基于 Logistic Regression 的分类器设计

经过一单元的模式识别课程的学习，我对最小二乘、梯度下降、PCA、SVD 分解、BP 神经网络、Logistic Regression（LR）等有了一定地了解，其中对 LR 算法比较热衷，课下花费了大量时间进行查阅资料，所以此次大作业我根据 LR 算法进行了多个分类器设计，包括二分类和多分类，其中考试录取预测问题的识别率达到了 89%，让我对模式识别这门课有了更深刻的认识。

下面将结合项目对基于 Logistic Regression 的分类器设计进行详细叙述。

一、理论概述

Logistic Regression, 中文有人译作“逻辑斯蒂回归”, 也有译作“对数几率回归”。后者具有更深厚理论理解，稍后会对这个名字的来源加以介绍。这里需要首先指明的是 Logistic 回归模型是一个分类模型而不是一个字面上的回归模型!

首先回顾一下线性回归模型:

$$f(x) = w_0x_0 + \ w_1x_1 + \ \cdots + w_nx_n + b \tag{1}$$

写成向量形式:

$$f(x) = \boldsymbol{w^T x + b} \tag{2}$$

写成广义线性回归模型:

$$y = g^{-1}(\boldsymbol{w^T x + b}) \tag{3}$$

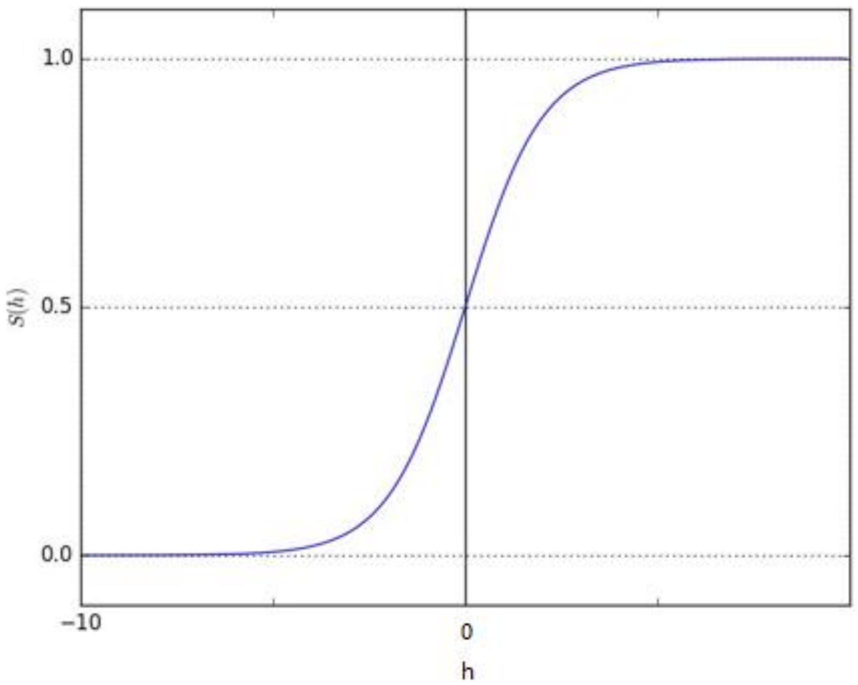
下面介绍从线性回归模型引出 logistic 回归的分类模型。线性回归模型只能够进行回归学习，但是若要是做分类任务则需要在“广义线性回归”模型中：只需找一个单调可微函数将分类任务的真实标记 y 与线性回归模型的预测值联系起来便

