哈尔滨工业大学计算机科学与技术学院

实验报告

课程名称： 机器学习

课程类型： 选修

实验题目： 逻辑回归

学号：1190201115
姓名： 陈宇豪

一、实验目的

理解逻辑回归模型。
掌握逻辑回归模型的参数估计算法。

二、实验要求及实验环境

实现两种损失函数的参数估计（1，无惩罚项；2.加入对参数的惩罚），可以采用梯度下降、共轭梯度或者牛顿法等。

三、设计思想（本程序中的用到的主要算法及数据结构）

3.1 主要算法：

用到的算法为梯度下降法，具体分析如下：

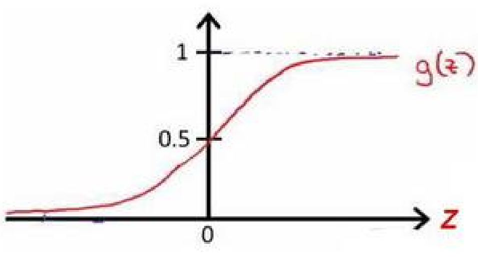
3.1.1 假设函数

从二元的分类问题开始讨论。 首先将因变量可能属于的两个类分别称为负向类和正向类，其中 0 表示负向类，1 表示正向类，则因变量 y ∈ 0，1 。故对于逻辑回归算法，首先要确定的是，对于这个算法，它的输出值永远在 0 到 1 之间 。

为此引入一个新的模型，该模型的输出变量范围始终在 0 和 1 之间。 逻辑回归模型的假设是

$$h_\theta(x) = g(\theta^T X)$$

其中，$g(z) = \dfrac{1}{1+e^{-z}}$，该函数的图像为



当 z 趋向于正无穷大时，g(z)无限趋向于 1。当 z 趋向于负无穷大时，g(z)无限趋向于 0.这样就满足了上述的前提。

```
sigmoid=1./(1+exp(-z));
```

3.1.2 决策边界

由于 y 的取值是离散的，上述假设函数的输出值是连续的，故设定决策边界，当 $h_\theta(x)$ 大于等于 0 时，预测 y=1，当 $h_\theta(x)$ 小于 0 时，预测 y=0。假设参数向量 $\theta=\begin{bmatrix}\theta_0\\ \theta_1\\ \theta_2\end{bmatrix}$，对于一组参数 X=$\begin{bmatrix}x_0\\ x_1\\ x_2\end{bmatrix}$， $h(x)=g(\theta^T X)$>0,即 $\theta_0 x_0+\theta_1 x_1+\theta_2 x_2>0$ 时，模型将预测 y=1。因此便可以绘制 $\theta_0 x_0+\theta_1 x_1+\theta_2 x_2$=0 这条直线作为分界线。

3.1.3 代价函数

分类的代价函数原理和线性回归类似，设 $J(\theta_0,\theta_1)=\frac{1}{m}\sum_{i=1}^{m}Cost(h_\theta(x^{(i)}),y^{(i)})$，但是 cost 函数不能和线性回归一样，否则得到的代价函数将是一个非凸函数。重新定义 cost 函数为：$Cost(h_\theta(x),y)=-y*\log(h_\theta(x))-(1-y)*\log(1-h_\theta(x))$。

对该函数进行分析，假设 y=1,显然函数的后半部分取值为 0，此时预测函数 h(x)越接近 1，则 log(h(x))的值越小，若预测函数非常接近零，那么代价将会趋向于无穷大。

到此，确定了代价函数之后，已经可以利用梯度下降法对参数进行求解了。令 $Cost(h_\theta(x),y)$ 关于 θ 求导，根据变化关系找到使 cost 最小的系数向量，得到 $\theta_j=\theta_j-\alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^i)-y^i)x_j^i$，下式中其中 train_rate 为变化率，lamda 为加入的正则项。

```
for j=1:train_times
    hypo_y=cal_sigmoid(X*w);
    for i=1:col
        tempw(i)=w(i)-train_rate*(sum((hypo_y-y).*X(:,i))+lamda*w(i));
    end
    new_cost=cal_cost(tempw,X,y);
```

同样，假如一次迭代后代价反而增大，考虑是学习率过大，应该将α进行一定的缩小。

3.2 数据生成

对于二维数据，协方差矩阵为 $C=\begin{bmatrix}\mathrm{cov}_{11}\,\mathrm{cov}_{12}\\ \mathrm{cov}_{21}\,\mathrm{cov}_{22}\end{bmatrix}$ 若满足朴素贝叶斯，则认为 X 与 Y 相互独立，cov12 和 cov21 都应为 0，反之则不为 0.利用 matlab 中的 mvmrnd 函数即可产

