**逻辑回归**

**import** java.io.\*;

**public** **class** LoadData {

//导入样本特征

**public** **static** **double**[][] Loadfeature(String filename) **throws** IOException{

File f = **new** File(filename);

FileInputStream fip = **new** FileInputStream(f);

// 构建FileInputStream对象

InputStreamReader reader = **new** InputStreamReader(fip,"UTF-8");

// 构建InputStreamReader对象

StringBuffer sb = **new** StringBuffer();

**while**(reader.ready()) {

sb.append((**char**) reader.read());

}

reader.close();

fip.close();

//将读入的数据流转换为字符串

String sb1 = sb.toString();

//按行将字符串分割,计算二维数组行数

String [] a = sb1.split("\n");

**int** n = a.length;

System.***out***.println("二维数组行数为:" + n);

//计算二维数组列数

String [] a0 = a[0].split("\t");

**int** m = a0.length;

System.***out***.println("二维数组列数为:" + m);

**double** [][] feature = **new** **double**[n][m];

**for** (**int** i = 0; i < n; i ++) {

String [] tmp = a[i].split("\t");

**for**(**int** j = 0; j < m; j ++) {

**if** (j == m-1) {

feature[i][j] = (**double**) 1;

}

**else** {

feature[i][j] = Double.*parseDouble*(tmp[j]);

}

}

}

**return** feature;

}

//导入样本标签

**public** **static** **double**[] LoadLabel(String filename) **throws** IOException{

File f = **new** File(filename);

FileInputStream fip = **new** FileInputStream(f);

// 构建FileInputStream对象

InputStreamReader reader = **new** InputStreamReader(fip,"UTF-8");

// 构建InputStreamReader对象,编码与写入相同

StringBuffer sb = **new** StringBuffer();

**while**(reader.ready()) {

sb.append((**char**) reader.read());

}

reader.close();

fip.close();

//将读入的数据流转换为字符串

String sb1 = sb.toString();

//按行将字符串分割,计算二维数组行数

String [] a = sb1.split("\n");

**int** n = a.length;

System.***out***.println("二维数组行数为:" + n);

//计算二维数组列数

String [] a0 = a[0].split("\t");

**int** m = a0.length;

System.***out***.println("二维数组列数为:" + m);

**double** [] Label = **new** **double**[n];

**for** (**int** i = 0; i < n; i ++) {

String [] tmp = a[i].split("\t");

Label[i] = Double.*parseDouble*(tmp[m-1]);

}

**return** Label;

}

}

**import** java.io.\*;

**public** **class** LRMain {

**public** **static** **void** main(String[] args) **throws** IOException{

// filename

String filename = "data.txt";

// 导入样本特征和标签

**double** [][] feature = LoadData.*Loadfeature*(filename);

**double** [] Label = LoadData.*LoadLabel*(filename);

// 参数设置

**int** samNum = feature.length;

**int** paraNum = feature[0].length;

**double** rate = 0.01;

**int** maxCycle = 1000;

// LR模型训练

LRtrainGradientDescent LR = **new** LRtrainGradientDescent(feature,Label,paraNum,rate,samNum,maxCycle);

**double** [] W = LR.Updata(feature, Label, maxCycle, rate);

//保存模型

String model\_path = "wrights.txt";

SaveModel.*savemodel*(model\_path, W);

//模型测试

}

}

**public** **class** LRtrainGradientDescent {

**int** paraNum; //权重参数的个数

**double** rate; //学习率

**int** samNum; //样本个数

**double** [][] feature; //样本特征矩阵

**double** [] Label;//样本标签

**int** maxCycle; //最大迭代次数

**public** LRtrainGradientDescent(**double** [][] feature, **double** [] Label, **int** paraNum,**double** rate, **int** samNum,**int** maxCycle) {

**this**.feature = feature;

**this**.Label = Label;

**this**.maxCycle = maxCycle;

**this**.paraNum = paraNum;

**this**.rate = rate;

**this**.samNum = samNum;

}

// 权值矩阵初始化

**public** **double** [] ParaInitialize(**int** paraNum) {

**double** [] W = **new** **double**[paraNum];

**for** (**int** i = 0; i < paraNum; i ++) {

W[i] = 1.0;