可以了！

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 $\boldsymbol{z = wx + b}$ 是一个实值，所以需要将实值 z 转化成 0/1 值。这就可以选择映射函数来实现。“sigmoid”函数就是一个很好的备选项：

$$y = \frac{1}{1 + e^{-z}} \tag{4}$$

该函数的图像如下所示：



由图像可知，当预测值大于 0 便可判断为正例，小于 0 则判断为反例。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，可以用于二分类问题：

$$y = \frac{1}{1 + e^{-(w^T x + b)}} \tag{5}$$

对上式的预测函数做一个变换，即得：

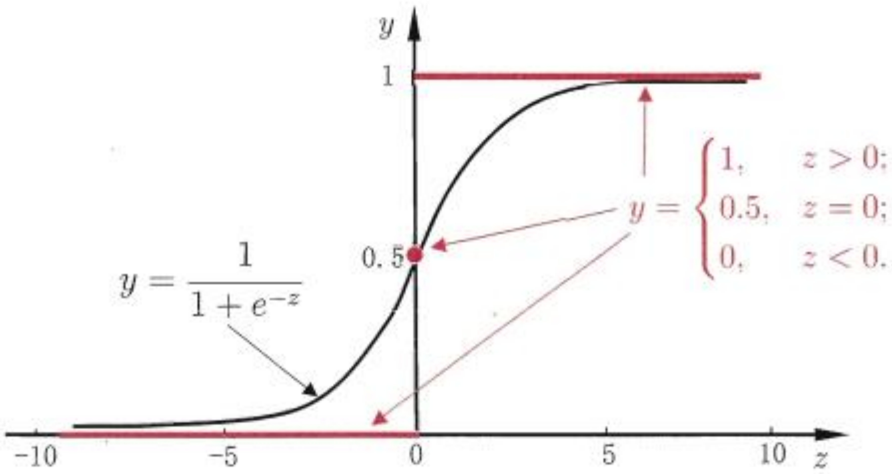
$$ln\frac{y}{1-y} = w^T x + b \tag{6}$$

若将 y 视作样本 x 作为正例的可能性，则 1-y 是其反例可能性，二者比值就 i 叫做“几率”（odds），反映了 x 作为正例的相对可能性。对几率取对数即得式（6）左侧部分，叫做“对数几率”（log odds，亦叫做 logit）。这就是“对数几率回归”名称的来源。

其实，最理想的是“单位阶跃函数”（***unit-step function***）：

$$y = \begin{cases} 0, & z < 0 \\ 0.5, & z = 0 \\ 1, & z > 0 \end{cases} \tag{7}$$

即若预测值 z 大于 0 就判定为正例，小于零则判为反例，临界值 0 则可判定为任意，如图示意：



二、构建目标函数

下面开始介绍如何根据评估策略求解 **模型：式（5）** 中的未知参数。

2.1 提出假设函数

若将式（5）中的 y 看作类后验概率估计 $p(y=1 \mid x)$，则式（6）可重写为：

$$ln\frac{p(y=1 \mid x)}{p(y=0 \mid x)} = \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x} + \boldsymbol{b} \tag{8}$$

显然有：

$$p(y=1|x) = \frac{e^{w^T x+b}}{1+e^{w^T x+b}} \tag{9}$$

$$p(y=0|x) = \frac{1}{1+e^{w^T x+b}} \tag{10}$$

为了便于讨论，令 $\boldsymbol{\theta} = (\boldsymbol{w}, \boldsymbol{b})$，则$\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x} + \boldsymbol{b} = \boldsymbol{\theta}^{\boldsymbol{T}}\boldsymbol{x}$，其中 x=(x; 1)

构造假设函数如下，其输出即是概率∈ [0, 1]：

$$\hbar_\theta(x) = g(\theta^T x) = \frac{1}{1+e^{\theta^T x}} \tag{11}$$

则有：

$$P(y=1|x;\theta) = h_\theta(x) \tag{12}$$

$$P(y=0|x;\theta) = 1 - h_\theta(x) \tag{13}$$

2.2 极大似然估计

通过最大似然估计，给定数据集{x, y}，式（12）（13）可合写为：

$$p(y|x;\theta) = [h_\theta(x)]^y \times [1-h_\theta(x)]^{1-y} \tag{14}$$

假设各样本独立同分布，取似然函数：

$$L(\theta) = \prod p(y|x;\theta) = \prod [h_\theta(x)]^y \times [1-h_\theta(x)]^{1-y} \tag{15}$$

