華東理工大學

模式识别大作业

题　　目　Logistics 回归预测广告点击率

学　　院　信息科学与工程

专　　业　信息与通信工程

组　　员　张建美

指导教师　赵海涛

完成日期：　2018 年 10 月 24 日

基于 Logistic 回归的广告点击和 iris 数据集的 python 实现

组员：张建美

通过模式识别学习，在赵海涛老师的辛勤指导下，对模式识别有了最基本的认识，这次做的题目是关于广告点击的问题。使用目前课程学过的 Logistic 回归进行模型的训练。Logistic 回归用于估计某种事物的可能性。 Logistic 回归用以二分类问题。多分类问题用 softmax 分类器。下文开始将探讨 logistic 回归，使用梯度下降算法将 logistic 回归算法应用到实例中。

一、logistic 回归

logistic 回归又称 logistic 回归分析，是一种广义的线性回归分析模型，常用于数据挖掘，疾病自动诊断，经济预测等领域。例如对于肿瘤这种，输出值也就是因变量为“是”或“否”，自变量就可以包括很多了，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。对于这种问题，我们可以先使用线性回归，首先给出线性回归模型：

$$\mathrm{f} = w_0 x_0 + w_1 x_1 + ... + w_n x_n + b$$

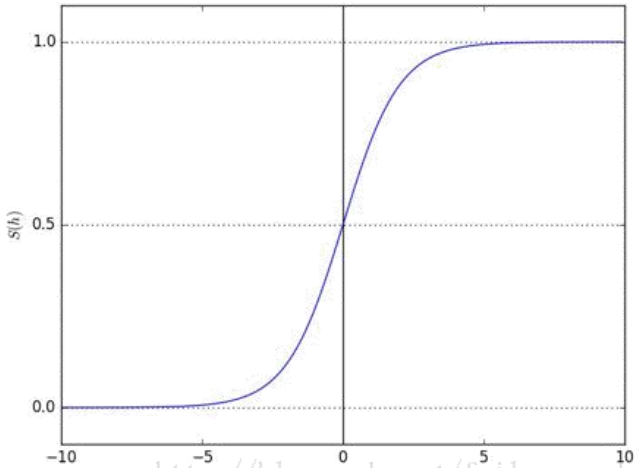
写成向量形式为：

$$f = w^T x + b$$

至于寻找参数 w 和 b，经常用最小二乘法。

我们希望找到一条拟合直线，也就是分类边界，根据数据的特征，把我们想要的不同的类型分隔开来，但是，结果有时会不尽如人意。于是我们就想到找到一个假设函数，来预测分类，这个分类与概率有关，如果该数据预测为 1 的概率大于某个值时，我们可以判别为 1，反之亦然，这就说明我们所需要的这个预测函数值是在 0 与 1 之间，而普通的 $\mathrm{h}_\theta(x)$ 函数存在函数值大于 1 和小于 0 的情况，于是我们要构造一个单调可微函数，可以将分类任务的真实标记 y 与线性回归模型的预测值联系起来且使 $0 \leqslant \mathrm{h}_\theta(x) \leqslant 1$，同时可以得到自变量的权重，从而可以大致了解到底哪些因素是影响结果的。同时根据该权值可以判断其结果的可能性，哪个对结果的影响比较大。

logistic 回归是处理二分类问题的，所以输出的标记 y={0,1}，并且线性回归模型产生的预测值 $\mathrm{z} = w^T x + b$ 是一个实值，所以我们将实值 z 转化成 0/1 值便可，这样有一个可选函数便是“Sigmoid 函数：



这样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

$$y=\frac{1}{1+e^{-w^{T}x+b}} \quad (1)$$

当 y 值≥0.5 时，预测为 1，当 y<0.5 时，预测为 0。我们来做一个变换：

$$\ln\frac{y}{1-y}=w^{T}x+b$$

们将式子中的 y 视为后验概率 p(y=1|x)，则上式可以重写为：

$$\ln\frac{p(y=1\mid x)}{p(y=0\mid x)}=w^{T}x+b$$

因此：$p(y=1\mid x)=\frac{e^{w^{T}x+b}}{1+e^{w^{T}x+b}}=h_{w}(x)$，$p(y=0\mid x)=\frac{1}{1+e^{w^{T}x+b}}=1-h_{w}(x)$，将两个式子合并：$p(y\mid x,w)=h_{w}(x)^{y}(1-h_{w}(x))^{1-y}$ （2）

