## 目录

**一、实验原理 1**

1、图片数据 1

2、线性拟合（逻辑回归） 2

3、Sigmod函数 2

4、损失函数（交叉熵） 3

5、成本函数 3

6、梯度下降 4

**二、实验代码流程图 7**

**三、代码讲解 8**

1、导入数据 8

2、Sigmoid函数 9

3、初始化参数 9

4、计算y的至于和成本值 9

5、梯度下降法找最优值 10

6、预测结果 11

7、建立预测模型 12

8、初始化数据，得到结果 12

9、画出学习曲线 14

10、不同学习率的学习曲线 14

**四、实验结果分析 15**

1、训练集和测试集的准确度 15

2、不同学习率时的准确度 15

3、学习曲线 16

4、不同学习率的学习曲线 16

**五、总结与感想 17**

**六、本课程的收获与建议 17**

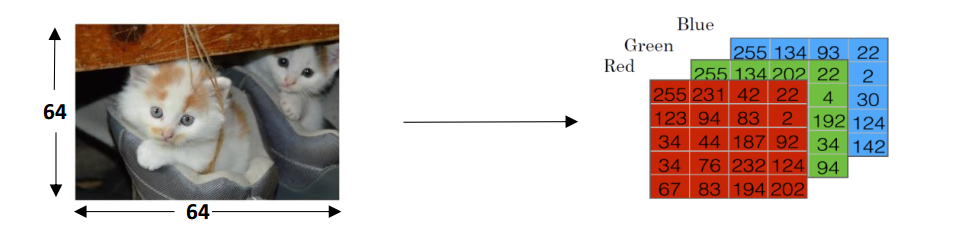
1、收获 17

2、建议 17

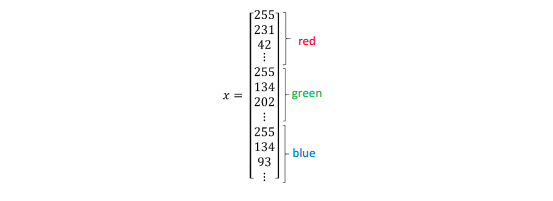
## 实验原理

### 1、图片数据

图片是一类非结构化的数据，在计算机中，一张图片以RGB编码方式存储时，是将它以红、绿、蓝为三基色，以每个像素点上三种颜色的十六进制颜色码进行编码，形成一个包含三个**信道（Channel）**，各信道大小和图片的大小相同的矩阵。例如图中的猫图大小为，那么表示为矩阵后的每个信道大小即为。



**模式识别（Pattern Recognition）**以及机器学习中，处理的各种类型的数据都需要用一些特征向量来表示。为了将上例中的图片表示为一个特征向量，将每三个信道中三种颜色的值进行拆分重塑，最终形成的特征向量的维数为:



实现一个猫分类器，需要准备大量的猫图及少量的非猫图，取其中大部分组成该分类器的训练样本，少部分组成测试样本。将这些样本都以上述的方式表示为特征向量的形式，则一个样本由一对进行表示，其中为维的特征向量，是该特征向量的标签（Label），根据该特征向量表示的是猫或非猫，值为0或1。个训练样本对将被表示为：



将训练集的中的特征向量以及它们的标签分别堆叠起来，组成两个矩阵：



则会是个大小为的矩阵，是个大小为的矩阵。

### 2、线性拟合（逻辑回归）

逻辑回归是一种用于解决监督学习（Supervised Learning）问题的学习算法。进行逻辑回归的目的，是使训练数据的标签值与预测出来的值之间的误差最小化。

猫分类器要实现的，是给定以一个维特征向量表示，标签为的一张图片，估计这张图片为猫图的概率，即：



有大量猫图的数据时，希望能用一个函数，来表示出。所以考虑使用线性拟合的方法，从大量数据中寻找规律，从而实现这个猫分类器。规定一个维向量和一个值作为参数，可得到线性回归的表达式：