对数似然函数为：

$$l(\theta) = logL(\theta) = \sum \left(ylogh_\theta(x) + (1-y)\log\left(1-h_\theta(x)\right)\right) \tag{16}$$

2.3 损失函数与代价函数

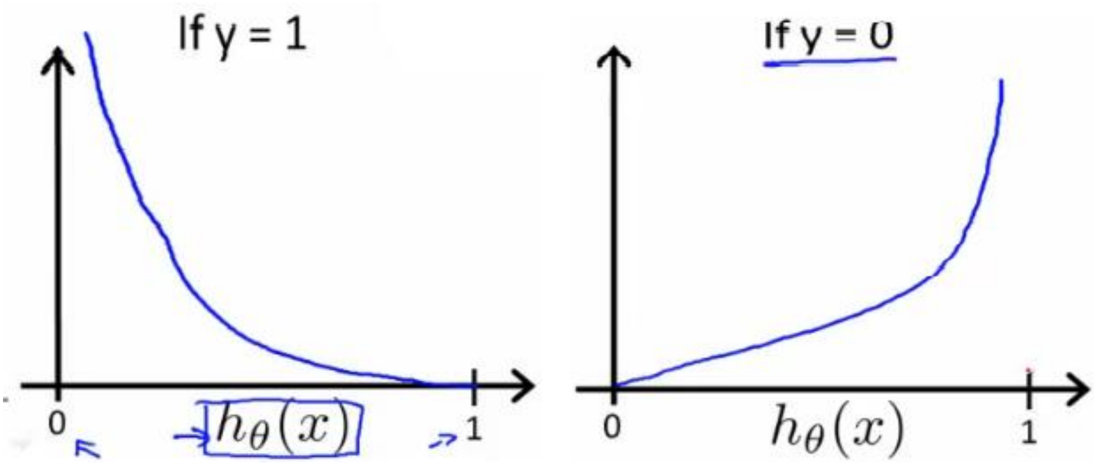
在某些资料中(16)式又叫作对数似然损失函数，即：

$$loss(h_\theta(x), y) = \ -y \ \ logh_\theta(x) - (1-y)\log\big(1-h_\theta(x)\big) \qquad (17)$$

分段表示为：

$$loss(h_\theta(x), y) = \begin{cases} -logh_\theta(x), & y = 1 \\ -\log\big(1-h_\theta(x)\big), & y = 0 \end{cases} \qquad (18)$$

用图示可以清晰理解损失函数的概率意义：



从左图中可以看出：对于 y=1 的样本，当预测的 h 值（概率）趋近于 1 的时候，损失函数是趋于 0 的；当预测的 h 值（概率）趋近于 0 的时候，损失函数趋于无穷大，即预测值 h 和训练集结果 y=0 差别越大，预测错误的代价则越趋于无穷大。右图可以同理分析，这里不再赘述。

查阅资料知道，**损失函数（Loss Function）**是定义在单个样本上的，算的是一个样本的误差；**代价函数（Cost Function）**是定义在整个训练集上的，是所有

