

**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 通信工程

组 员 方观寿

指导教师 赵海涛

**完成日期： 2018 年 10 月23日**

**目 录**

[摘 要 3](#_Toc528183107)

[**一.** **题目简介** 4](#_Toc528183108)

[**二.** **整体解决方案** 4](#_Toc528183109)

[**2.1** **方案设计** 4](#_Toc528183110)

[**2.2** **方案分析** 5](#_Toc528183111)

[**三.** **数据处理** 5](#_Toc528183112)

[**3.1** **数据提取** 5](#_Toc528183113)

[**3.2** **代码实现：** 6](#_Toc528183114)

[**3.3** **数据预处理** 6](#_Toc528183115)

[**3.2.1** **等频离散化处理** 7](#_Toc528183116)

[**3.2.2** **等频离散化处理实现** 8](#_Toc528183117)

[**四.** **预测模型** 8](#_Toc528183118)

[**4.1** **理论推导** 9](#_Toc528183119)

[**4.2** **算法实现** 12](#_Toc528183120)

[**4.1.1** **加载数据** 12](#_Toc528183121)

[**4.1.2** **计算当前损失（正向传播）** 12](#_Toc528183122)

[**4.1.3** **计算当前梯度（反向传播）和更新参数（反向传播）** 14](#_Toc528183123)

[**4.1.4** **预测模型** 15](#_Toc528183124)

[**4.1.5** **训练模型** 15](#_Toc528183125)

[**4.1.6** **绘制结果图** 17](#_Toc528183126)

[**五.** **测试结果和结果分析** 18](#_Toc528183127)

[**总 结** 20](#_Toc528183128)

# 摘 要

经过一个多月课程的模式识别学习，在赵海涛老师的辛勤指导下，对模式识别这个领域有了一定的了解，也激发了对这个领域的兴趣，开始这个领域的探索。本次作业要求是对广告点击率（CTR）进行预测。根据以往网站的点击率数据，获得广告点击率预测模型可以为公司带来巨大的收益。

网站共享了Criteo一周的广告数据，包含：train.csv,test.csv和subssion.csv三个数据文件。关于预测，使用目前课程学过的Logistic Regression进行模型的训练。步骤主要是：提取数据，预处理数据，训练模型，得到预测模型。提取数据采用pandas库可以轻松获取数据；数据预处理较为复杂，通过观察发现，数据分布范围大，很难发现规律，根据网站的提示，考虑对连续特征做等频的离散化处理，对类目特征做one hot encoding；训练模型就用一层BP神经网络进行训练；预测模型可以根据test.csv的数据训练得到结果，再和subssion.csv的结果进行比较得到模型的准确度。

通过几天的探索，最终完成程序的编写及调试，并尝试各种参数的调整以求得到最优的效果。目前，程序跑完时间在30s内。

关键词：训练集， 测试集， 预处理数据， BP神经网络， 预测模型

1. **题目简介**

Criteo是一家第三方展示广告公司，与世界上超过4000家电子商务公司有合作关系。说到广告，关注的最多的就是点击率了。我们经常能听说某某科学家通过建立更好的点击率预测模型，为公司带来上亿的增量收入。本题我们使用Criteo所共享的一周展示广告数据，数据中提炼了13个连续特征、26个离散特征和用户是否点击了该页面广告的标签。请你训练出合适的模型，预测用户在不同的特征下是否会点击广告。

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，第二列为点击概率。使用Logistics回归，在这里就是使用训练集去更新参数，通过设置迭代次数和学习率去不断优化参数，以求得到最好的预测模型。

