第五节、逻辑回归

1. 线性回归

基本形式:给定由d哥属性描述的示例x = (x1;x2;x3;……;xn)，其中xi是x在第i个属性上的取值，线性模型(linear mode)试图学得一个通过属性的线性组合来进行预测的函数，即

f(x) = w1x1 + w2x2 + w3x3 + …… + wdxd + d,

一般用向量形式写成

f(x) = (w^t\*)\*x + b

其中w = (w1;w2;w3;……’wd).w和b学得之后，模型就得以确定

线性回归试图学得

f(xi) = wxi + b,使得f(xi) ~\_ui.

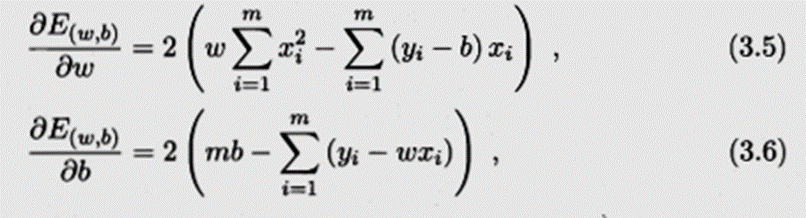
如何确定w和b呢？显然，关键在于如何衡量f(x)与y之间的差别，2.3节介绍过，均方误差(2.2)是回归任务中最常用的性能度量，因此我们可试图让均方误差最小化，即

(w\*,b\*) = arg min(w,b)E(f(xi) – yi)^2

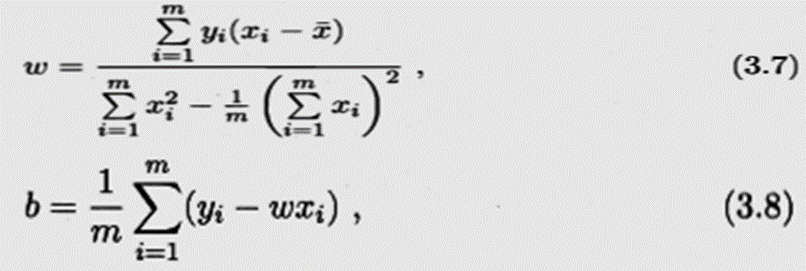
= arg min(w,b)E(yi – wxi – b)^2

均方误差由非常好的几何意义，它对应了常用的欧几里得距离或者简称“欧氏距离”，给予均方误差最小化进行模型求解的方法称为“最小二乘法”。在线性回归中，最小二乘法就是试图找到一条直线，是所有样本到直线上的欧氏距离之和最小

求解w和b使E(w,b) = E(yi – wxi – b)^2最小化的过程，称为线性回归模型的最小二乘“参数估计”。我们可将E(w,b)分别对w,b,求导，得到



然后令(3.5)和(3.6)为零可得到w和b最优解的闭式



其中x- =1/mExi为x的均值

1. 逻辑回归

由来:对于二类线性可分的数据集，使用线性感知器就可以很好的分类。

但是如果二类线性不可分的数据集，我们无法找到一条直线能够将两种类别很好的去部分，即线性回归的分类法对于线性不可分的数据无法有效分类。

数据线性可分可以使用线性分类器，如果数据线不可分，可以使用非线性分类器

如果我我们想知道对于一个二类分类问题，对于具体的一个样例，我们不仅想知道该类属于某一类，而且还想知道该类属于某一类的概率多大,有什么办法呢？

线性回归和非线性回归的分类问题都不能给予解答，因为线性回归和非线性回归的问题，假设其分类函数如下：



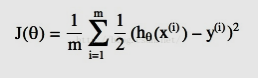
y的阈值处于（−∞，+∞），此时不能很好的给出属于某一类的概率，因为概率的范围是[0,1],我们需要一个更好的映射函数，能够将分类的结果很好的映射成为[0,1]之间的概率，并且这个函数能够具有很好的可微分性。在这种需求下，人们找到了这个映射函数，即逻辑斯谛函数，也就是我们常说的sigmoid函数，其形式如下：



1. 代价函数

我们通过对判定边界的说明，知道会有合适的参数使得模型具有很好的分类判定边界，那么问题就来了，我们如何判定我们的参数是否合适，有多合适呢？更进一步，我们有没有办法去求得这样的合适参数呢？

这就是我们要提到的代价函数与梯度下降了。所谓的代价函数（Cost Function），其实是一种衡量我们在这组参数下预估的结果和实际结果差距的函数，比如说线性回归的代价函数定义为：