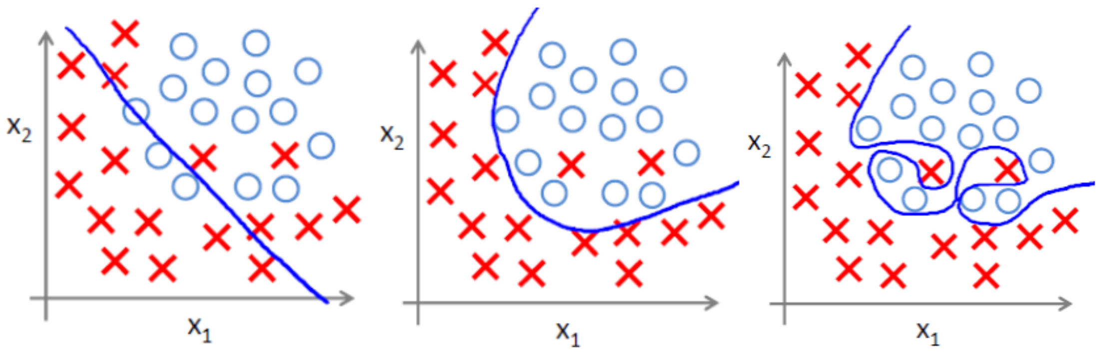
样本误差的平均，也就是损失函数的平均。因而可根据（17）式进一步得到代价函数的表达式如下：

$$
\begin{aligned}
& cost(h_\theta(x), y) \\
&= \frac{1}{m}\sum_{i=1}^{m} loss^{(i)} \\
&= -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log h_\theta\left(x^{(i)}\right) + \left(1 - y^{(i)}\right)\log\left(1 - h_\theta\left(x^{(i)}\right)\right)\right] \qquad (19)
\end{aligned}
$$

根据代价最小化标准即可认定（19）式作为目标函数，但为了模型的完善性和合理性，需进一步处理。

2.4 正则化

对于线性回归或逻辑回归的损失函数构成的模型，可能会有些权重很大，有些权重很小，导致过拟合（就是过分拟合了训练数据），使得模型的复杂度提高，泛化能力较差（对未知数据的预测能力）。下面左图即为欠拟合，中图为合适的拟合，右图为过拟合：



过拟合问题往往源自过多的特征。减少特征数量是一种可选方法，但往往在特征较多时使用正则化来避免过拟合。

正则化是结构风险最小化策略的实现，是在经验风险上加一个正则化项或惩罚项。正则化项一般是模型复杂度的单调递增函数，模型越复杂，正则化项就越大。在 Logistic 回归问题中一般取平方损失，即参数的 L2 范数（也可以取 L1 范数），参数的 L2 范数表达式如下：

$$\lambda\|\theta\|_2^2 = \lambda\sum_{j=1}^{n}\theta_j^2 \tag{20}$$

其中 λ 称为正则项系数，表示了对模型复杂度的惩罚度：其值越大，则对拟合数据的损失惩罚越小，避免过分拟合，但偏差较大，在未知数据上的方差较小，可能出现欠拟合；其值越小，则说明比较注重对训练数据的拟合，使得偏差较小，但是可能导致过拟合。

2.5 目标函数

结合上述模型与策略，即可提出具备正则项的目标函数：

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log h_\theta\left(x^{(i)}\right) + \left(1-y^{(i)}\right)\log\left(1-h_\theta\left(x^{(i)}\right)\right)\right] + \frac{\lambda}{m}\sum_{j=1}^{n}\theta_j^2 \tag{21}$$

三、根据目标函数求解模型参数

3.1 Sigmoid 函数的良好性质

Sigmoid 函数具有简洁的导数形式，便于接下来通过梯度下降法求解未知参数。下面给出求导过程：

$$
\begin{aligned}
g(z) &= \frac{d}{dz}\frac{1}{1+e^{-z}} \\
&= \frac{1}{(1+e^{-z})^2} \quad e^{-z} \\
&= \frac{1}{(1+e^{-z})} \cdot \quad \frac{e^{-z}}{(1+e^{-z})} \\
&= \frac{1}{(1+e^{-z})} \cdot \left(1 - \quad \frac{1}{(1+e^{-z})}\right) \\
&= g(z) \cdot [1 - g(z)]
\end{aligned} \tag{22}
$$

3.2 梯度求解

对（21）式的目标函数求梯度，可得梯度：

$$\frac{\partial}{\partial \theta_j} \; J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)} + \frac{\lambda}{m}\theta_j \tag{23}$$