}

**return** W;

}

//计算每次迭代后的预测误差

**public** **double** [] PreVal(**int** samNum,**int** paraNum, **double** [][] feature,**double** [] W) {

**double** [] Preval = **new** **double**[samNum];

**for** (**int** i = 0; i< samNum; i ++) {

**double** tmp = 0;

**for**(**int** j = 0; j < paraNum; j ++) {

tmp += feature[i][j] \* W[j];

}

Preval[i] = Sigmoid.*sigmoid*(tmp);

}

**return** Preval;

}

//计算误差率

**public** **double** error\_rate(**int** samNum, **double** [] Label, **double** [] Preval) {

**double** sum\_err = 0.0;

**for**(**int** i = 0; i < samNum; i ++) {

sum\_err += Math.*pow*(Label[i] - Preval[i], 2);

}

**return** sum\_err;

}

//LR模型训练

**public** **double**[] Updata(**double** [][] feature, **double**[] Label, **int** maxCycle, **double** rate) {

// 先计算样本个数和特征个数

**int** samNum = feature.length;

**int** paraNum = feature[0].length;

//初始化权重矩阵

**double** [] W = ParaInitialize(paraNum);

// 循环迭代优化权重矩阵

**for** (**int** i = 0; i < maxCycle; i ++) {

// 每次迭代后，样本预测值

**double** [] Preval = PreVal(samNum,paraNum,feature,W);

**double** sum\_err = error\_rate(samNum,Label,Preval);

**if** (i % 10 == 0) {

System.***out***.println("第" + i + "次迭代的预测误差为:" + sum\_err);

}

//预测值与标签的误差

**double** [] err = **new** **double**[samNum];

**for**(**int** j = 0; j < samNum; j ++) {

err[j] = Label[j] - Preval[j];

}

// 计算权重矩阵的梯度方向

**double** [] Delt\_W = **new** **double**[paraNum];

**for** (**int** n = 0 ; n < paraNum; n ++) {

**double** tmp = 0;

**for**(**int** m = 0; m < samNum; m ++) {

tmp += feature[m][n] \* err[m];

}

Delt\_W[n] = tmp / samNum;

}

**for**(**int** m = 0; m < paraNum; m ++) {

W[m] = W[m] + rate \* Delt\_W[m];

}

}

**return** W;

}

}

**import** java.io.\*;

**public** **class** SaveModel {

**public** **static** **void** savemodel(String filename, **double** [] W) **throws** IOException{

File f = **new** File(filename);

// 构建FileOutputStream对象

FileOutputStream fip = **new** FileOutputStream(f);

// 构建OutputStreamWriter对象

OutputStreamWriter writer = **new** OutputStreamWriter(fip,"UTF-8");

//计算模型矩阵的元素个数

**int** n = W.length;

StringBuffer sb = **new** StringBuffer();

**for** (**int** i = 0; i < n-1; i ++) {

sb.append(String.*valueOf*(W[i]));

sb.append("\t");

}

sb.append(String.*valueOf*(W[n-1]));

String sb1 = sb.toString();

writer.write(sb1);

writer.close();

fip.close();

}

**public** **static** **void** saveresults(String filename, **double** [] pre\_results) **throws** IOException{

File f = **new** File(filename);

// 构建FileOutputStream对象

FileOutputStream fip = **new** FileOutputStream(f);

// 构建OutputStreamWriter对象

OutputStreamWriter writer = **new** OutputStreamWriter(fip,"UTF-8");

//计算预测结果的个数

**int** n = pre\_results.length;

StringBuffer sb = **new** StringBuffer();

**for** (**int** i = 0; i < n-1; i ++) {

sb.append(String.*valueOf*(pre\_results[i]));

sb.append("\n");

}

sb.append(String.*valueOf*(pre\_results[n-1]));

String sb1 = sb.toString();

writer.write(sb1);

writer.close();

fip.close();

}

}

**public** **class** Sigmoid {

**public** **static** **double** sigmoid(**double** x) {

**double** i = 1.0;

**double** y = i / (i + Math.*exp*(-x));

**return** y;

}

}