1. **整体解决方案**
2. **方案设计**

本次作业的操作流程就如下图所示，

建立1层BP神经网络

目标函数使用交叉熵公式

使用梯度下降法更新参数

迭代n次，优化参数

建立预测模型

数据预处理

数据提取

由图可知，该系统主要分为数据处理部分以及训练预测模型两部分。本次实验也将围绕这两个部分进行探究。

1. **方案分析**

首先，数据处理方面经过人眼分析，没有任何的特征，分布杂乱，根据网上的提示需要考虑对连续特征做等频的离散化处理，对类目特征做one hot encoding。

其次，建立预测模型是使用Logistic回归算法，当然处理的方法还有很多，比如：主流的“logistic regression + L1正则”，[Factorization Machine](http://blog.csdn.net/bitcarmanlee/article/details/52143909)模型(FM主要是为了解决数据稀疏的情况下，特征怎样组合的问题)。但无论用何算法，本次实验的核心在于通过已有数据评分预测为评价电影得分。

因此，根据目前所学的知识点来看，我选择了logistic regression方法来预测该模型。Logistic回归为概率型非线性回归模型，是研究二分类观察结果y与一些影响因素X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，比如医学中根据病人的一些症状来判断它是否患有某种病。

1. **数据处理**
   * 1. **数据提取**

首先查看一下原始数据的结构，如下表所示：

|  |  |  |  |
| --- | --- | --- | --- |
| Id | Label | I1~I13 | C1~C26 |
| 10000743 | 1 | 0 1 227 1 173 18 50 1 7 1 | 1cfdf714 |
| 10000159 | 1 | 1 1 2 27 2 4 2 2 1 1 2 | 6c9c9cf3 |
| 10001166 | 0 | 1 0 806 1752 142 2 0 50 0 1 | 05db9164 |

表中的数据只是截取了原始数据中的一部分。

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几乎没有使用到，因为使用矩阵保存数据可以严格保证每行用户和数据相对应，所以可以不用使用该数据。

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；

第十六列到四十一列数据为类别特征。实验会使用这两组数据来训练模型。

最终，考虑到时间以及编程复杂度问题，在原始数据表中我们只采用C1~C13这13列数据作为训练数据来进行模型的训练。

* + 1. **代码实现：**

训练集数据在train.csv文件中，只采用C1~C13的数据作为测试数据，Lable就是要求的Y.测试集数据在test.csv和subssion.csv文件中。使用pandas库的函数从csv文件中取出是DataFrame数据格式，先对多余3个空缺值的数据删除，再使用mean方法求出所有列的平均值，将剩下的空缺值填补成每列的平均值。对于测试集的数据从两个文件夹中取出，再根据Id列使用merge函数将两列数据进行拼合，后面的操作就跟训练集数据一样了。

def load\_dataset():

train\_dataset = pd.read\_csv('train.csv',parse\_dates=True,usecols=range(1,15))

train\_dataset = train\_dataset.dropna(thresh=11)

for column in list(train\_dataset.columns):

mean\_val = train\_dataset[column].mean()

train\_dataset[column].fillna(mean\_val,inplace=True)

train\_dataset = train\_dataset.values

train\_set\_x\_orig = train\_dataset[:,1:15]

train\_set\_y\_orig = train\_dataset[:,0:1]

test\_dataset1 = pd.read\_csv('test.csv',parse\_dates=True,usecols=range(0,14))

test\_dataset2 = pd.read\_csv('submission.csv',parse\_dates=True,usecols=range(0,2))

test\_dataset = pd.merge(test\_dataset2,test\_dataset1,on='Id')

test\_dataset = test\_dataset.dropna(thresh=11)

for column in list(test\_dataset.columns):

mean\_val = test\_dataset[column].mean()

test\_dataset[column].fillna(mean\_val,inplace=True)

test\_dataset = test\_dataset.values

print(test\_dataset)

test\_set\_x\_orig = test\_dataset[:,2:16]

test\_set\_y\_orig = test\_dataset[:,1:2]

return train\_set\_x\_orig,train\_set\_y\_orig,test\_set\_x\_orig,test\_set\_y\_orig

* + 1. **数据预处理**

如前所述，数据过于离散且分布人工没法得出模型，需要对数据进行预处理，根据网上的提示，考虑对连续特征做等频的离散化处理，对类目特征做one hot encoding。