生符合或不符合朴素贝叶斯的数据。如下图，产生的是两组满足朴素贝叶斯的数据点，第一组的坐标均值为（-1，-0.3），cov11 为 0.3，cov22 为 0.4。num 为产生数据点的数量。

```
num=15;
X1=create_data([-1,-0.3],[0.3 0;0 0.4],num,0);
X2=create_data([1,0.3],[0.3 0;0 0.4],num,1);
```

```
data=mvnrnd(mu, S, num);
```

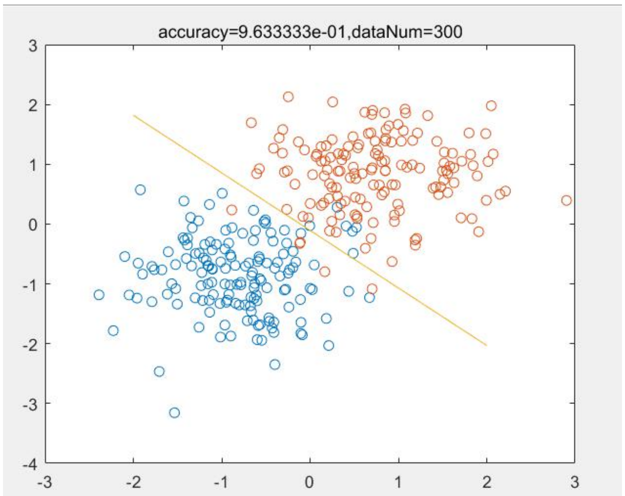
若想产生不符合朴素贝叶斯的数据集，将 cov12 和 cov21 改为不为零的数即可。

四、实验结果与分析

4.1 利用随机产生数进行训练和测试

数据集数量为 300，正则项取值为 10^-3。

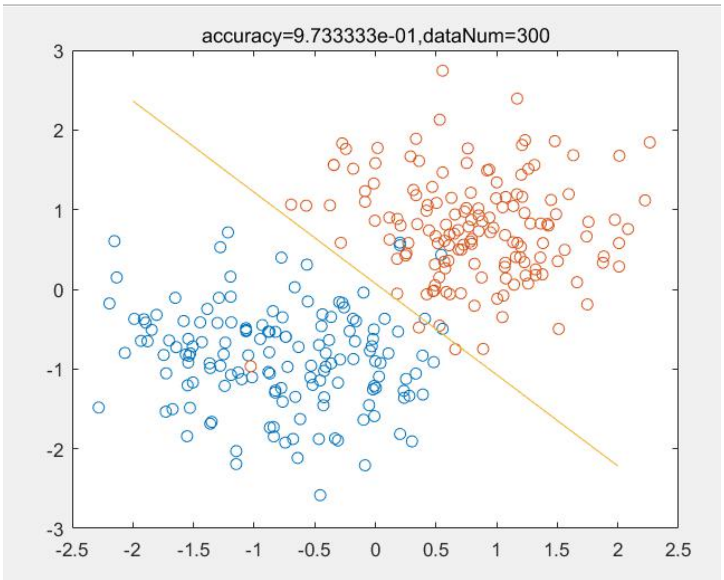
4.1.1 满足朴素贝叶斯无正则项：




```
accuracy =

    0.9633
```

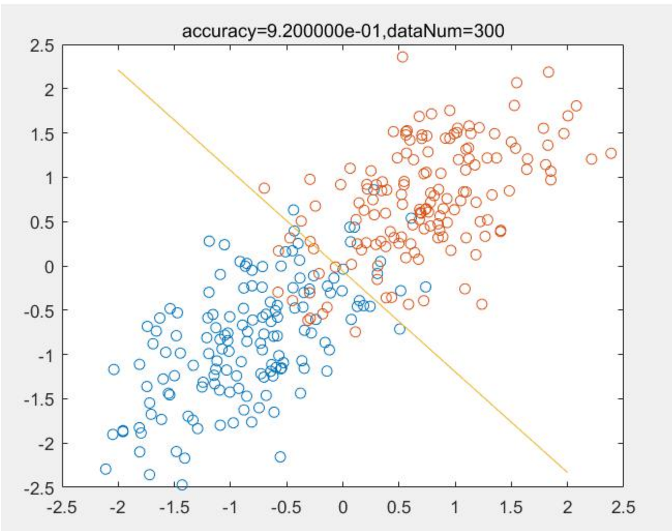
4.1.2 满足朴素贝叶斯有正则项：




```
accuracy =

    0.9733
```