二.求解权重 W

由于 $h(x)=w_0x_0+w_1x_1+....+w_nx_n=w^{T}x$ 现在我们要找一组 $w_0,w_1,.....,w_n$，使得所有的最接近 h(x),这就是我们要找的权重 W。采用的是梯度下降法。

步骤一：

（1）式似然函数为：$L(W)=\Pi\ h_{w}(x_i)^{y_i}(1-h_{w}(x_i))^{1-y_i}$

取对数再乘以 1/m 为：$J(W)=\frac{1}{m}\sum_{i=1}^{m}\ y_i\ln h_{w}(x_i)+(1-y_i)\ln(1-h_{w}(x_i))$

做个变换，可以梯度下降：$J(W)=-\frac{1}{m}\sum_{i=1}^{m}\ y_i\ln h_{w}(x_i)+(1-y_i)\ln(1-h_{w}(x_i))$（3）

目标是最小化损失函数：$\min_{w}J(w)$

步骤二：

使用梯度下降算法求解参数 w，因此参数 w 的迭代式为：$w_{j+1} = w_j + \alpha \nabla J(w_j)$（4）

其中α学习率，α >0，一般取 0.1,0.01,0.001，看情况而定，也可以最小化目标函数，利用一元函数知道来求解当α为何值时，目标函数值最小，此时的α即为所求最优学习率。

其中对损失函数 J（w）进行微分可得：$\frac{\partial J(w_j)}{\partial w_j} = \nabla J(w_j) = -\frac{1}{m}\sum_{i=1}^{m}(h_w(x_i - y_i)x_j^i)$

所以得到最终参数 w 的迭代式为：$w_{j+1} = w_j + \alpha -\frac{1}{m}\sum_{i=1}^{m}(h_w(x_i - y_i)x_j^i)$

上式将（1/m）去掉不影响结果，于是写成：$w_{j+1} = w_j + \alpha \sum_{i=1}^{m}(h_w(x_i - y_i)x_j^i)$

步骤三：

先看数据集 X

$$x = \begin{bmatrix} x^1 \\ \dots \\ x^m \end{bmatrix} = \begin{bmatrix} x_0^1 & \cdots & x_n^1 \\ \vdots & \ddots & \vdots \\ x_0^m & \cdots & x_n^m \end{bmatrix}$$

其中 m 是数据的个数，有 n 个数据特征，就有 n 维。

然后是标签 y

$$y = \begin{bmatrix} y^1 \\ \dots \\ y^m \end{bmatrix}$$

最后是权重：$W = [w_1 .... w_n]$

根据（3）式。得出权重的迭代公式为：

$$w_{j+1} = w_j + \alpha X^T(\frac{1}{e^{-W^T X}} - Y)$$（5）

以上就是 logistic 回归的数学推导

二．应用实例

（1）关于广告点击的 logistic 回归的分类：

应用 logistic 回归的一般步骤：提取数据，预处理数据，训练模型，得到预测模型，然后是评价模型

a.分析并加载数据集：

本次实验主要使用的数据文件是 train.csv（训练集）,test.csv（测试集）和 subssion(训练集的结果)。数据文件 train.csv 提供了 1599 条的用户访问网页和点

击广告记录的对应特征，l1～l13 为计数特征，c1～c26 为类别特征。Label 表示用户是否点击广告，0 为未点击，1 为点击; 数据文件 test.csv 与 train.csv 类似，提供了 train.csv 之后一段时间的用户访问网页和点击广告记录对应特征。数据文件 subssion.csv，根据测试集给出的用户访问记录，预测出用户点击某个广告的概率，第一列为记录 Id，第二列为用户是否点击广告。

以训练集为例：

	Id	Label	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13	C1
2	10000743	1	1	0	1		227	1	173	18	50	1	7	1		75ac2fe6
3	10000159	1	4	1	1	2	27	2	4	2	2	1	1		2	05db9164
4	10001166	1	0	806			1752	142	2	0	50	0	1			05db9164
5	10000318	0	2	-1	42	14	302	38	25	38	90	1	3		38	05db9164