1. **等频离散化处理**

在特征工程中，特别是logistic regression上，需要把一些连续特征进行离散化处理。离散化除了一些计算方面等等好处，还可以引入非线性特性，也可以很方便的做cross-feature。离散特征的增加和减少都很容易，易于模型的快速迭代。此外，噪声很大的环境中，离散化可以降低特征中包含的噪声，提升特征的表达能力。

连续特征离散化最常用的方法：

（1）划分区间

如1-100岁可以划分为：（0-18）未成年、（18-50）中青年、（50-100）中老年.

这其中包括等距划分、按阶段划分、特殊点划分等。

（2）卡方检验（CHI）

分裂方法，就是找到一个分裂点看，左右2个区间，在目标值上分布是否有显著差异，有显著差异就分裂，否则就忽略。这个点可以每次找差异最大的点。合并类似，先划分如果很小单元区间，按顺序合并在目标值上分布不显著的相邻区间，直到收敛。卡方值通常由χ2分布近似求得。

χ2表示观察值与理论值之问的偏离程度。计算这种偏离程度的基本思路如下，其计算公式为：

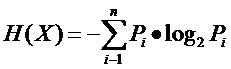
\chi^2=\sum \frac{(A-E)^2}{E}=\sum_{i=1}^k \frac{(A_i-E_i)^2}{E_i}=\sum_{i=1}^k \frac{(A_i-np_i)^2}{np_i}　　(i=1，2，3，…，k)

当n比较大时，χ2统计量近似服从k-1(计算Ei时用到的参数个数)个自由度的卡方分布。

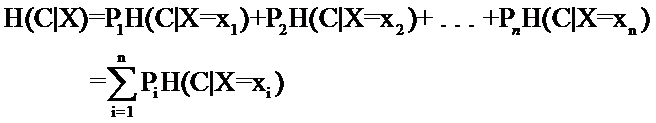
（3）信息增益法（IG）

这个和决策树的学习很类似。分裂方法，就是找到一个分裂点看，左右2个区间，看分裂前后信息增益变化阈值，如果差值超过阈值（正值，分列前-分裂后信息熵），则分裂。每次找差值最大的点做分裂点，直到收敛。合并类似，先划分如果很小单元区间，按顺序合并信息增益小于阈值的相邻区间，直到收敛。，信息增益为总的熵减去某个分类标准对应的熵。

熵：



条件熵：



1. **等频离散化处理实现**

对连续型特征进行预处理，即对连续型特征进行标准化，一种常用的标准化方法是将特征缩放至给定的最小值和最大值之间，通常在0和1之间，可以使用MinMaxScaler类实现。

思路如下：将所有的连续型特征取出来放在一起进行标准化，要注意区分正向指标和负向指标

代码如下：

def normorlize\_data(matrix\_data\_orig):

#将数据集进行归一化处理

scaler = MinMaxScaler( )