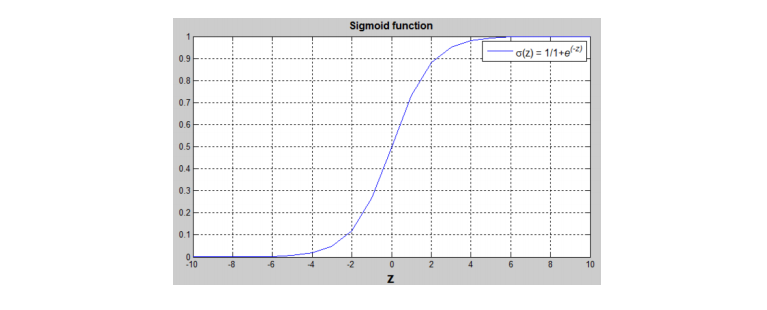
由于为概率值，取值范围为，简单地进行线性拟合，得出的可能非常大，也可能为负值。

### 3、Sigmod函数

由于可能非常大，也可能为负值，这时便需要一个逻辑回归单元来对其值域进行约束。以sigmoid函数为逻辑回归单元，而sigmoid函数的表达式为：



其函数图像为：



由函数图像可知，sigmoid函数有几个很好的性质：

- 当趋近于正无穷大时，

- 当趋近于负无穷大时，

- 当时，

所以可以用Sigmoid函数来约束的值域，此时：



### 4、损失函数（交叉熵）

用损失函数（Loss Function）来衡量单个样本预测值与真实值之间的差异，换句话说，损失函数用来计算单个训练样本的预测值与真实值之间的误差。平方误差（Square Loss）是一种常用的损失函数，但在逻辑回归中一般不使用这个损失函数，因为在训练参数过程中，使用这个损失函数将得到一个非凸函数，最终将存在很多局部最优解，这种情况下使用梯度下降（Gradient Descent）将无法获得最优解。通过推导，得到应用很广的交叉熵（Cross Entropy）损失函数，表达式为：



函数有如下性质：

当时，

当时，

### 5、成本函数

为了训练逻辑回归模型中的参数w和b，使得输出值与真实值y尽可能一致，即尽可能准确地判断一张图是否为猫，需要定义一个成本函数（Cost Function）作为衡量的标准。

定义一个函数来表示整个训练数据集差异的平均值，即成本函数（Cost Function）：

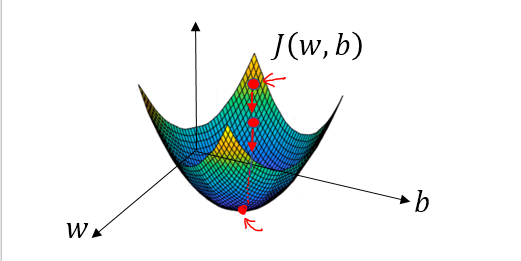


故，函数的最终形式是：



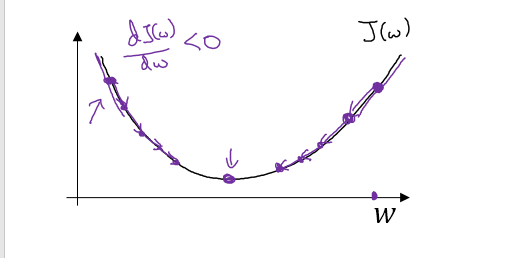
### 6、梯度下降

想找到使成本函数的值最小的参数w和b的值，一般采用梯度下降法。标量场中某一点上的梯度指向标量场增长最快的方向，梯度的长度是这个最大的变化率。



在空间坐标中以w，b为轴画出损失函数J(w,b)的三维图像，可知这个函数为一个凸函数。为了找到合适的参数，先将w和b赋一个初始值，正如图中的小红点。

在逻辑回归中，几乎任何初始化方法都有效，通常将参数初始化为零。随机初始化也起作用，但通常不会在逻辑回归中这样做，因为这个成本函数是凸的，无论初始化的值是多少，总会到达同一个点或大致相同的点。梯度下降就是从起始点开始，试图在最陡峭的下降方向下坡，以便尽可能快地下坡到达最低点，这个下坡的方向便是此点的梯度值。



在二维图像中来看，顺着导数的方向，下降速度最快，用数学公式表达即是：



其中的“:=”意思为赋值，为学习率，通常为一个小于1的数，用来控制梯度下降过程中每一次移动的规格，相当于迈的步子大小。的不宜太小也不宜过大：太小会使迭代次数增加，容易陷入局部最优解；太大容易错过最优解。

