

**模式识别大作业**

题 目 Logistic回归

学 院 信息科学与工程

专 业 信息与通信工程

组 员 季思凡

指导教师 赵海涛

**完成日期： 2018 年 10 月24日**

**模式识别作业报告——**Logistic回归

组员：季思凡

经过数周的模式识别课程的学习，在赵老师的辛勤教导、在与同学们的积极讨论、在网络上的自发学习中，本人也终于对logistic回归算法有初步的认识和了解，并且准备通过本次针对猫的数据集的训练来加深自己对该算法的理解，同时巩固自己所学习的内容。

经过一段时间的程序编写、修改与调试后，尝试了一些优化算法以及其他方法之后，将准确率稍稍提高了百分之10左右，以下是我的代码和思路的详细介绍。

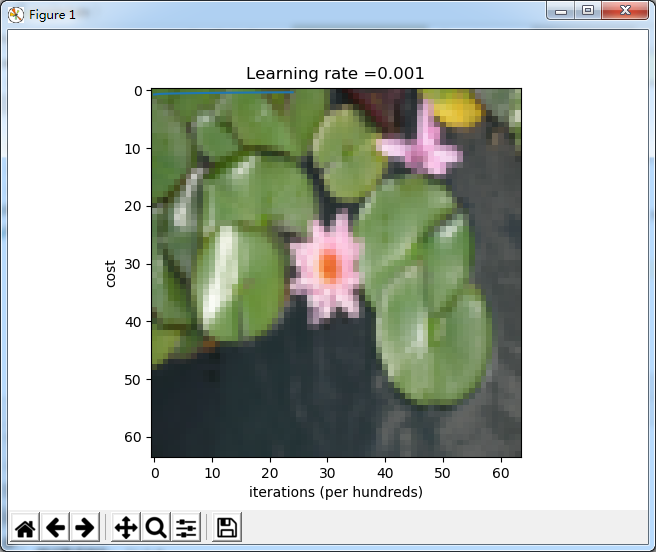
1. **整体结局方案**

**2.1数据集的介绍**

本次练习所用的数据集分为训练集和测试集，由于存放的格式为hdf5，所以可以通过以下代码来查看训练集中的图像：

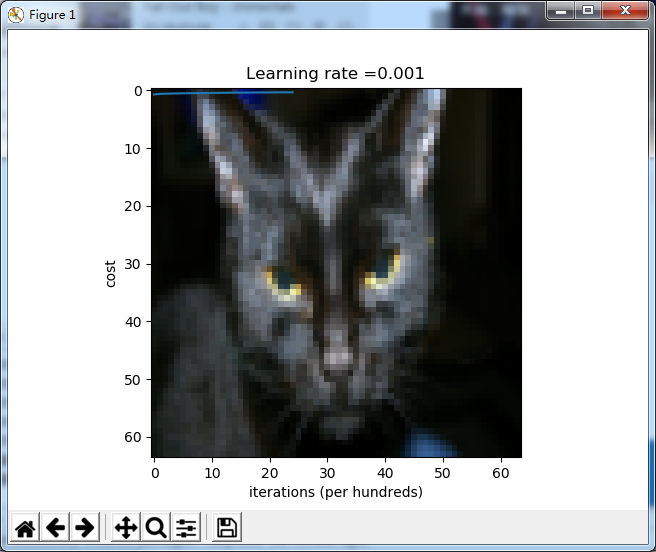
|  |
| --- |
| index = 26  plt.imshow(train\_set\_x\_orig[index]) |

运行结果如下：



|  |
| --- |
| index = 25  plt.imshow(train\_set\_x\_orig[index]) |

运行结果如下：



所以，同时在此次练习的训练集中我们一共包含了209张图片，以及50张测试集的图片，每一幅图也都是64\*64\*3像素的。

**2.2加载资料包中的数据**

在安装完必要的库之后，便开始对数据进行加载，代码如下：

|  |
| --- |
| import numpy as np  import h5py  def load\_dataset():  train\_dataset = h5py.File('datasets/train\_catvnoncat.h5', "r")  train\_set\_x\_orig = np.array(train\_dataset["train\_set\_x"][:]) # your train set features  train\_set\_y\_orig = np.array(train\_dataset["train\_set\_y"][:]) # your train set labels  test\_dataset = h5py.File('datasets/test\_catvnoncat.h5', "r")  test\_set\_x\_orig = np.array(test\_dataset["test\_set\_x"][:]) # your test set features  test\_set\_y\_orig = np.array(test\_dataset["test\_set\_y"][:]) # your test set labels  classes = np.array(test\_dataset["list\_classes"][:]) # the list of classes  train\_set\_y\_orig = train\_set\_y\_orig.reshape((1, train\_set\_y\_orig.shape[0]))  test\_set\_y\_orig = test\_set\_y\_orig.reshape((1, test\_set\_y\_orig.shape[0]))  return train\_set\_x\_orig, train\_set\_y\_orig, test\_set\_x\_orig, test\_set\_y\_orig, classes |