4.1.3 不满足朴素贝叶斯无正则项：

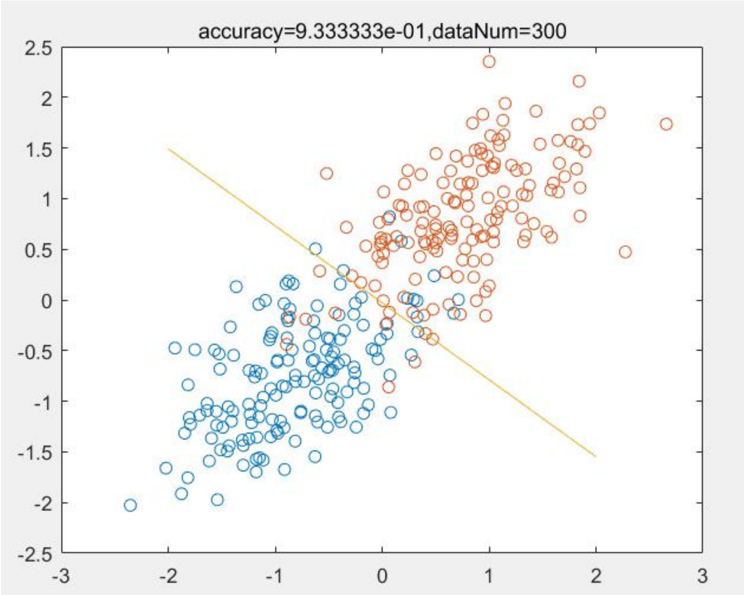


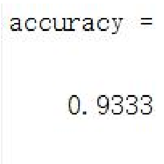

```
accuracy =

    0.9200
```

这里设定 cov12 为 0.2，可以看到参数之间存在一定相关性，散点图相较符合朴素贝叶斯的要更加聚拢，呈现条状。

4.1.4 不满足朴素贝叶斯有正则项：





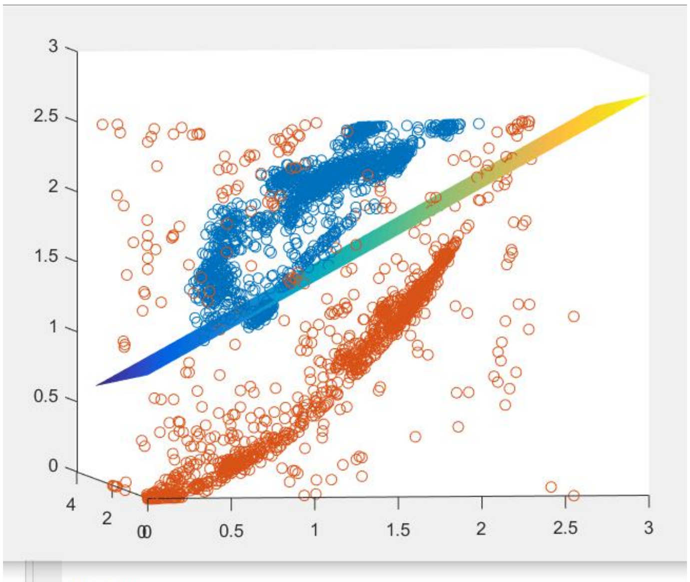
通过以上四次实验可以大致看出，满足朴素贝叶斯的散点图的拟合效果较好，但准确率提高不大，不超过百分之5。有正则项的拟合效果比没有正则项的也要稍好，但在实验条件下准确率提高也不到百分之2.

4.2UCI 数据训练并测试

利用 UCI 网站上关于皮肤分类的样本进行测试，原数据有二十多万个，由于梯度下降法训练速度较慢，我从中随机抽取了 344 组数据作为训练集，3103 组数据作为验证集，其中结果为 1 和结果为 0 的数据数量都保持大致相同。