**求导数：**

成本函数：



对求导：



其中：

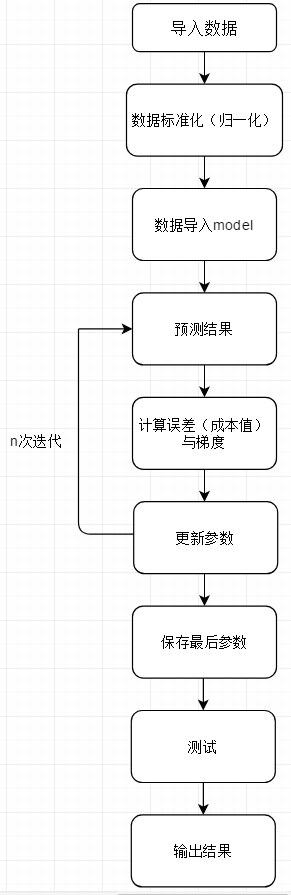


故：



注意：将公式转换为代码时，需要注意数据的格式问题，数据格式不同可能会引起公式表示方式不同。

## 二、实验代码流程图



## 三、代码讲解

### 1、导入数据

|  |
| --- |
| *#本项目训练及测试图片集是以 HDF5 二进制数据格式储存的* **def** load\_dataset():  train\_dataset = h5py.File(**"train\_cat.h5"**,**"r"**) *#读取训练数据* test\_dataset = h5py.File(**"test\_cat.h5"**, **"r"**) *#读取测试数据* train\_set\_x\_orig = np.array(train\_dataset[**"train\_set\_x"**][:]) *#原始训练集，train\_set\_x：图像矩阵* train\_set\_y\_orig = np.array(train\_dataset[**"train\_set\_y"**][:]) *#原始训练集的标签集（y=0非猫,y=1是猫），train\_set\_y：标签列表* test\_set\_x\_orig = np.array(test\_dataset[**"test\_set\_x"**][:]) *#原始训练集，train\_set\_x：图像矩阵* test\_set\_y\_orig = np.array(test\_dataset[**"test\_set\_y"**][:]) *#原始训练集的标签集（y=0非猫,y=1是猫），train\_set\_y：标签列表* train\_set\_y\_orig = train\_set\_y\_orig.reshape((1,train\_set\_y\_orig.shape[0])) *#原始训练集的标签集设为（1\*m）* test\_set\_y\_orig = test\_set\_y\_orig.reshape((1,test\_set\_y\_orig.shape[0])) *#原始测试集的标签集设为（1\*m）* classes = np.array(test\_dataset[**"list\_classes"**][:]) *#list\_classes：[b'non-cat' b'cat']* **return** train\_set\_x\_orig, train\_set\_y\_orig, test\_set\_x\_orig, test\_set\_y\_orig, classes |

### 2、Sigmoid函数

|  |
| --- |
| *#sigmoid函数:用sigmoid 函数来约束线性拟合（逻辑回归）中 y^ 的值域[0,1]* **def** sigmoid(z):  s = 1/(1+np.exp(-z))  **return** s |

### 3、初始化参数

|  |
| --- |
| *#初始化参数w,b ：定一个nx 维向量 w 和一个值 b 作为参数，以得到线性回归的表达式* **def** initialize\_with\_zeros(dim):  w = np.zeros((dim,1)) *#w为一个dim\*1矩阵* b = 0  **return** w, b |

### 4、计算y的值域和成本值

|  |
| --- |
| *#计算 Y\_hat,成本函数 J 以及 dw，db* **def** propagate(w, b, X, Y):  m = X.shape[1] *#样本个数* Y\_hat = sigmoid(np.dot(w.T,X)+b)  cost = -(np.sum(np.dot(Y,np.log(Y\_hat).T)+np.dot((1-Y),np.log(1-Y\_hat).T)))/m *#成本函数* dw = (np.dot(X,(Y\_hat-Y).T))/m  db = (np.sum(Y\_hat-Y))/m   cost = np.squeeze(cost) *#压缩维度* grads = {**"dw"**: dw,  **"db"**: db} *#梯度* **return** grads, cost |