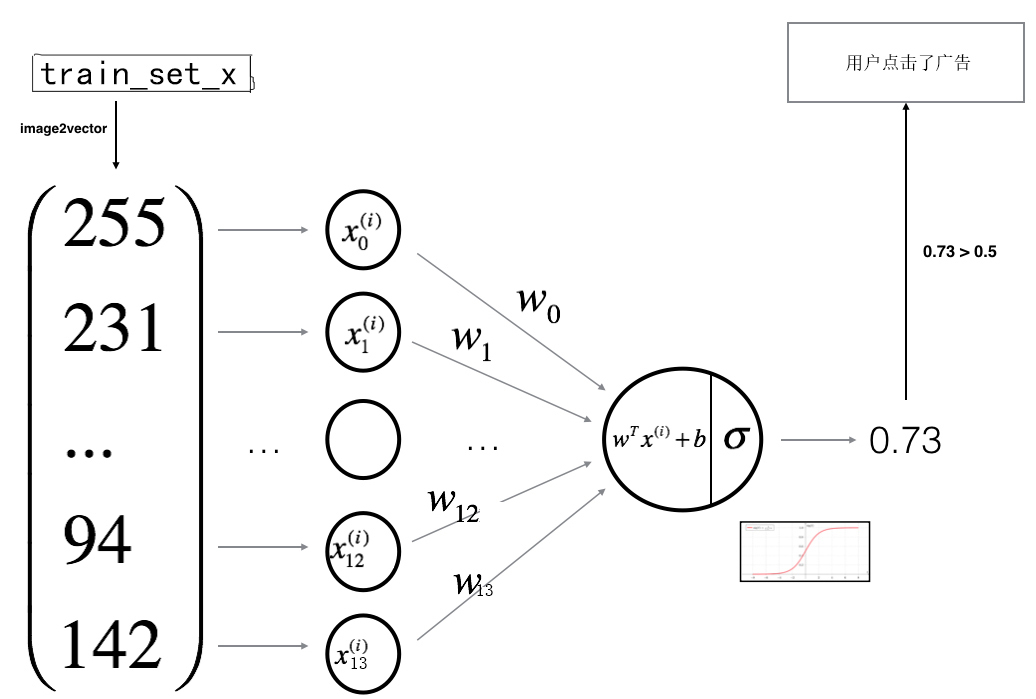
scaler.fit(matrix\_data\_orig)

scaler.data\_max\_

return scaler.transform(matrix\_data\_orig)

1. **预测模型**

预测模型使用一层BP神经网络实现，原理图如下：



从图中可以看出，我们是把需要进行训练的数据从左边输入，在这个实验中，维度是13，经过数据预处理后输入到系统中。经过一个计算：w\*x+b后得到一个输出a,再经过激励函数sigmoid函数，获得一个0到1之间的输出，通过对该输出判断是否大于0.5，大于0.5判为1，也即用户点击了该广告，若小于0.5，判为用户未点击广告。

* 1. **理论推导**

下面正式地来讲Logistic回归模型。

考虑具有n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

这里称为Logistic函数。其中

那么在x条件下y不发生的概率为

所以事件发生与不发生的概率之比为



这个比值称为事件的发生比（the odds of experiencing an event），简记为odds。

对odds取对数得到



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m个观测样本，观测值分别为y1,y2,y3,…,ym，设为给定条件下得到yi=1的概率，同样地，yi=0的概率为，所以得到一个观测值的概率为。

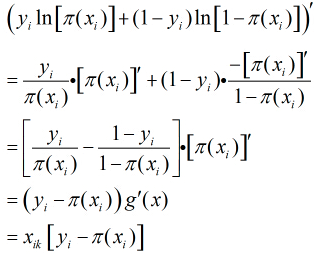
因为各个观测样本之间相互独立，那么它们的联合分布为各边缘分布的乘积。得到似然函数为



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w0,w1,w2,….,wn，使得L(w)取得最大值，对函数L(w)取对数得到



继续对这n+1个wi分别求偏导，得到n+1个方程，比如现在对参数wk求偏导，由于



所以得到



这样的方程一共有n+1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，一般方法似乎不能解之，所以我们引用了牛顿-拉菲森迭代方法求解。

实际上在上述似然函数求最大值时，可以用梯度上升[算法](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库" \t "_blank)，一直迭代下去。梯度上升算法和牛顿迭代相比，收敛速度慢，因为梯度上升算法是一阶收敛，而牛顿迭代属于二阶收敛,梯度下降。当然求最小值就是梯度下降，而求最大值相对就是梯度上升。