第一列数据表示的是用户编号，不同编号代表不同的人，这个数据在求解过程中没有使用到。

第二列数据表示用户是否点击广告，1 表示点击，0 表示不点击；

第三到第十五列数据代表用户的计数特征，后续需要以此作为训练数据

我们只需要 train.csv 的 Label 和 I1-I13 的数字特征，用 Lable 作为输出的真实值因变量 Y，I1-I13 的数字特征使我们需要训练模型所需要的自变量。因为数据相差较大，为了防止数据集的方差对结果有较大的影响，我们对取出来的数据集采取归一化处理：程序如下：

```
def loadDataSet():
    dataMatrix = []
    datalabel = []
    style.use('ggplot')
    train = pd.read_csv('train.csv')
    train = train.fillna(0)#把数据中 null 的设为 0
    date= train.ix[:, 2:15]#取出 I1-I13 的数字特征
    label = train.ix[:, 1:2]#取出真实标记
    datalabel = np.mat(label)
    dataMatrix = np.mat(date)
    #对数据进行归一化处理
    minmax_x_train = MinMaxScaler()
    x_train_std = minmax_x_train.fit_transform(dataMatrix)
    dataMatrix = np.mat(x_train_std)
    return dataMatrix,datalabel
```

同样的，测试集和训练集的处理方法是一样的

B:开始对数据进行 logistic 回归分类：

我们需要解出我们所需要的权重 W：

先定义我们所需要的预测函数，sigmoid 函数：

```
def sigmoid(X):
    return 1.0/(1+np.exp(-X))
```

根据（5）式，用梯度下降法求出权重 W，梯度下降法的步骤是：

先初始化每个回归权重，初始化为 1

重复迭代次数 m 次：

计算整个数据集梯度

使用学习率α*梯度更新回归权重

返回回归权重

```
def graAscent(dataMatrix,matLabel,num):
    m,n=np.shape(dataMatrix)#1599,13
    w=np.ones((n,1))#13,1
    alpha=0.01
    for i in range(num):
        E=dataMatrix.transpose()*(sigmoid(dataMatrix*w)-matLabel)#梯度
        w=w-alpha*E
    return w
```

我还写了一个随机梯度下降法求权重的，结果发现没有什么大的差别，以下是随机梯度下降的程序：

```
def stocGraAscent(dataMatrix,matLabel):
    m,n=np.shape(dataMatrix)
    matMatrix=np.mat(dataMatrix)
    w=np.ones((n,1))
    alpha=0.001
    num=1000 #这里的这个迭代次数对于分类效果影响很大，很小时分类效果很差
    for i in range(num):
        for j in range(m):
            error=sigmoid(matMatrix[j]*w)-matLabel[j]
            w=w-alpha*matMatrix[j].transpose()*error
    return w
```

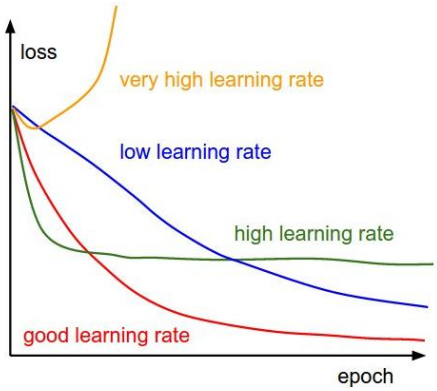
梯度下降算法在每次更新回归系数时都需要遍历整个数据集，该方法在处理小数据时还尚可，但如果有数十亿样本和成千上万的特征，那么该方法的计算复杂度太高了，改进方法便是一次仅用一个数据点来更新回归系数，此方法便称为随机梯度上升算法！由于可以在更新样本到来时对分类器进行增量式更新，因而随机梯度上升算法是一个“在线学习算法”。而梯度上升算法便是“批处理算法”。

C：算出权重 W 了，我们要根据（1）式进行分类，预测值大于 0.5，分为 1，小于等于 0.5 分为 0

```
def predict(w,X):
    m = X.shape[0]#取列数
    Y_prediction = np.zeros((m,1))#初始化预测值，初始化为 0
    A = sigmoid(np.dot(X,w))
    for i in range(A.shape[0]):
        if A[i,0]>0.5:#预测值大于 0.5 判为 1
            Y_prediction[i ,0]=1
        else:
```

```
            Y_prediction[i ,0]=0
    return Y_prediction
```

D：我们根据数据集进行了分类，也要对这个分类器进行评价：分别看两个指标，（1）用真实值减去预测值看其准确率有多少；（2）根据（3）式，看每一次迭代的损失值，是否收敛，这主要看我们的学习率设置的合不合适，因为如果学习率太小，会导致网络 loss 下降非常慢，如果学习率太大，那么参数更新的幅度就非常大，就会导致网络收敛到局部最优点，或者 loss 直接开始增加。就如下图：