为了方便，我们要把维度为（64，64，3）的numpy数组重新构造为（64 x 64 x 3，1）的数组,在此之后，我们的训练和测试数据集是一个numpy数组，然后通过如下代码查看降维效果：

|  |
| --- |
| print ("训练集降维最后的维度： " + str(train\_set\_x\_flatten.shape))  print ("训练集\_标签的维数 : " + str(train\_set\_y.shape))  print ("测试集降维之后的维度: " + str(test\_set\_x\_flatten.shape))  print ("测试集\_标签的维数 : " + str(test\_set\_y.shape)) |

运行结果：

|  |
| --- |
| 训练集降维最后的维度： (12288, 209)  训练集\_标签的维数 : (1, 209)  测试集降维之后的维度: (12288, 50)  测试集\_标签的维数 : (1, 50) |

**2.3图像的标准化**

为了表示彩色图像，必须为每个像素指定红色，绿色和蓝色通道（RGB），因此像素值实际上是从0到255范围内的三个数字的向量。机器学习中一个常见的预处理步骤是对数据集进行居中和标准化，这意味着可以减去每个示例中整个numpy数组的平均值，然后将每个示例除以整个numpy数组的标准偏差。但对于图片数据集，它更简单，更方便，几乎可以将数据集的每一行除以255（像素通道的最大值），因为在RGB中不存在比255大的数据，所以我们可以放心的除以255，让标准化的数据位于[0,1]之间，通过如下代码标准化数据集：

|  |
| --- |
| train\_set\_x = train\_set\_x\_flatten / 255  test\_set\_x = test\_set\_x\_flatten / 255 |

这里也用到了在模式识别课程中学习过的神经网络，但是只有一层，但若在后续的优化过程中，想要提高准确率，降低误差率，可以通过多层的神经网络来达到效果，譬如在后续的调试修改中，便可用到5层神经网络，最后的准确率也相对于之前提高了百分15。

**2.4循环迭代**

构建sigmoid()，需要使用 sigmoid（w ^ T x + b） 计算来做出预测，代码如下：

|  |
| --- |
| def sigmoid(z):  s = 1 / (1 + np.exp(-z))  return s |

经测试，sigmoid函数值符合要求，接着初始化需要的参数w和b，以及执行“前向”和“后向”传播步骤来学习参数

|  |
| --- |
| def initialize\_with\_zeros(dim):  w = np.zeros(shape = (dim,1))  b = 0    assert(w.shape == (dim, 1))  assert(isinstance(b, float) or isinstance(b, int))  return (w , b)  def propagate(w, b, X, Y):    m = X.shape[1]  #正向传播  A = sigmoid(np.dot(w.T,X) + b)  cost = (- 1 / m) \* np.sum(Y \* np.log(A) + (1 - Y) \* (np.log(1 - A)))  #反向传播  dw = (1 / m) \* np.dot(X, (A - Y).T)  db = (1 / m) \* np.sum(A - Y)    assert(dw.shape == w.shape)  assert(db.dtype == float)  cost = np.squeeze(cost)  assert(cost.shape == ())    grads = {  "dw": dw,  "db": db  }  return (grads , cost) |

所以initialize\_with\_zeros（）和propagate（）函数分别代表初始化函数和学习函数。

**2.5参数的更新**

目标是通过最小化成本函数来学习和。对于参数，更新规则是，其中 是学习率，代码如下：

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

**2.6值的预测**

optimize函数会输出已学习的w和b的值，我们可以使用w和b来预测数据集X的标签。

现在我们要实现预测函数predict（）。计算预测有两个步骤：

1. 计算
2. 将a的值变为0（如果激活值<= 0.5）或者为1（如果激活值> 0.5），然后将预测值存储在向量Y\_prediction中。

以下是预测函数：

|  |
| --- |
| def predict(w , b , X ):  m = X.shape[1]  Y\_prediction = np.zeros((1,m))  w = w.reshape(X.shape[0],1)  A = sigmoid(np.dot(w.T , X) + b)  for i in range(A.shape[1]):  Y\_prediction[0,i] = 1 if A[0,i] > 0.5 else 0  assert(Y\_prediction.shape == (1,m))  return Y\_prediction |

**2.7主函数**

将上述函数写入主函数中：

|  |
| --- |
| def model(X\_train , Y\_train , X\_test , Y\_test , num\_iterations = 2000 , learning\_rate = 0.5 , print\_cost = False):  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)  w , b = parameters["w"] , parameters["b"]  Y\_prediction\_test = predict(w , b, X\_test)  Y\_prediction\_train = predict(w , b, X\_train)  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 |