由于，如果，那么得到

现在我们要找一组，使得所有的最接近，设

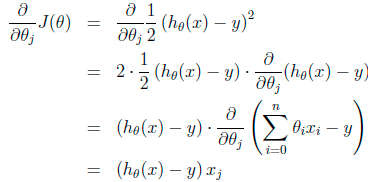
现在我们要找一组，使得最小。这就是今天要介绍的梯度下降。

梯度下降的原理是这样的：首先对于向量赋初值，可以赋随机值，也可以全赋为0，然后改变的值，使得

按梯度下降最快的方向进行，一直迭代下去最终会得到局部最小值。即



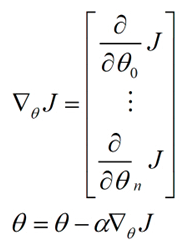
表示梯度最陡的那个方向，表示步长，也就是说每次向下降最快的方向走多远。进一步有



所以



简化一下就是



* 1. **算法实现**

1. **加载数据**

from load\_data import load\_dataset

train\_set\_x\_orig , train\_set\_y , test\_set\_x\_orig , test\_set\_y = load\_dataset()

1. **计算当前损失（正向传播）**

1）计算当前损失，也就是1层神经网络的前向传播过程，这里使用sigmoid作为激励函数，激励函数如下：



实现代码：

def sigmoid(z):

"""

参数：

z - 任何大小的标量或numpy数组。

返回：

s - sigmoid（z）

"""

s = 1 / (1 + np.exp(-z))

return s

2）初始化我们参数w和b

def initialize\_parameter(dim):

"""

此函数为w创建一个维度为（dim，1）的向量，并将b初始化为0。

参数：

dim - 我们想要的w矢量的大小（或者这种情况下的参数数量）

返回：

w - 维度为（dim，1）的初始化向量。

b - 初始化的标量（对应于偏差）

"""

w = np.random.random(shape = (dim,1))#将w随机初始化为0-1之间的浮点数值

b = 0

#使用断言来确保我要的数据是正确的

assert(w.shape == (dim, 1)) #w的维度是(dim,1)

assert(isinstance(b, float) or isinstance(b, int)) #b的类型是float或者是int

return (w , b)

3）前向传播和反向传播

def propagate(w, b, X, Y):

"""

实现前向和后向传播的成本函数及其梯度。

参数：

w - 权重，大小不等的数组（num\_px \* num\_px \* 3，1）

b - 偏差，一个标量

X - 矩阵类型为（num\_px \* num\_px \* 3，训练数量）

Y - “标签”矢量（点击则为0，不点击则为1），矩阵维度为(1,训练数据数量)

返回：

cost- 逻辑回归的负对数似然成本

dw - 相对于w的损失梯度，因此与w相同的形状

db - 相对于b的损失梯度，因此与b的形状相同

"""

m = X.shape[1]

#正向传播

A = sigmoid(np.dot(w.T,X) + b) #计算激活值

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

#反向传播

dw = (1 / m) \* np.dot(X, (A - Y).T)

db = (1 / m) \* np.sum(A - Y)

#使用断言确保我的数据是正确的

assert(dw.shape == w.shape)

assert(db.dtype == float)

cost = np.squeeze(cost)

assert(cost.shape == ())

#创建一个字典，把dw和db保存起来。

grads = {

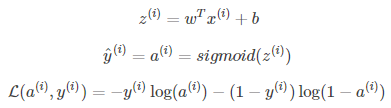
"dw": dw,

"db": db

}

return (grads , cost)

y\_hat和损失函数的公式如下，cost损失函数（目标函数）就是使用Logistic回归函数。由推断可得：dw = (1/m)(X\*(A-Y).T)和db = (1/m)Σ（A-Y）。



1. **计算当前梯度（反向传播）和更新参数（反向传播）**

目标是通过最小化成本函数*J*来学习W和B。对于参数*θ*，更新规则是θ=θ−α\*dθ,其中α是学习率。

def optimize(w , b , X , Y , num\_iterations , learning\_rate , print\_cost = False):

"""