在后面调用 loss 那个式子的时候，直接采用矩阵形式运行时，会报出错误，不管用点乘 dot 都不行，就只能把矩阵在转化为数组在运行，这样是可以。

```
def loss(X,Y,num,print_cost=False):
   #costs=loss(weight,dataMatrix,matLabel, num)
    m, n = np.shape(dataMatrix)
    w = np.ones((n, 1))
    alpha = 0.01
    #assert (cost.shape == ())
    costs = []
    print_cost=0
    for i in range(num):
        # 记录成本
        error = sigmoid(dataMatrix * w) - matLabel
        w = w - alpha * dataMatrix.transpose() * error
        A = sigmoid(np.dot(X, w))
        w = np.array(w)#转化为数组便于调用 loss 函数
        A = np.array(A)
        Y= np.array(Y)
        cost = (- 1 / m) * np.sum(Y * np.log(A) + (1 - Y) * (np.log(1 - A)))#logistic 回归的
的损失函数
        if i % 100== 0:
            costs.append(cost)
            print("迭代的次数: %i ， 误差值： %f" % (i, cost))
    return costs
```

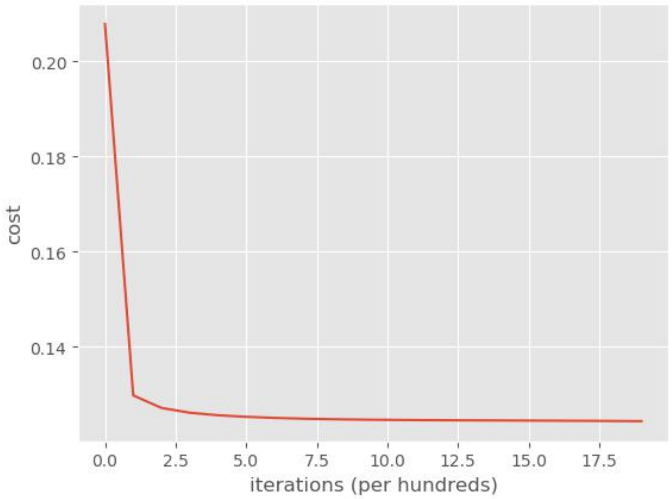
关于准确率：

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

E. 整个程序运行结果为：

我们选用学习率为 0.01：先看 loss 曲线：

这是测试集的：



对于准确率和损失值：

```
迭代的次数: 1300 , 误差值:  0.124547
迭代的次数: 1400 , 误差值:  0.124523
迭代的次数: 1500 , 误差值:  0.124500
迭代的次数: 1600 , 误差值:  0.124475
迭代的次数: 1700 , 误差值:  0.124447
迭代的次数: 1800 , 误差值:  0.124417
迭代的次数: 1900 , 误差值:  0.124383
```

```
训练集的准确度为:  78.48655409631019 %
测试集的准确度为:  80.25 %
```

F. 输出数据：

```
if __name__ == '__main__':
    dataMatrix,matLabel=loadDataSet()
    num=2000
    #weight=graAscent(dataMatrix,matLabel)
    weight= graAscent(dataMatrix,matLabel,num)
    print(weight)
    print(weight.shape)
    dataMatrix1, matLabel1= loadTest()
    #draw(weight)
    y=predict(weight,dataMatrix)
    y1=predict(weight,dataMatrix1)
    print(y.T)
    print(y1.T)
```

```
    print("训练集的准确度为：", format(100 - np.mean(np.abs(y - matLabel) * 100)), "%")
    print("测试集的准确度为：", format(100 - np.mean(np.abs(y1 - matLabel1) * 100)), "%")
    costs = loss(dataMatrix,matLabel,num)
    #costs = np.squeeze(costs)
    plt.plot(costs)
    plt.ylabel('cost')
    plt.xlabel('iterations (per hundreds)')
    #lr=0.01
    #plt.title("学习率：",'lr')
    plt.show()
```

（2）对于 iris 数据集分类：

就是加载文件不一样，其他的都是一样的思路，可以先观察 iris 数据集，它分为 3 类，数据集包含 150 个数据集，分为 3 类，每类 50 个数据，每个数据包含 4 个属性。可通过花萼长度，花萼宽度，花瓣长度，花瓣宽度 4 个属性预测鸢尾花卉属于三个种类中的哪一类。

