**Logistic回归**

**陈显锋 Y30180636**

**Logistic回归**为概率型非线性回归模型，是研究二分类观察结果与一些影响因素之间关系的一种多变量分析方法。通常的问题是，研究某些因素条件下某个结果是否发生。

我们先从LR分类器说起。LR分类器，即**Logistic Regression Classifier。**

在分类情形下，经过学习后的LR分类器是一组权值，当测试样本的数据输入时，这组权值与测试数据按照线性加和得到

 （1）

这里是每个样本的个特征。

之后按照sigmoid函数的形式求出

 （2）

由于**sigmoid函数**的定义域为，值域为，因此最基本的LR分类器适合对两类目标进行分类。所以**Logistic回归**最关键的问题就是研究如何求得这组权值。这个问题是用**极大似然估计**来做的。

**Logistic回归模型：**

考虑具有个独立变量的向量，设条件慨率为根据观测量相对于某事件发生的概率。那么**Logistic回归**模型可以表示为

 （3）

这里称为Logistic函数。其中。

那么在条件下不发生的概率为

 （4）

所以事情发生与不发生的概率之比为

 （5）

这个比值称为事件的发生比，简记为odds。

对odds取对数得到

 （6）

可以看出Logistic回归都是围绕一个Logistic函数来展开的。

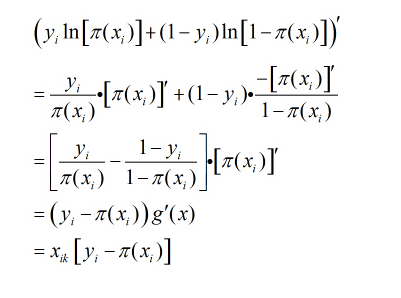
接下来就讲如何用**极大似然估计**求分类器的参数：假设有个观测样本，观测值分别为，设为给定条件下得到的概率，同样地，的概率为，所以得到一个观测值的概率为。

因为各个观测样本之间相互独立，那么它们的联合分布为各边缘分布的乘积。得到似然函数为

 （7）然后我们的目标是求出使这一似然函数的值最大的参数估计，最大似然估计就是求出参数，使得取得最大值，对函数取对数得到

 （8）

继续对这个分别求偏导，得到个方程，比如现在对参数求偏导，由于

 （9）

所以得到

 （10）

这样的方程一共有个，所以现在的问题转化为解这个方程形成的方程组。上述方程比较复杂，我们用牛顿迭代方法求解。

**用Logistics回归对iris数据分类**

**import** numpy **as** np  
**from** sklearn.datasets **import** load\_iris  
**from** sklearn.decomposition **import** PCA  
**import** matplotlib.pyplot **as** plt  
**import** math  
  
iris = load\_iris()  
newdata = iris.data  
newdata = np.array(newdata)  
  
*#PCA降维*pca = PCA(n\_components=2)  
pca.fit(iris.data)  
iris\_1 = pca.transform(iris.data)  
  
plt.figure(1)  
plt.scatter(iris\_1[:, 0], iris\_1[:, 1])  
plt.show()  
  
*#定义sigmoid函数***def** my\_sig(w, x):  
 w = w.T  
 r = np.dot(w, x)  
 s = 1.0 / (1 + math.exp(-1\*r))  
 **return** s  
  
*#梯度下降法***def** my\_grad(w, x, y):  
 len\_y = len(y)  
 sum\_0 = 0  
 **for** i **in** range(0, 99):  
 s = my\_sig(w, x[:, i])  
 x1 = x[:, i]  
 x1 = x1.reshape(x1.shape[0], 1)  
 sum\_0 = sum\_0 + (s - y[i]) \* x1  
  
 g = 1 / len\_y \* sum\_0  
 **return** g  
  
*#损失函数***def** my\_fun(w, x, y):  
 len\_y = len(y)  
 sum\_0 = 0  
 **for** i **in** range(0, len\_y-1):  
 s = my\_sig(w, x[:, i])  
 sum\_0 = sum\_0+y[i]\*math.log(s)+(1-y[i])\*math.log(1-s)  
 f = -1/len\_y\*sum\_0  
 **return** f  
  
*#构造训练样本和测试样本*iris\_1 = iris\_1[:100, :]  
iris\_1 = iris\_1.T  
a = np.ones((1, 100))  
x = np.vstack((iris\_1, a))  
y = np.vstack((np.ones((50, 1)), np.zeros((50, 1))))  
  
*#梯度下降法求解Logistic回归*w = np.array([[1], [1], [1]])  
f\_test = np.zeros(5000)  
**for** i **in** range(0, 4999):  
 w = w - 0.1 \* my\_grad(w, x, y)  
 f\_test[i] = my\_fun(w, x, y)  
  
plt.figure(2)  
plt.plot(np.arange(5000), f\_test)  
plt.show()  
y\_test = np.zeros(100)  
**for** i **in** range(0, 99):  
 y\_test[i] = my\_sig(w, x[:, i])  
print(y\_test)  
plt.figure(3)  
plt.scatter(np.arange(100),y\_test)  
plt.show()