根据梯度下降法可得 θ 的更新过程：

$$\theta_{\mathrm{j}} := \theta_j - \alpha\frac{\partial}{\partial \theta_j}J(\theta), \qquad (j = 0, \cdots, n) \tag{24}$$

式中为α学习步长，又叫做学习率。

3.3 向量化

向量化目标函数及其梯度表达式，得：

$$J(\theta) = \; -\frac{1}{m}\left(y^T\log(h)\right) + \left(1 - y^T\right)\log(1-h) + \frac{\lambda}{m}\theta^T\theta \tag{25}$$

$$\frac{\partial}{\partial \theta} \; J(\theta) = \frac{1}{m}X^T(h-y) + \frac{\lambda}{m}\theta \tag{26}$$

其中，$\boldsymbol{X}$是m × n的特征矩阵，列向量 $\boldsymbol{h}$ 是 sigmoid 函数的取值：

$$\boldsymbol{h} = g(\boldsymbol{X\theta}) \tag{27}$$

3.4 梯度下降法

根据上述的向量化公式，可按下述步骤迭代更新θ值：

（1）求 $\boldsymbol{h} = g(\boldsymbol{X\theta})$；

（2）求 $\boldsymbol{error} = \boldsymbol{h} - \boldsymbol{y}$；

（3）求 $\boldsymbol{\theta} \coloneqq \boldsymbol{\theta} - \alpha \cdot X^T \cdot \boldsymbol{error} - \lambda\boldsymbol{\theta}$

四、代码实现

这里给出**梯度下降法**的代码实现：

```
# 梯度下降法
def grad_desc(data_mat, label_mat, rate, times, lamb=1):
    """
    :param data_mat: 数据特征
    :param label_mat: 数据标签
    :param rate: 速率
    :param times: 循环次数
    :lamb: 正则项系数，默认为 1
    :return: 参数向量
    """
    data_mat = np.mat(data_mat)
    label_mat = np.mat(label_mat)
    m, n = np.shape(data_mat)
    weight = np.ones((n, 1))
    for i in range(times):
        h = sigmoid(data_mat * weight)
        error = h - label_mat
        weight = weight - rate * data_mat.transpose() * error - lamb * weight
    return weight
```

从上面代码分析可以看出，虽然只有十多行的代码，但是里面却隐含了太多的细节，如果没有相关基础确实是非常难以理解的。

这里将**随机梯度下降法**的代码实现也一并给出：

```
# 随机梯度下降法
def random_grad_desc(data_mat, label_mat, rate, times, lamb=1):
    data_mat = np.mat(data_mat)
    m, n = np.shape(data_mat)
    weight = np.ones((n, 1))
    for i in range(times):
        for j in range(m):
            h = sigmoid(data_mat[j] * weight)
            y = 1 if label_mat[j] == 1 else 0
            error = h - y
            weight = weight - rate * data_mat[j].transpose() * error- lamb * weight
    return weight
```

五、多类分类问题的 Logistic Regression

上面介绍的 Logistic 回归模型是二项分类模型，用于二类分类。可以将其推广到多项 Logistic 回归模型（multi-nominal logistic regression model），用于多类分类。假设离散型随机变量 Y 的取值集合是{1，2，…，K}，那么多项回归模型是：

$$P(Y=k|x)=\frac{\exp(w_k\cdot x)}{1+\sum_{k=1}^{K-1}\exp(w_k\cdot x)}, \quad k=1,2,\cdots,K-1 \tag{23}$$

$$P(Y=K|x)=\frac{1}{1+\sum_{k=1}^{K-1}\exp(w_k\cdot x)} \tag{24}$$

根据上式，我们可以简化判断 *exp1*，*exp2*，… *exp n-1*，具体如代码所示

```
def classifier(X, weights):
    m, n = X.shape
    X = np.hstack([np.ones((m, 1)), X])        # 为 X 增加偏置列

    exp1 = np.exp(np.matmul(X, weights[0]))
    exp2 = np.exp(np.matmul(X, weights[1]))

    categories = []
    for e1, e2 in zip(exp1, exp2):
        if not (e1 + e2):
            category = 3
        elif not e1:
            category = 2
        elif not e2:
            category = 1
        categories.append(category)

    return categories
```