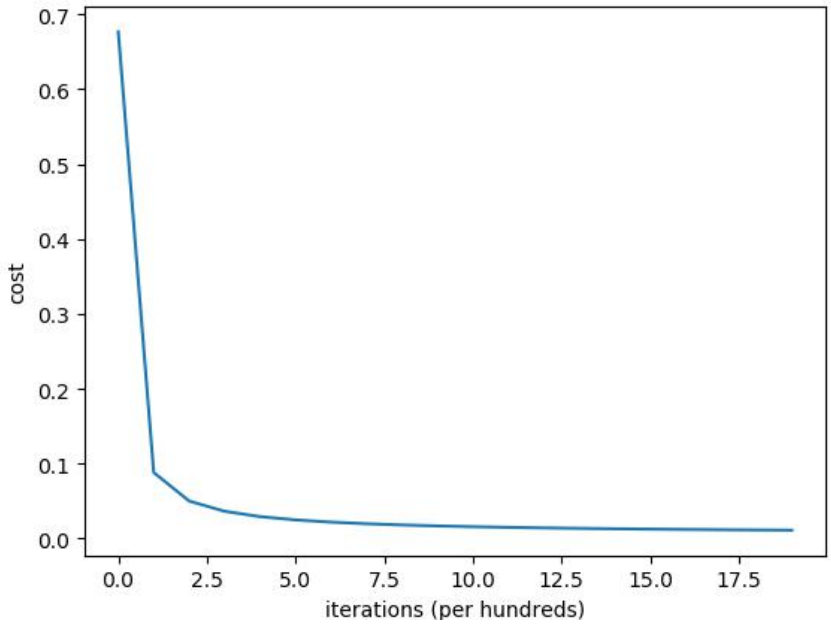
{'data': array([[5.1, 3.5, 1.4, 0.2],
[4.9, 3. , 1.4, 0.2],
[4.7, 3.2, 1.3, 0.2],
[4.6, 3.1, 1.5, 0.2],
'target': array([0, 0, 0,.....,0,1, 1, 1, 1,.....1, 1, 1, 2,..., 2, 2, 2]), 'target_names': array(['setosa', 'versicolor', 'virginica']由 python 加载的数据可以看出，'data' 有 150 行，4 列，代表着鸢尾花的 4 个特征向量，'target' 是有 0,1,2 三类，0 代表山鸢尾花，1 代表 versicolor，2 代表 virginica。

由于我们的 logistic 回归是解决二分类问题，只需要输出值为 0 或 1，所以我们需要舍去后 50 个数据集，程序：

```
def loadDataSet():
    dataMatrix = []
    datalabel = []
    #style.use('ggplot')
    iris = load_iris()
    data = iris.data
    target = iris.target
    X = data[0:100]#取前 100 行，有 4 个特征，100*4
    Y = target[0:100]
    datalabel = np.mat(Y)
    datalabel=np.transpose(datalabel)
    dataMatrix = np.mat(X)
    #对数据进行归一化
    minmax_x_train = MinMaxScaler()
```

```
    x_train_std = minmax_x_train.fit_transform(dataMatrix)
    dataMatrix = np.mat(x_train_std)
    return dataMatrix,datalabel
```

程序运行结果：




```
迭代的次数: 1300 ， 误差值:  0.013332
迭代的次数: 1400 ， 误差值:  0.012792
迭代的次数: 1500 ， 误差值:  0.012320
迭代的次数: 1600 ， 误差值:  0.011902
迭代的次数: 1700 ， 误差值:  0.011530
迭代的次数: 1800 ， 误差值:  0.011197
迭代的次数: 1900 ， 误差值:  0.010895
```

```
  1.  1.  1.  1.]]
准确度为:  100.0 %
迭代的次数: 0 ， 误差值:  0.676901
迭代的次数: 100 ， 误差值:  0.087998
```

三．总结：

本实验主要由两部分组成，一部分是数据处理，另一部分是训练模型。有个奇怪的问题，关于广告点击的实例，训练集发现 cost 的值过大，收敛到 0.5 左右，但是测试集就不会这样，经查资料，在训练（最小化 cost）的过程中，当某一维的特征所对应的权重过大时，而此时模型的预测和真实数据之间距离很小，通过规则化项就可以使整体的 cost 取较大的值，对数据预处理，不管进行归一化，正则化（也为了防止进入局部最小解），规范化，训练集的准确集还有 cost 的值都不变，但测试集结果就很好，查过资料，对于这个数据集，原来的数据集很大，是不是截取的这 1599 个数据不合适？所以用了 iris 数据集来测试这个模型，发现就没有这个问题。那就是数据处理的问题，我目前会的数据处理

只有这三个，请老师指正。通过 logistic 回归，很好的跟原理相结合，对这个分类方法印象很深刻，学了理论再用程序编出来，感觉很不错。