

**模式识别大作业**

题 目 **logistic回归原理分析及分类应用**

学 院 信息科学与工程

专 业 控制科学与工程

组 员 左春玲

指导教师 赵海涛

**完成日期： 2018 年 10 月25日**

**logistic回归原理分析及分类应用**

**一、logistic回归和线性回归的关系**

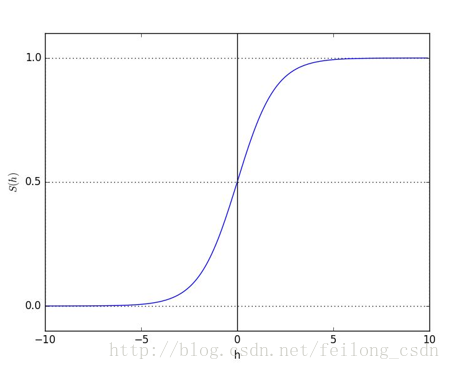
首先给出线性回归模型：

写成向量形式为：

logistic回归为二分类模型，利用sigmoid函数构造预测模型：

Sigmoid函数为:

图象如下所示：



我们在原来的线性回归模型外套上sigmoid函数便形成了logistic回归模型的预测函数，可以用于二分类问题。

二、**Logistic回归模型**

考虑具有n个独立变量的向量, 构造预测模型：

设条件概率为根据观测量相对于某事件x发生的概率为

那么在x条件下y=0的概率为

综合起来即为

**三、构造cost函数**

取似然函数为

对数似然函数为：

最大似然估计就是要求得使l(w)取最大值时的w，令

乘了一个负的系数，所以J(w)取最小时的w为要求的最佳参数，可以用梯度下降法求得。

**四、梯度下降法求J(w)的最小值**

根据梯度下降法可得**w**的更新过程：

式中为学习步长，求偏导如下：

上式求解过程中用到如下的公式：

因此，w的更新过程可以写成：

因为式中本来为一常量，所以

**五、梯度下降过程向量化**

首先对于引入的数据集**x**来说，均是以矩阵的形式引入的，**x**的每一行为一条训练样本，而每一列为不同的特征值，如下：

,

其中m是数据样本的个数，n是数据的维度，也就是数据特征的数量。

约定待求的参数**w**的矩阵形式为：

先求 并记为**A：**

求并记为**E：**

g(**A**)的参数**A**为一列向量，所以实现**g(~)**函数时要支持列向量作为参数，并返回列向量。由上式可知可以由一次计算求得。

再看w的更新过程，当j=0时：

同时可以求出，

综合起来就是：

下面便可以在实例中应用此迭代公式进行实际的分析了，下面便以简单的iris数据集的二分类问题为例来分析logistic回归算法的使用。

Python程序如下：

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.model\_selection import train\_test\_split

import datetime

from sklearn import preprocessing

import numpy as np

from sklearn.preprocessing import MinMaxScaler

from sklearn.datasets import load\_iris

def loadDataSet(): #读取数据集并处理函数

dataMatrix = []

datalabel = []

plt.style.use('ggplot') #使用自带的样式进行美化

iris = load\_iris() #获取数据

data = iris.data

target = iris.target

X = data[0:100] #选择前两类各50组数据

Y = target[0:100]

datalabel = np.mat(Y) #使数据变成标准矩阵形式

datalabel=np.transpose(datalabel) #进行转置

dataMatrix = np.mat(X)

minmax\_x\_train = MinMaxScaler()

x\_train\_std = minmax\_x\_train.fit\_transform(dataMatrix) #使数据标准化

dataMatrix = np.mat(x\_train\_std)

return dataMatrix,datalabel

def sigmoid(X):

return 1.0/(1+np.exp(-X)) #sigmoid函数形式

def graAscent(dataMatrix,matLabel,num): #梯度下降

m,n=np.shape(dataMatrix)

w=np.ones((n,1)) #将w初始化为1

alpha=0.01 #设置学习速率为alpha

for i in range(num): #num为迭代次数

error=sigmoid(dataMatrix\*w)-matLabel

w=w-alpha\*dataMatrix.transpose()\*error

return w

def predict(w,X): #预测函数

m = X.shape[0] #取列数

Y\_prediction = np.zeros((m,1))

#w = w.ones(X.shape[0],1)

A = sigmoid(np.dot(X,w))

for i in range(A.shape[0]):

if A[i,0]>0.5:

Y\_prediction[i ,0]=1

else:

Y\_prediction[i ,0]=0

assert(Y\_prediction.shape == (m,1))

return Y\_prediction

def loss(X,Y,num,print\_cost=False): #损失函数

#costs=loss(weight,dataMatrix,matLabel, num)

m, n = np.shape(dataMatrix)

w = np.ones((n, 1))

alpha = 0.01

costs = []

print\_cost=0

for i in range(num):

# 记录损失

error = sigmoid(dataMatrix \* w) - matLabel #损失误差

w = w - alpha \* dataMatrix.transpose() \* error #更新w

A = sigmoid(np.dot(X, w))

w = np.array(w)

A = np.array(A)

Y= np.array(Y)

cost = (- 1 / m) \* np.sum(Y \* np.log(A) + (1 - Y) \* (np.log(1 - A)))

if i % 10== 0:

costs.append(cost)

print("迭代的次数: %i ， 误差值： %f" % (i, cost))

return costs

if \_\_name\_\_ == '\_\_main\_\_':

dataMatrix,matLabel=loadDataSet()

print(dataMatrix.shape) #打印训练数据维度

print(matLabel.shape) #打印训练目标维度

num = 2000 #设置迭代次数

#weight=graAscent(dataMatrix,matLabel)

weight= graAscent(dataMatrix,matLabel,num) #计算权值

print(weight.shape) #打印权值矩阵维度

y=predict(weight,dataMatrix) #训练数据的预测结果

print(y.T)

print("训练集的准确度为：", format(100 - np.mean(np.abs(y - matLabel) \* 100)), "%")

costs = loss(dataMatrix, matLabel, num)

plt.plot(costs)

plt.ylabel('cost')

plt.xlabel('iterations (per hundreds)')

plt.show()

结果为：

(100, 4)

(100, 1)

(4, 1)

[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.

0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.

1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.

1. 1. 1. 1.]]

训练集的准确度为： 100.0 %