数据集中有三个参数，故训练结果可以用三维图像展示。

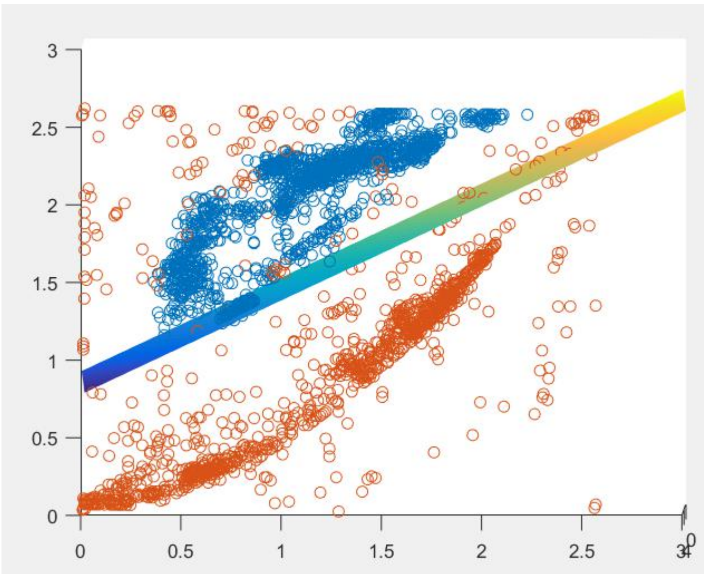
4.2.1 无正则项：



```
accuracy =

    0.9317
```

4.2.2 有正则项：



```
accuracy =

    0.9555
```

五、结论

1. 用梯度下降法进行分类时，满足朴素贝叶斯的散点图的拟合效果较好，但准确率提高不大。有正则项的拟合效果比没有正则项的也要稍好。
2. 数据量规模较大时，不再适合用梯度下降法，因为需要非常多的训练时间。

六、参考文献

https://veal98.gitee.io/cs-wiki/#/%E4%BA%BA%E5%B7%A5%E6%99%BA%E8%83%BD/%E6%9C%BA%E5%99%A8%E5%AD%A6%E4%B9%A0/%E5%90%B4%E6%81%A9%E8%BE%BE/4-%E9%80%BB%E8%BE%91%E5%9B%9E%E5%BD%92+%E6%AD%A3%E5%88%99%E5%8C%96?id=%f0%9f%8d%9c-%e9%80%bb%e8%be%91%e5%9b%9e%e5%bd%92-logistic-regression

七、附录：源代码（带注释）

源代码地址：

https://github.com/1190201115/HIT-machineLearningLab/tree/main/lab2

二维逻辑回归部分：

主函数

```
%%主函数
%%num 为产生一类数据的数量，由于是二分的，所以总的数据量为 2m
num=150;
lamda=10^-4;
%%随机数生成，产生的两类坐标向量列相加为 X
X1=create_data([-0.8,-0.8],[0.4 0;0 0.4],num,0);
X2=create_data([0.8,0.8],[0.4 0;0 0.4],num,1);
X=[X1;X2];

label=X(:,4);%%label
X(:,4)=[];%%X 每列依次为 1，x,y，消去第四列（label）

%%通过梯度下降法得到分类线的系数 w
w=gradient(X,label,lamda);

%%绘图
Px=(-2:0.01:2);
Py=(w(1)+w(2)*Px)/(-w(3));
plot(Px,Py);
accuracy=cal_accuracy(w,X,label,2*num);
```

```
string1=sprintf('%s%d,%s%d','accuracy=',accuracy,'dataNum=',2*num);
title(string1);
```

数据产生函数

```
function [X] = create_data(mu,S,num,label)
%产生以 mu 为均值，S 为方差，num 为数量的离散数据点

data=mvnrnd(mu, S, num);
plot(data(:,1),data(:,2),'o');

if(label==1)
X=ones(num,4);
else
X=zeros(num,4);
X(:,1)=ones(num,1);
end
X(:,2)=data(:,1);
X(:,3)=data(:,2);
hold on;
end
```

计算 sigmoid

```
function [sigmoid ] = cal_sigmoid( z )
%将数据控制在 0 到 1
sigmoid=1./(1+exp(-z));
end
```

代价计算函数：

```
function [ Cost ] = cal_cost( theta,X,label )
%计算当前代价（误差）
[row,~]=size(X);
Cost=0;
for i=1:row

Cost=-label(i)*log(cal_sigmoid(X(i,:)*theta))-(1-label(i))*log(1-cal_sigmoid(X(i,:)*t
heta))+Cost;
end
```

精确度计算函数

```
function [accuracy    ] = cal_accuracy( w,X,label,num)
%计算验证集中正确预测数据数/总数据数
%
data_x=X(:,2);
data_y=X(:,3);
[row,~]=size(X);
mylabel=zeros(row,1);
for i=1:row
        if (w(1)+w(2)*data_x(i)+w(3)*data_y(i))<0
                mylabel(i)=0;
        else
                mylabel(i)=1;
        end
end

%%利用预测的 label 向量和实际的标记异或，再对向量内部求和，得到预测错误的数据数
bitxor(mylabel,label);
accuracy=(1-sum(bitxor(mylabel,label))/num)

end
```