### 5、梯度下降法找最优值

|  |
| --- |
| *#梯度下降找出最优解 #num\_iterations-梯度下降次数 learning\_rate-学习率，即参数ɑ* **def** optimize(w, b, X, Y, num\_iterations, learning\_rate, print\_cost = **False**):*#num\_iterations-梯度下降次数 learning\_rate-学习率，即参数ɑ* costs = [] *#记录成本值* **for** i **in** range(num\_iterations): *#循环进行梯度下降* 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: *#每100次记录一次成本值* costs.append(cost)   **if** print\_cost **and** i % 100 == 0: *#打印成本值* print (**"循环%i次后的成本值: %f"** %(i, cost))   params = {**"w"**: w,  **"b"**: b} *#最终参数值* grads = {**"dw"**: dw,  **"db"**: db}*#最终梯度值* **return** params, grads, costs |

### 6、预测结果

|  |
| --- |
| *#预测出结果* **def** predict(w, b, X):  m = X.shape[1] *#样本个数* Y\_prediction = np.zeros((1,m)) *#初始化预测输出* w = w.reshape(X.shape[0], 1) *#转置参数向量w-->T* Y\_hat = sigmoid(np.dot(w.T,X)+b) *#最终得到的参数代入方程* **for** i **in** range(Y\_hat.shape[1]):  **if** Y\_hat[:,i]>0.5:  Y\_prediction[:,i] = 1  **else**:  Y\_prediction[:,i] = 0   **return** Y\_prediction |

### 7、建立预测模型

|  |
| --- |
| *#建立整个预测模型* **def** model(X\_train, Y\_train, X\_test, Y\_test, num\_iterations = 2000, learning\_rate = 0.5, print\_cost = **False**): *#num\_iterations-梯度下降次数 learning\_rate-学习率，即参数ɑ* w, b = initialize\_with\_zeros(X\_train.shape[0]) *#初始化参数w，b* parameters, grads, costs = optimize(w, b, X\_train, Y\_train, num\_iterations, learning\_rate, print\_cost) *#梯度下降找到最优参数* w = parameters[**"w"**]  b = parameters[**"b"**]   Y\_prediction\_train = predict(w, b, X\_train) *#训练集的预测结果* Y\_prediction\_test = predict(w, b, X\_test) *#测试集的预测结果* train\_accuracy = 100 - np.mean(np.abs(Y\_prediction\_train - Y\_train)) \* 100 *#训练集识别准确度* test\_accuracy = 100 - np.mean(np.abs(Y\_prediction\_test - Y\_test)) \* 100 *#测试集识别准确度* print(**"训练集识别准确度: {} %"**.format(train\_accuracy))  print(**"测试集识别准确度: {} %"**.format(test\_accuracy))   d = {**"costs"**: costs,  **"Y\_prediction\_test"**: Y\_prediction\_test,  **"Y\_prediction\_train"** : Y\_prediction\_train,  **"w"** : w,  **"b"** : b,  **"learning\_rate"** : learning\_rate,  **"num\_iterations"**: num\_iterations}   **return** d |

### 8、初始化数据，得到结果

|  |
| --- |
| *#初始化数据* train\_set\_x\_orig, train\_set\_y, test\_set\_x\_orig, test\_set\_y, classes = load\_dataset()  m\_train = train\_set\_x\_orig.shape[0] *#训练集中样本个数* m\_test = test\_set\_x\_orig.shape[0] *#测试集总样本个数* train\_set\_x\_flatten = train\_set\_x\_orig.reshape(train\_set\_x\_orig.shape[0],-1).T *#原始训练集的设为（12288\*209）* test\_set\_x\_flatten = test\_set\_x\_orig.reshape(test\_set\_x\_orig.shape[0],-1).T *#原始测试集设为（12288\*50）* train\_set\_x = train\_set\_x\_flatten/255. *#将训练集矩阵标准化* test\_set\_x = test\_set\_x\_flatten/255. *#将测试集矩阵标准化* d = model(train\_set\_x, train\_set\_y, test\_set\_x, test\_set\_y, num\_iterations = 2000, learning\_rate = 0.005, print\_cost = **True**) |

