人工智能第七次实验报告

逻辑回归 Logistic Regression

**逻辑回归（Logistic Regression）**是用于处理因变量为分类变量的回归问题，常见的是二分类或二项分布问题，也可以处理多分类问题，它实际上是属于一种分类方法。  
二分类问题的概率与自变量之间的关系图形往往是一个S型曲线，如图所示，采用的Sigmoid函数实现。  
logistic回归又称logistic回归分析，主要在流行病学中应用较多，比较常用的情形是探索某疾病的危险因素，根据危险因素预测某疾病发生的概率，等等。例如，想探讨胃癌发生的危险因素，可以选择两组人群，一组是胃癌组，一组是非胃癌组，两组人群肯定有不同的体征和生活方式等。这里的因变量就是--是否胃癌，即“是”或“否”，为两分类变量，自变量就可以包括很多了，例如年龄、性别、饮食习惯、幽门螺杆菌感染等。自变量既可以是连续的，也可以是分类的。通过logistic回归分析，就可以大致了解到底哪些因素是胃癌的危险因素。  
与多重线性回归的比较  
logistic回归(Logistic regression) 与多重线性回归实际上有很多相同之处，最大的区别就在于他们的因变量不同，其他的基本都差不多，正是因为如此，这两种回归可以归于同一个家族，即广义线性模型（generalized linear model）。这一家族中的模型形式基本上都差不多，不同的就是因变量不同，如果是连续的，就是多重线性回归，如果是二项分布，就是logistic回归，如果是poisson分布，就是poisson回归，如果是负二项分布，就是负二项回归，等等。只要注意区分它们的因变量就可以了。[1]  
logistic回归的因变量可以是二分非线性差分方程类的，也可以是多分类的，但是二分类的更为常用，也更加容易解释。所以实际中最为常用的就是二分类的logistic回归。

源代码：#include <iostream>

#include <string>

#include <fstream>

#include <sstream>

#include <vector>

#include <cmath>

template <typename DataType>

double sigmoid(DataType z) {

return 1.0/(1+exp((-1)\*z));

}

template <typename DataType, typename WeightType>

double getMatResult(typename::std::vector<DataType> &data, typename::std::vector<WeightType> &weights) {

double result=0.0;

for(size\_t i=0;i<data.size();++i) {

result+=data.at(i)\*weights.at(i);

}

return result;

}

template <typename DataType>

void DisplayData(typename::std::vector<std::vector<DataType> > &vv) {

std::cout<<"the number of data: "<<vv.size()<<std::endl;

for(size\_t i=0;i<vv.size();++i) {

for(typename::std::vector<DataType>::iterator it=vv[i].begin();it!=vv[i].end();++it) {

std::cout<<\*it<<" ";

}

std::cout<<std::endl;

}

}

template <typename DataType, typename WeightType>

double CostFun(typename::std::vector<std::vector<DataType> > &vv, typename::std::vector<WeightType> &v\_weights) {

double J=0.0;

typename::std::vector<DataType> v\_x;

for(size\_t i=0;i<vv.size();++i) {

v\_x.clear();

v\_x.push\_back(vv[i][0]);

v\_x.push\_back(vv[i][1]);

v\_x.push\_back(vv[i][2]);

double z=getMatResult(v\_x,v\_weights);

J=J+vv[i][3]\*log2(sigmoid(z))+(1-vv[i][3])\*log2(1-sigmoid(z));

}

J=-J/vv.size();

return J;

}

int main() {

std::ifstream infile\_feat("train.data");

std::string feature;

float feat\_onePoint;

std::vector<float> lines;

std::vector<double> v\_weights;

std::vector<std::vector<float> > lines\_feat;

lines\_feat.clear();

v\_weights.clear();

for(size\_t i=0;i<3;++i) {

v\_weights.push\_back(1.0);

}

while(!infile\_feat.eof()) {

getline(infile\_feat, feature);

if(feature.empty())

break;

std::stringstream stringin(feature);

lines.clear();

lines.push\_back(1.0);

while(stringin >> feat\_onePoint) {

lines.push\_back(feat\_onePoint);

}

lines\_feat.push\_back(lines);

}

infile\_feat.close();

std::cout<<"display train data: "<<std::endl;

DisplayData(lines\_feat);

double res=CostFun(lines\_feat, v\_weights);

std::cout<<"the value of cost function: "<<res<<std::endl;

std::vector<double> v\_x;

while(true) {

double grad0=0.0,grad1=0.0,grad2=0.0;

for(size\_t i=0;i<lines\_feat.size();++i) {

v\_x.clear();

v\_x.push\_back(lines\_feat[i][0]);

v\_x.push\_back(lines\_feat[i][1]);

v\_x.push\_back(lines\_feat[i][2]);

grad0+=(lines\_feat[i][3]-sigmoid(getMatResult(v\_x,v\_weights)))\*lines\_feat[i][0];

grad1+=(lines\_feat[i][3]-sigmoid(getMatResult(v\_x,v\_weights)))\*lines\_feat[i][1];

grad2+=(lines\_feat[i][3]-sigmoid(getMatResult(v\_x,v\_weights)))\*lines\_feat[i][2];

}

grad0=grad0/lines\_feat.size();

grad1=grad1/lines\_feat.size();

grad2=grad2/lines\_feat.size();

//0.03为学习率阿尔法

v\_weights[0]=v\_weights[0]+0.03\*grad0;

v\_weights[1]=v\_weights[1]+0.03\*grad1;

v\_weights[2]=v\_weights[2]+0.03\*grad2;

double res\_new;

res\_new=CostFun(lines\_feat,v\_weights);

if(std::abs(res\_new-res)<0.0000000001)

break;

res=res\_new;

}

for(size\_t i=0;i<3;++i) {

std::cout<<v\_weights.at(i)<<" ";

}

std::cout<<std::endl;

lines\_feat.clear();

infile\_feat.open("test.data");

while(!infile\_feat.eof()) {

getline(infile\_feat, feature);

if(feature.empty())

break;

std::stringstream stringin(feature);

lines.clear();

lines.push\_back(1.0);

while(stringin >> feat\_onePoint) {

lines.push\_back(feat\_onePoint);

}

lines\_feat.push\_back(lines);

}

infile\_feat.close();

std::cout<<"display test data: "<<std::endl;

DisplayData(lines\_feat);

for(size\_t i=0;i<lines\_feat.size();++i) {

v\_x.clear();

v\_x.push\_back(lines\_feat[i][0]);

v\_x.push\_back(lines\_feat[i][1]);

v\_x.push\_back(lines\_feat[i][2]);

res=getMatResult(v\_x,v\_weights);

double lable=sigmoid(res);

for(size\_t j=0;j<4;++j) {

std::cout<<lines\_feat[i][j]<<" ";

}

if(lable>0.5)

std::cout<<" 1"<<std::endl;

else

std::cout<<" 0"<<std::endl;

}

return 0;

}

运行结果：