**2.1算法原理及公式**

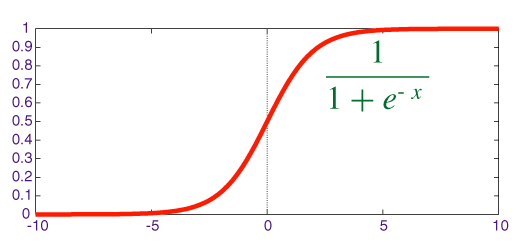
对于logistic回归一般公式如下：



其中：



所以，sigmoid的函数图形如下：



我们可以看到，sigmoid的函数输出是介于（0，1）之间的，中间值是0.5，于是之前的公式 hθ(x)hθ(x) 的含义就很好理解了，因为 hθ(x)hθ(x) 输出是介于（0，1）之间，也就表明了数据属于某一类别的概率，例如 ：

hθ(x)hθ(x)<0.5 则说明当前数据属于A类；

hθ(x)hθ(x)>0.5 则说明当前数据属于B类。

所以我们可以将sigmoid函数看成样本数据的概率密度函数，有了上面的公式，我们接下来需要做的就是怎样去估计参数 θθ 了。

首先我们来看， θθ 函数的值有特殊的含义，它表示 hθ(x)hθ(x) 结果取1的概率，因此对于输入x分类结果为类别1和类别0的概率分别为：





根据上式，接下来我们可以使用概率论中极大似然估计的方法去求解损失函数，首先得到概率函数为：



因为样本数据(m个)独立，所以它们的联合分布可以表示为各边际分布的乘积,取似然函数为：





取对数似然函数：





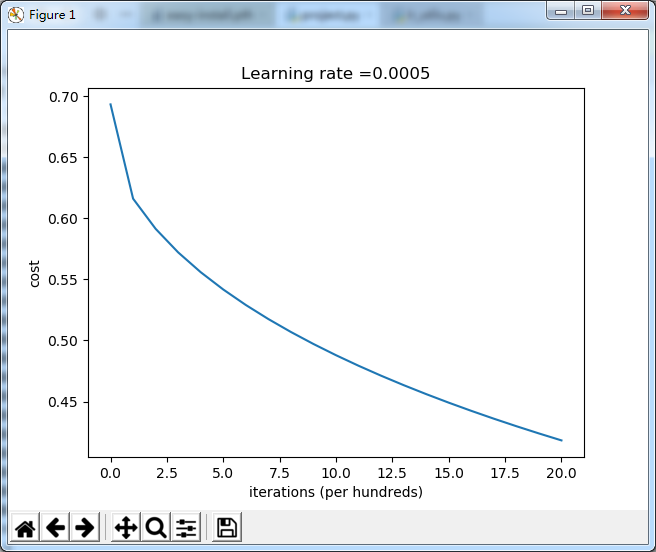
最大似然估计就是要求得使取最大值时的，这里可以使用梯度上升法求解。我们稍微变换一下：



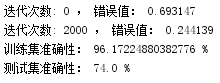
因为乘了一个负的系数，然后就可以使用梯度下降算法进行参数求解了。

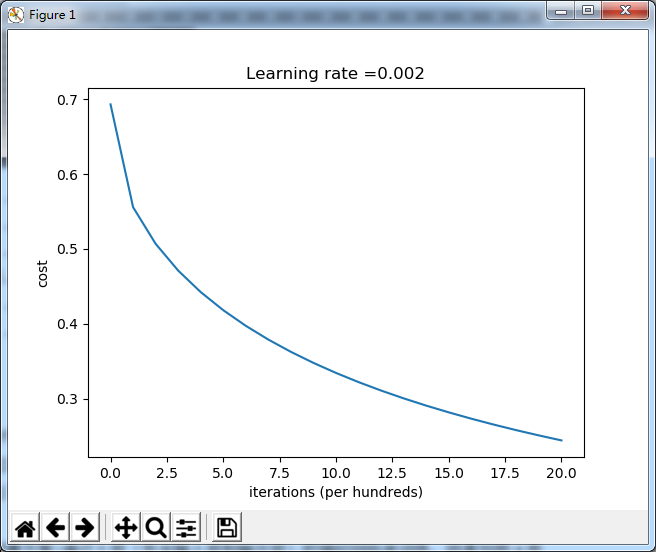
**3.1程序结果及分析**





从结果可以得出，虽然训练集的准确率很高，但测试集的准确率只有66%，所以在此思考了如何优化算法的方法。当然通过增加数据集的数量可以明显的增加测试集以及训练集的准确率，但如何在数据集不变的情况下增加准确率和减小误差值成为了值得思考的问题，在本次练习中，便出现了两种解决方案，第一种是通过增加神经网络的层数，虽然算法更加复杂，但是可以显著的提高准确率；第二种则是改变一些参数值，也可以提高最后结果的准确率，所以在测试实验后，结果如下：





1. **小组分工**

程序设计及编写：季思凡

程序调试：季思凡

实验报告：季思凡

1. **作业总结**

在本次练习初，也遇到了许多麻烦，由于是第一次接触logistic回归以及神经网络的领域，在开始编写代码的时候，也是无从下手，但是随着模式识别课程的深入，以及通过在网上搜索资料与同学进行分享讨论，我对于这门课程的理解也是更进一层，对于代码的编写也有了灵感，当然也从中参照了网上的一些例子。

最后，在对代码的不断调试与修改的过程中，也使得准确率达到了80%左右，并且在深度学习这块也算是打下了一定的基础，产生了兴趣。