梯度下降法

```
function [w ] = gradient(X,y,lamda)
%梯度下降法求系数向量

%%col 为 X 的列数，即变量数+1，X 的第一列为 1
[~,col]=size(X);

%%初始系数向量设置为全 0
w = zeros(col, 1);
old_cost=cal_cost(w,X,y);
tempw=w;

%%设置步长和最大训练次数
train_rate=0.02;
train_times=100000;
for j=1:train_times
        hypo_y=cal_sigmoid(X*w);%%计算预测的 label 值（0 到 1 之间）
```

```
    for i=1:col
        tempw(i)=w(i)-train_rate*(sum((hypo_y-y).*X(:,i))+lamda*w(i));
    end
    new_cost=cal_cost(tempw,X,y);
    %%满足精度时，记录训练次数，退出迭代
    if abs(old_cost-new_cost)<10^-6
        train_times=j;
        train_times
        %%输出训练次数
        break;
    end
    if new_cost<old_cost
        w=tempw;
        old_cost=new_cost;
        continue;
    end
    if new_cost>=old_cost
        train_rate=train_rate/2;
    end

end
end
```

UCI 验证函数

UCI 训练函数

```
function [ output_args ] = UCI_data(    )
%从 UCI 下载了一份分类数据，对其进行处理，Skin.txt 为训练集数据

%%读文件，x，y，z 分别为三个变量的列向量，kind 为分类向量，因为原数据用 1 和 2 分类，为了直接用之前的函数，故改为 0 和 1
file=load('D:\matlab\code_here\logistic_lab2\Skin.txt');
x=file(:,1);
y=file(:,2);
z=file(:,3);
x=x./100;
y=y./100;
z=z./100;
kind=file(:,4);
kind=kind-1;
[row,~]=size(x);
```

```
line1=ones(row,1);
X=[line1,x,y,z];
w=gradient(X,kind,0.9)

%%训练得到系数向量 w 后，传递到 test 函数中绘图
UCI_test(w);

%{
ax = gca;
ax.XAxisLocation = 'origin';
ax.YAxisLocation = 'origin';
%}
end
```

UCI 验证函数

```
function [ ] = UCI_test(w)
%绘图，计算精确度
% test0 中存了 label 为 1 的数据，处理后 label 为 0
file=load('D:\matlab\code_here\logistic_lab2\Skintest0.txt');
x0=file(:,1);
y0=file(:,2);
z0=file(:,3);
x0=x0./100;
y0=y0./100;
z0=z0./100;
plot3(x0,y0,z0,'o');
hold on;
kind1=file(:,4);
kind1=kind1-1;

%%test1 中存了 label 为 2 的数据，处理后为 1
file=load('D:\matlab\code_here\logistic_lab2\Skintest1.txt');
x1=file(:,1);
y1=file(:,2);
z1=file(:,3);
x1=x1./100;
y1=y1./100;
z1=z1./100;
plot3(x1,y1,z1,'o');
hold on;
kind2=file(:,4);
kind2=kind2-1;
kind=[kind1;kind2];
```

```
%%绘图范围
Px=(0:0.01:3);
Py=(0:0.01:3);
[x,y]=meshgrid(Px,Py);
Pz=(w(1)+w(2)*x+w(3)*y)/(-w(4));
surf(x,y,Pz);
shading interp

%%计算精确度
[row1,~]=size(x0);
[row2,~]=size(x1);
row=row1+row2;
line=ones(row,1);
x=[x0;x1];
y=[y0;y1];
z=[z0;z1];
X=[line,x,y,z];
UCI_accuracy(w,X,kind);
end
```

UCI 精确度计算函数

```
function [ accuracy ] = UCI_accuracy( w,X,kind )
%计算三维函数的精确度
% 和 cal_accuracy 原理完全一致，不想改那个函数了就又另外写了一个这个
data_x=X(:,2);
data_y=X(:,3);
data_z=X(:,4);
[row,~]=size(X);
kind0=zeros(row,1);
for i=1:row
    if (w(1)+w(2)*data_x(i)+w(3)*data_y(i)+w(4)*data_z(i))>0
        kind0(i)=1;
    else
        kind0(i)=0;
    end
end
bitxor(kind0,kind);
accuracy=1-sum(bitxor(kind0,kind))/row
end
```