此函数通过运行梯度下降算法来优化w和b

参数：

w - 权重，大小不等的数组（num\_px \* num\_px \* 3，1）

b - 偏差，一个标量

X - 维度为（num\_px \* num\_px \* 3，训练数据的数量）的数组。

Y - “标签”矢量（点击则为1，不点击则为0），矩阵维度为(1,训练数据的数量)

num\_iterations - 优化循环的迭代次数

learning\_rate - 梯度下降更新规则的学习率

print\_cost - 每100步打印一次损失值

返回：

params - 包含权重w和偏差b的字典

grads - 包含权重和偏差相对于成本函数的梯度的字典

成本 - 优化期间计算的所有成本列表，将用于绘制学习曲线。

提示：

我们需要写下两个步骤并遍历它们：

1）计算当前参数的成本和梯度，使用propagate（）。

2）使用w和b的梯度下降法则更新参数。

"""

costs = []

for i in range(num\_iterations):

grads, cost = propagate(w, b, X, Y)

dw = grads["dw"]

db = grads["db"]

w = w - learning\_rate \* dw

b = b - learning\_rate \* db

#记录成本

if i % 100 == 0:

costs.append(cost)

#打印成本数据

if (print\_cost) and (i % 100 == 0):

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

params = {

"w" : w,

"b" : b }

grads = {

"dw": dw,

"db": db }

return (params , grads , costs)

可以看到，更新参数是用预先定义好的超参数α(也就是学习率)，w = w-α\*dw;同理有b = b-α\*db.

1. **预测模型**

def predict(w , b , X ):

"""

使用学习逻辑回归参数logistic （w，b）预测标签是0还是1，

参数：

w - 权重，大小不等的数组（num\_px \* num\_px \* 3，1）

b - 偏差，一个标量

X - 维度为（num\_px \* num\_px \* 3，训练数据的数量）的数据

返回：

Y\_prediction - 包含X中所有图片的所有预测【0 | 1】的一个numpy数组（向量）

""

m = X.shape[1] #图片的数量

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

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

#计预测猫在图片中出现的概率

A = sigmoid(np.dot(w.T , X) + b)

for i in range(A.shape[1]):

#将概率a [0，i]转换为实际预测p [0，i]

Y\_prediction[0,i] = 1 if A[0,i] > 0.5 else 0

#使用断言

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

return Y\_prediction

通过训练集训练得出的参数来构造模型，然后使用训练集或者测试集进行模型的预测。

1. **训练模型**

def model(X\_train , Y\_train , X\_test , Y\_test , num\_iterations = 2000 , learning\_rate = 0.5 , print\_cost = False):

"""

通过调用之前实现的函数来构建逻辑回归模型

参数：

X\_train - numpy的数组,维度为（num\_px \* num\_px \* 3，m\_train）的训练集

Y\_train - numpy的数组,维度为（1，m\_train）（矢量）的训练标签集

X\_test - numpy的数组,维度为（num\_px \* num\_px \* 3，m\_test）的测试集

Y\_test - numpy的数组,维度为（1，m\_test）的（向量）的测试标签集

num\_iterations - 表示用于优化参数的迭代次数的超参数

learning\_rate - 表示optimize（）更新规则中使用的学习速率的超参数

print\_cost - 设置为true以每100次迭代打印成本

返回：

d - 包含有关模型信息的字典。

"""

w , b = initialize\_with\_zeros(X\_train.shape[0])

parameters , grads , costs = optimize(w , b , X\_train , Y\_train,num\_iterations , learning\_rate , print\_cost)

#从字典“参数”中检索参数w和b

w , b = parameters["w"] , parameters["b"]

#预测测试/训练集的例子

Y\_prediction\_test = predict(w , b, X\_test)

Y\_prediction\_train = predict(w , b, X\_train)