### 9、画出学习曲线

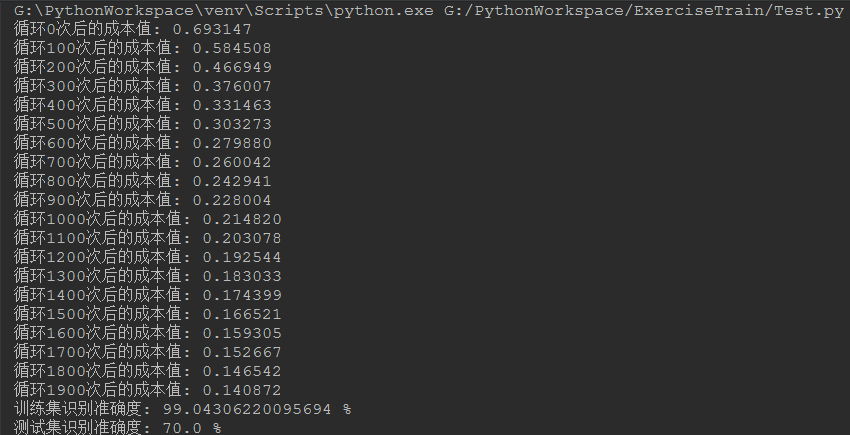
|  |
| --- |
| *# 画出学习曲线* costs = np.squeeze(d[**'costs'**]) plt.plot(costs) plt.ylabel(**'cost'**) plt.xlabel(**'iterations (per hundreds)'**) plt.title(**"Learning rate ="** + str(d[**"learning\_rate"**])) plt.show() |

### 10、不同学习率的学习曲线

|  |
| --- |
| *#学习率不同时的学习曲线* learning\_rates = [0.01, 0.001, 0.0001] models = {} **for** i **in** learning\_rates:  print (**"学习率: "** + 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() |

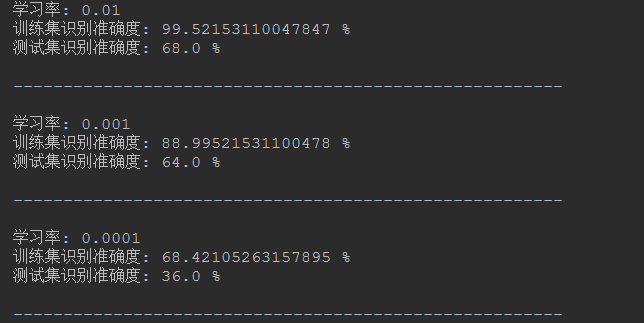
## 四、实验结果与分析

### 1、训练集和测试集的准确度

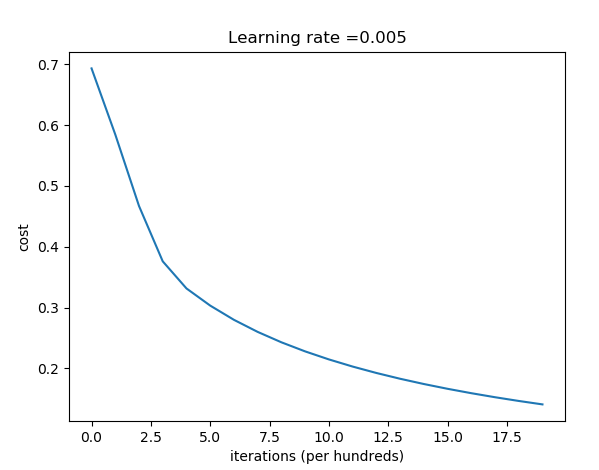


训练集识别准确率接近100％，测试集的准确率有70％。由于训练使用的小数据集，而且逻辑回归是线性分类器，所以这个结果对于这个简单的模型实际上还是不错。

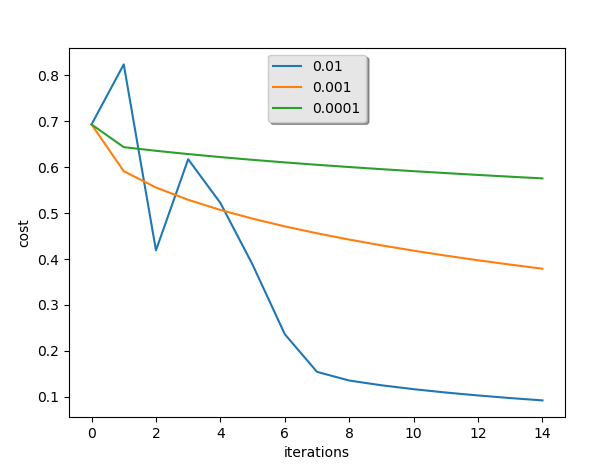
### 2、不同学习率时的准确度



### 3、学习曲线



### 4、不同学习率的学习曲线



**扫码关注微信公众号：孔乙己学习成长录**

**了解更多**

****