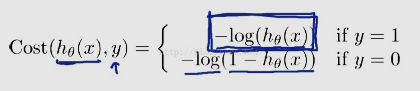


但是这会引发代价函数为“非凸”函数的问题，简单一点说就是这个函数有很多个局部最低点

而我们希望我们的代价函数是一个如下图所示，碗状结构的凸函数，这样我们算法求解到局部最低点，就一定是全局最小值点。

因此，上述的Cost Function对于逻辑回归是不可行的，我们需要其他形式的Cost Function来保证逻辑回归的成本函数是凸函数。

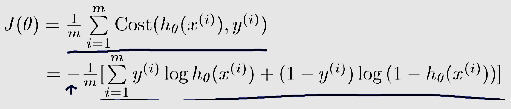
我们跳过大量的数学推导，直接出结论了，我们找到了一个适合逻辑回归的代价函数:



**梯度下降算法**

梯度下降算法是调整参数θ使得代价函数J(θ)取得最小值的最基本方法之一。从直观上理解，就是我们在碗状结构的凸函数上取一个初始值，然后挪动这个值一步步靠近最低点的过程，如下图所示：

简化的逻辑回归的代价函数：



从数学上理解，我们为了找到最小值点，就应该朝着下降速度最快的方向(导函数/偏导方向)迈进，每次迈进一小步，再看看此时的下降最快方向是哪，再朝着这个方向迈进，直至最低点。

1. sklearn中的应用

在sklearn中，与逻辑回归有关的主要是这3类。

**LogisticRegression**、**LogisticRegressionCV**和**logistic\_regression\_path**

其中LogisticRegression和LogisticRegressionCV的主要区别LogisticRegressionCV使用了交叉验证来选择正则化系数C。而LogisticRegression需要自己每次指定一个正则化系数。除了交叉验证，以及选择正则化系数C以外，LogisticRegressionLogisticRegressionCV的使用方法基本相同。

**1、正则化选择参数:penalty**

LogisticRegression和LogisticRegressionCV默认就带了正则化项，penalty参数可选择的值为“l1”“l2”,分别对应L1的正则化和L2的正则化，默认是L2的正则化。

在调参时如果我们主要的目的只是为了解决过拟合，一般选择L2正则化就够了，如果还是过拟合即预测效果差的时候就可以考虑L1正则化。

**2、优化算法选择参数：solver**

solver参数决定了我们对逻辑回归损失函数的优化，有4种算法可以选择，分别是：

liblinear:使用了开源的liblinear库实现，内部使用了坐标轴下降法来迭代优化损失函数。

lbfgs:拟牛顿法的一种，利用损失函数二阶导数矩阵即海森矩阵来迭代优化损失函数。

newton-cg:也是牛顿法家族的一种，利用损失函数二阶导数矩阵即海森矩阵来迭代优化损失函 数。

sag:即随机平均梯度下降，是梯度下降法的变种，和普通下降法的区别是每次迭代仅仅用一部分的样本来计算梯度，适合于样本数据多的时候。

**3、分类方式选择参数：multi\_class**

milti\_class参数决定了我们分类方式的选择，有ovr和 multinomial两个值可以选择，默认是ovr。

ovr相对简单但大多分类效果相对略差。而multinomial分类相对精确，但是分类速度没有ovr快。

如果选择了ovr，则4中损失函数的优化方法liblinear,newton-cg,lbfgs 和sag都可以选择， 但是 如果选择了multinomial，则只能选择newton-cg,lbfgs和sag了。

**4、类型权重参数：class\_weight**

class\_weight参数用于标示分类模型中各种类型的权重，可以不输入，即不考虑权重，或者说所有类型的权重一样。

如果选择输入的话，可以选择balanced让类库自己计算类型权重，或者我们自己输入各个类型的权重， 比如对于0,1的二元模型，我们可以定义class\_weight={0:0.9,1:0.1},这样类型0的权重为90%，而类型1的权重为10%。

如果class\_weight选择balanced，那么类库会根据训练样本来计算权重。某种类型样本量越多，则权重越低，样本量越少，则权重越高。

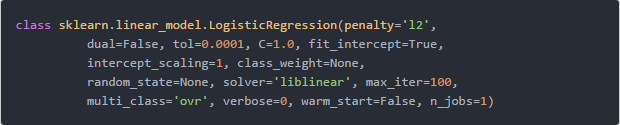
**5、样本权重参数：sample\_weight**

由于样本不平衡，导致样本不是总体样本的无偏估计，从而可能导致我们的模型预测能力

下降。遇到这种情况，我们可以通过调节样本权重来尝试解决这个问题。调节样本权重的方法

有两种，第一种是在class\_weight使用balancecd，第二种是在调用fit函数时，通过sample\_

weight来自己调节每个样本的权重。



前的数据分类存储

if int(labelMat1[i])== 1:

xcord1.append(dataArr[i,1]); ycord1.append(dataArr[i,2])

else:

xcord2.append(dataArr[i,1]); ycord2.append(dataArr[i,2])

for i in range(n): #将训练后的数据分类存储

if int(labelMat2[i])== 1:

xcord3.append(dataArr[i,1]); ycord3.append(dataArr[i,2])

else:

xcord4.append(dataArr[i,1]); ycord4.append(dataArr[i,2])

