6}$$

以上式作为损失函数，对其求导，得到

$$\frac{\partial \ln L(w)}{\partial w}=-\sum_{i=1}^{m}\left(y^{(i)}-\frac{1}{1+\exp w^T x^{(i)}}\right)x^{(i)}=-\sum_{i=1}^{m}\left(y^{(i)}-P(Y=1|x^{(i)};w)\right)x^{(i)} \tag{7}$$

即为损失函数的梯度，记 $p^{(i)} = P(Y=1|x^{(i)};w)$ 有

$$\frac{\partial \ln L(w)}{\partial w} = -\sum_{i=1}^{m}(y^{(i)} - p^{(i)})x^{(i)} \tag{8}$$

写成矩阵形式为

$$\frac{\partial \ln L(w)}{\partial w} = -X^T(y-p) \tag{9}$$

其中

$$X = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(m)} & x_2^{(m)} & \cdots & x_n^{(m)} \end{bmatrix},\ y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix},\ p = \begin{bmatrix} p^{(1)} \\ p^{(2)} \\ \vdots \\ p^{(m)} \end{bmatrix} \tag{10}$$

使用梯度下降法，每次迭代更新参数为

$$w := w - \frac{\partial \ln L(w)}{\partial w} \tag{11}$$

即

$$w := w + X^T(y-p) \tag{12}$$

代码实现

本次算法由`C++`实现。

`LogisticRegression` 类

首先，所有Logistic Regression相关运算封装在模板
类 `LogisticRegression<size_t Dimension>` 中，其中 `Dimension` 为特征维度，类的成员定义如下

```
template<size_t Dimension>
class LogisticRegression {
private:
    Vector<Dimension> weight {};
    double LearningRate;
}
```

`weight` 即为式11的参数w存储了 `LogisticRegression` 的唯一参数权重矢量， `LearningRate` 即为式11的参数α即学习率，是每次迭代更新参数时的步长。
`LogisticRegression` 类的构造函数定义如下

```
template<size_t Dimension>
class LogisticRegression {
public:
    explicit LogisticRegression(double learningRate) : LearningRate(learningRate) { }

    explicit LogisticRegression(double learningRate, double initWeight) :
        LearningRate(learningRate) {
        for (int i = 0; i < Dimension; ++i) {
            weight[i] = initWeight;
        }
    }

    explicit LogisticRegression(double learningRate,
                                std::initializer_list<double> initWeights) :
        LearningRate(learningRate) {

        if (initWeights.size() != Dimension) {
            throw std::invalid_argument("Invalid dimensions for initialization");
        }

        size_t index = 0;
        for (const auto& v : initWeights) {
            this->weight[index] = v;
            index++;
        }
    }
}
```

提供最基本的初始化学习率和各种方式初始化权重矢量的构造函数。
拟合是通过调用`fit()`函数实现的，其定义如下

```
template<size_t Dimension>
class LogisticRegression {
public:
    template<size_t DataNumber>
    void fit(const Matrix<DataNumber, Dimension + 1> &data, size_t fitTimes) {
        const auto X = data.template SubColumns<Dimension>(0);
        const auto r = data.SubColumn(Dimension);

        for (int i = 0; i < fitTimes; ++i) {
            fitOnce(X, r);
        }
    }
}
```

`data`为训练数据，`fitTimes`为迭代次数。其首先会将训练数据`data`分离为特征矩阵`X`和标签矢量`r`，然后调用`fitOnce()`函数进行`firTimes`次迭代，`fitOnce()`定义如下

```
template<size_t Dimension>
class LogisticRegression {
private:
    template<size_t DataNumber>
    inline void fitOnce(
            const Matrix<DataNumber, Dimension> &inputData,
            const Vector<DataNumber> &outputData) {
        auto p = possibilityPositive(inputData);
        auto g = gradient(inputData, outputData, p);
        weight -= g * LearningRate;
    }
}
```

该函数对应了式10的迭代公式，其先通过特征矩阵`X`计算出当前参数下各个样本为正例的概率矢量`p`，然后通过梯度函数`gradient()`计算出当前参数下的梯度矢量`g`，最后更新参数`weight`。
阳性概率计算函数`possibilityPositive()`函数定义如下

```
template<size_t Dimension>
class LogisticRegression {
public:
    inline double possibilityPositive(const Vector<Dimension> &x) {
        return possibilityPositive(weight, x);
    }

    static inline double possibilityPositive(const Vector<Dimension> weight, const Vector<Dimensio
        return 1 / (1 + std::exp((weight ^ T) * x) + intercept);
    }

private:
    template<size_t DataNumber>
    Vector<DataNumber> possibilityPositive(const Matrix<DataNumber, Dimension> &X) {
        Vector<DataNumber> result;
        for (int i = 0; i < DataNumber; i++) {
            auto x = X.SubRow(i) ^ T;
            result[i] = possibilityPositive(x);
        }
        return result;
    }
}
```