#打印训练后的准确性

print("训练集准确性：" , format(100 - np.mean(np.abs(Y\_prediction\_train - Y\_train)) \* 100) ,"%")

print("测试集准确性：" , format(100 - np.mean(np.abs(Y\_prediction\_test - Y\_test)) \* 100) ,"%")

d = {

"costs" : costs,

"Y\_prediction\_test" : Y\_prediction\_test,

"Y\_prediciton\_train" : Y\_prediction\_train,

"w" : w,

"b" : b,

"learning\_rate" : learning\_rate,

"num\_iterations" : num\_iterations }

return d

构建好模型的前向传播和后向传播，激励函数，梯度下降，数据提取等就可以进行模型的训练。模型训练也是一个不断优化参数的过程。

1. **绘制结果图**

learning\_rates = [0.01, 0.001, 0.0001]

models = {}

for i in learning\_rates:

print ("learning rate is: " + str(i))

models[str(i)] = model(train\_set\_x, train\_set\_y, test\_set\_x, test\_set\_y, num\_iterations = 1500, learning\_rate = i, print\_cost = False)

print ('\n' + "-------------------------------------------------------" + '\n')

for i in learning\_rates:

plt.plot(np.squeeze(models[str(i)]["costs"]), label= str(models[str(i)]["learning\_rate"]))

plt.ylabel('cost')

plt.xlabel('iterations')

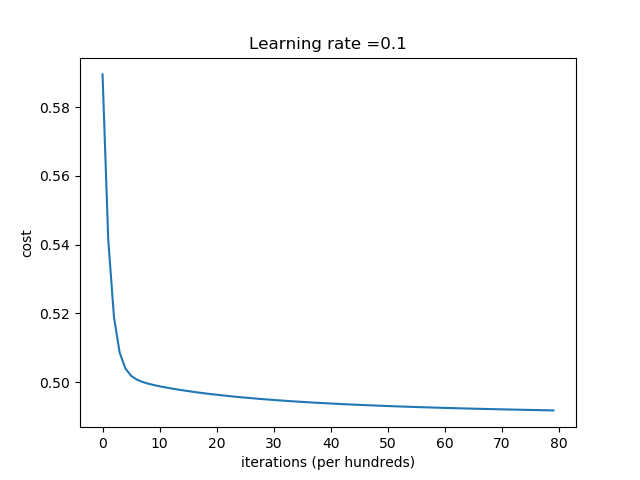
legend = plt.legend(loc='upper center', shadow=True)

frame = legend.get\_frame()

frame.set\_facecolor('0.90')

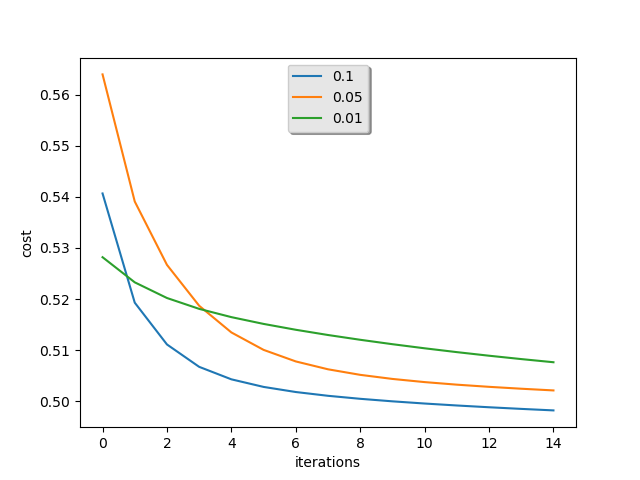
plt.show()

1. **测试结果和结果分析**



上图所示是迭代8000次，cost的曲线，由图可以看到从刚开始的0.63到第100次跌倒的0.49快速下降；但当过了100次迭代以后，cost下降速度并不明显。

为了得到较好的预测结果，本次实验参数多次调整，但最终cost仍难不够理想。下图所示是三种不同的学习率产生的cost曲线，从结果可以看到，cost后面下降并不明显，问题一度陷入尴尬。