fig = plt.figure("LogisticRegression") #新建一个画图窗口

ax = fig.add\_subplot(111) #添加一个子窗口

ax.set\_title('Original')

ax.scatter(xcord1, ycord1, s=30, c='red', marker='s')

ax.scatter(xcord2, ycord2, s=30, c='green')

x = arange(-3.0, 3.0, 0.1) #定义x轴

y = (-weights[0] - weights[1]\*x) / weights[2] # x2 = f(x1) 定义y轴 a0\*1+a1\*x+a2\*y

ax.plot(x, y) #画一条直线

plt.xlabel('X1'); plt.ylabel('X2')

plt.figure("logisticRegression")

plt.title('Forecast')

plt.scatter(xcord3, ycord3, s=30, c='red', marker='s')

plt.scatter(xcord4, ycord4, s=30, c='green')

plt.plot(x,y)

plt.xlabel('X1');plt.ylabel('X2')

plt.show()

def getResult(dataArr,A):

h = sigmoid(mat(dataArr)\*A) #预测结果h(a)的值

H = []

for i in range(shape(h)[0]):

if h[i,0] > 0.5:

H.append(1)

else:

H.append(0)

return H

import numpy as np

import xlrd

from sklearn.linear\_model import LogisticRegression

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

from sklearn.metrics import classification\_report

def open\_excel(file):

"""

打开excel文件获取数据

:param file: 文件所在的位置

:return: 文件数据

"""

try:

data = xlrd.open\_workbook(file)

return data

except Exception as e:

print(str(e))

def split\_feature(row):

"""

将该行特征处理后放入列表中

:param row:一行特征数据

:return: 返回数据列表

"""

app = []

for i in range(16):

app = app + [row[i]]

return app

def loadDataSet(path, training\_sample, colnameindex=0, by\_name=u'sheet1'):

"""

加载数据

:param path: 数据文件存放路径

:param training\_sample: 数据文件名

:param colnameindex: 文件列名下标

:param by\_name: 表名

:return: 数据集和类别标签

"""

dataMat = [] # 定义数据列表

labelMat = [] # 定义标签列表

filename = path + training\_sample # 形成特征数据的完整路径

data = open\_excel(filename) # 打开文件获取数据

table = data.sheet\_by\_name(by\_name) # 获得数据表

nrows = table.nrows # 得到表数据总行数

colnames = table.row\_values(colnameindex) # 某一行数据 ['user\_id', 'age\_range', 'gender', 'merchant\_id','label']

for rownum in range(1, nrows): # 也就是从Excel第二行开始，第一行表头不算

row = table.row\_values(rownum) # 取一行数据

'''

判断2,3,6列数据是否为空，若为空则丢弃该行数据

'''

if row[1] == '' or row[2] == '' or row[5] == '':

continue

if row:

app = split\_feature(row) # 将特征值转化为列表

dataMat.append(app)

labelMat.append(float(row[16])) # 获取类别标签

return dataMat, labelMat

def show\_accuracy(a, b, tip):

"""

计算准确率

:param a: 真实类别

:param b: 预测标签

:param tip: 描述

:return: 准确率

"""

acc = a.ravel() == b.ravel()

print("%s Accuracy:%.3f" % (tip, np.mean(acc)))

def main():

"""

主函数

:return: null

"""

path = "E:\\"

training\_sample = 'featuredata.xls' # 特征数据文件

trainingSet, trainingLabels = loadDataSet(path, training\_sample) # 取特征数据和标签数据

x = np.array(trainingSet) # 将数据部分列表（list）格式转化为数组(array)格式

y = np.array(trainingLabels) # 将标签部分的列表（list）格式转化为数组格式（array）

'''

将数据分为训练数据和测试数据两部分

x\_train 训练数据

x\_test 测试数据

y\_train 训练数据标签

y\_test 测试数据标签

'''

train\_data, test\_data, train\_label, test\_label = train\_test\_split(x, y, random\_state=1, test\_size=0.3)

#选择模型

clf = LogisticRegression()

#把数据交给模型训练

clf.fit(train\_data, train\_label)

hat\_test\_label = clf.predict(test\_data)

print(classification\_report(test\_label, hat\_test\_label))

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

"""

程序入口

"""

main()

from LogisticRegression import \*

def test\_logistic\_regression():

path="F:\\AIData\Ch05\\"

training\_sample = 'trainingSet.txt' #训练数据文件

testing\_sample = 'testingSet.txt' #测试训练文件

trainingSet, trainingLabel = loadDataSet(path,training\_sample) #读入训练数据

A = gradAscent(trainingSet, trainingLabel) # 回归系数a的值

testingSet, testingLabel = loadDataSet(path, testing\_sample) #读入测试数据

h = getResult(testingSet,A) #预测结果

plotBestFit(A.getA(),testingSet,testingLabel,h) #图形化展示

'''

程序入口

'''

if \_\_name\_\_ == "\_\_main\_\_":

test\_logistic\_regression()