该函数对应了式1，其拥有三个重载，第一个重载计算了当前参数下特征矢量`x`为正例的概率；第二个重载计算了参数`weight`下特征矢量`x`为正例的概率；第三个重载计算了参数`weight`下特征矩阵`X`中各个样本为正例的概率矢量，前两个重载同时也是我们使用模型作预测所使用的函数，

梯度函数`gradient()`定义如下

```
template<size_t Dimension>
class LogisticRegression {
private:
    template<size_t DataNumber>
    inline Vector<Dimension> gradient(
            const Matrix<DataNumber, Dimension> &X,
            const Vector<DataNumber> &y,
            const Vector<DataNumber> &p) {
        return (X ^ T) * (y - p);
    }
}
```

该函数对应了式12，其计算了当前参数下的梯度矢量。

主程序

首先进行基本的头文件包含和命名空间声明

```
#include <iostream>

#include "Matrix.hpp"
#include "Vector.hpp"
#include "LogisticRegression.hpp"

using std::cout;
using std::endl;
using namespace Math;
```

包含了基本输出、矩阵、矢量和Logistic Regression类的头文件，并声明了`Math`命名空间。
程序的初始数据定义如下

```
const double DecisionThreshold = 0.5;   // 判决门限

const size_t DataNumber = 9;     // 原始数据数量
const size_t Dimension = 5;      // 数据维度

using RawDataType = const Matrix<DataNumber, Dimension + 1>;
RawDataType &getRawData() { // 原始样本数据
    static RawDataType RawData {
        { 4,  3.4,  100,  3,  10, 1 },
        { 6,  4.1,  210,  1,  8 , 1 },
        { 8,  6.7,  600,  2,  16, 0 },
        { 10, 8.5,  1600, 6,  11, 0 },
        { 5,  4.8,  150,  13, 12, 0 },
        { 18, 15.6, 120,  21, 20, 1 },
        { 2,  3.4,  80,   1,  10, 1 },
        { 12, 7.9,  600,  4,  11, 0 },
        { 16, 12,   780,  8,  8 , 0 },
    };
    return RawData;
}
```

`DecisionThreshold`为判决门限，当阳性概率大于该门限时，判定为阳性，否则判定为阴性。`DataNumber`为原始数据数量，`Dimension`为数据维度，`RawDataType`为原始数据类型，`getRawData()`为获取原始数据的函数。
同时定义函数`result()`以判决结果

```
inline std::string result(double possibility) {
    if (possibility >= DecisionThreshold) {
        return "Positive";
    } else {
        return "Negative";
    }
}
```

主函数中，进行了另一部分数据的定义

```
const Vector<Dimension> Sample10 { 9, 15,  800, 7,  16 };   // 待预测样本
const Vector<Dimension> Sample11 { 3, 4.2, 189, 11, 7  };   // 待预测样本

const size_t FitTimes = 1000000;    // 拟合次数
const double LearningRate = 0.1;    // 学习率
```

包括了两个待预测样本`Sample10`和`Sample11`，`FitTimes`为拟合次数，`LearningRate`为学习率。
接下来，初始化`LogisticRegression`类的实例，将初始化权重设置为全1

```
LogisticRegression<Dimension> lr(LearningRate, 1);
```

而后进行拟合

```
lr.fit<DataNumber>(getRawData(), FitTimes);
```

拟合完毕进行预测

```
auto possibility10 = lr.possibilityPositive(Sample10);  // 预测样本10
auto possibility11 = lr.possibilityPositive(Sample11);  // 预测样本11
```

最后打印权重矩阵和预测结果

```
// 打印结果
cout << "ResultWeight = " << "[" << (lr.getWeight() ^ T) << "]" << endl;  // 打印训练结果权重
cout << endl;

cout << "Sample10 = " << "[" << (Sample10 ^ T) << "]" << endl;
cout << "Possibility = " << possibility10 << ", " << result(possibility10) << endl;
cout << endl;

cout << "Sample11 = " << "[" << (Sample11 ^ T) << "]" << endl;
cout << "Possibility = " << possibility11 << ", " << result(possibility11) << endl;
cout << endl;
```

结果展示

程序打印初始数据如下

```
RawData: 9 * 5
4 3.4 100 3 10 1
6 4.1 210 1 8 1
8 6.7 600 2 16 0
10 8.5 1600 6 11 0
5 4.8 150 13 12 0
18 15.6 120 21 20 1
2 3.4 80 1 10 1
12 7.9 600 4 11 0
16 12 780 8 8 0

InitWeight = [1 1 1 1 1]
FitTimes = 1000000
LearningRate = 0.1
DecisionThreshold = 0.5
```

训练结果即预测如下

```
ResultWeight = [-463.22 -349.514 40.1316 386.046 -578.937]

Sample10 = [9 15 800 7 16]
Possibility = 0, Negative

Sample11 = [3 4.2 189 11 7]
Possibility = 0, Negative
```

预测两样本阳性概率均为0，即均为阴性。