在最终的程序调试中，主要通过修改学习率和w,b的初始化，最重要的是改变数据预处理方式。但最终，就结果还是不尽人意。

<pre>====================测试model====================

迭代的次数: 0 ， 误差值： 0.620260

迭代的次数: 100 ， 误差值： 0.556587

迭代的次数: 200 ， 误差值： 0.552114

迭代的次数: 300 ， 误差值： 0.548585

迭代的次数: 400 ， 误差值： 0.545338

迭代的次数: 500 ， 误差值： 0.542344

迭代的次数: 600 ， 误差值： 0.539583

迭代的次数: 700 ， 误差值： 0.537039

迭代的次数: 800 ， 误差值： 0.534696

迭代的次数: 900 ， 误差值： 0.532538

迭代的次数: 1000 ， 误差值： 0.530551

迭代的次数: 1100 ， 误差值： 0.528723

迭代的次数: 1200 ， 误差值： 0.527041

迭代的次数: 1300 ， 误差值： 0.525493

迭代的次数: 1400 ， 误差值： 0.524070

迭代的次数: 1500 ， 误差值： 0.522761

迭代的次数: 1600 ， 误差值： 0.521558

迭代的次数: 1700 ， 误差值： 0.520451

迭代的次数: 1800 ， 误差值： 0.519433

迭代的次数: 1900 ， 误差值： 0.518497

迭代的次数: 2000 ， 误差值： 0.517635

训练集准确性： 77.11069418386492 %

测试集准确性： 80.56426332288402 %

learning rate is: 0.1

训练集准确性： 76.92307692307692 %

测试集准确性： 79.62382445141066 %

-------------------------------------------------------

learning rate is: 0.05

训练集准确性： 76.92307692307692 %

测试集准确性： 79.62382445141066 %

-------------------------------------------------------

learning rate is: 0.01

训练集准确性： 76.92307692307692 %

测试集准确性： 79.62382445141066 %

仔细查看每次的cost结果，显示确实没有很好的训练好模型。Cost超过100次迭代后，下降就很不明显了。最难以接受的问题是，训练集的测试结果竟然比训练集的测试结果要好。

**总 结**

经过1个多星期的学习和研究，这个实验的模型就算建立起来了，但针对这个广告点击率的问题却并没有得到很大的解决。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。刚开始，我以为构建模型是最重要的，所以对数据只是进行简单的处理，就是将所有数据集的空缺补上0，然后再将所有数据归一化，最后形成一个由0-1组成的1599\*13维度的矩阵。测试出来的结果很不理想，刚开始怀疑自己的模型有问题，但是使用其他数据集进行测试，得到的结果却是很好，而且各方面的理论推导和代码实现都不存在问题。思来想去，还是得好好处理数据。

数据处理成了这次实验最大的难题，数据分布过于复杂，且数据量也不是人工可以进行审核的。经过同学的指导，首先将超过3个空缺值的数据删掉，其次，将剩余的空缺项用每列的平均值来替换。结果显示，准确度下降了，同时测试集的准确度也下降了。这不得不怀疑是不是数据本身存在问题，经过同学的讨论，对数据进行了一系列的统计，结果发现：

(1).训练样本中，Lable项1和0的比例是1：4；换句话说，就算是进行伯努利实验，得到1的概率是20%，得到0的概率是80%。这也就可以解释为什么测试集的准确率总是80%，一点都不差。

(2).数据集后面26个数据，网上说是26个离散的类别特征，但经过统计，竟然存在802个不同的类型，这对只有26个特征的数据来说不太合理，而且给的样本也只有1559个，对于802个特征，这实在让人费解。

因为存在1：4的样本比率，所以我觉得这个样本是存在一些问题的，至少对于我现在的知识来说，解决还是有很大的困难。但是，这并不影响我对Logistic回归的理解以及应用。通过这次作业，真正的激发我对模式识别这门课的浓厚兴趣。Logistics对二分类问题的处理还是有很高的地位，相对来说，在牺牲少量的准确度的情况下，Logistics回归在速度上有很大的优势。学无止境，这条路上想必是艰辛和乐趣并存，本次作业，感谢对