六、项目相关介绍

6.1 问题介绍

我们将建立一个逻辑回归模型来预测一个学生是否被大学录取。假设你是一个大学系的管理员，你想根据两次考试的结果来决定每个申请人的录取机会。你有以前的申请人的历史数据，你可以用它作为逻辑回归的训练集。对于每一个培训例子，你有两个考试的申请人的分数和录取决定。为了做到这一点，我们将建立一个分类模型，根据考试成绩估计入学概率。（**多分类问题见“红酒品质分类”问题，关键在于数据划分和预测判断，上文已经提及，这里不再赘述）**

6.2 部分代码示例

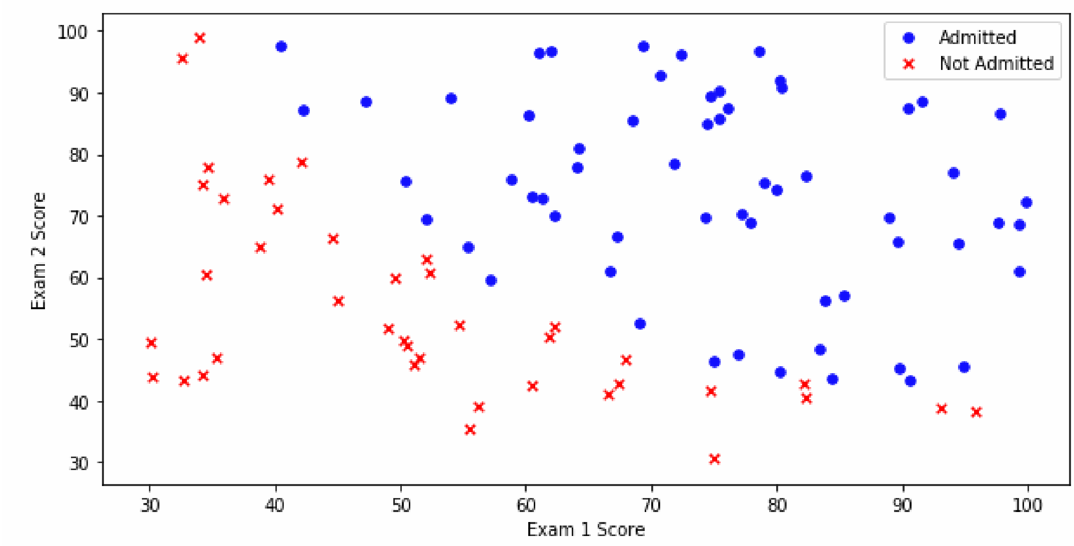
1. 导入数据集

```
path = 'LogiReg_data.txt'

pdData = pd.read_csv(path, header=None, names=['Exam 1', 'Exam 2', 'Admitted'])
```

2. 数据示图

```
positive = pdData[pdData['Admitted'] == 1]

negative = pdData[pdData['Admitted'] == 0]



fig, ax = plt.subplots(figsize=(10,5))

ax.scatter(positive['Exam 1'], positive['Exam 2'], s=30, c='b', marker='o', label='Admitted')

ax.scatter(negative['Exam 1'], negative['Exam 2'], s=30, c='r', marker='x', label='Not Admitted')

ax.legend()
```

```
ax.set_xlabel('Exam 1 Score')

ax.set_ylabel('Exam 2 Score')
```



3. 数据处理

```
pdData.insert(0, 'Ones', 1)  # 插入一列，保障系数 beta0
```

4. 预测

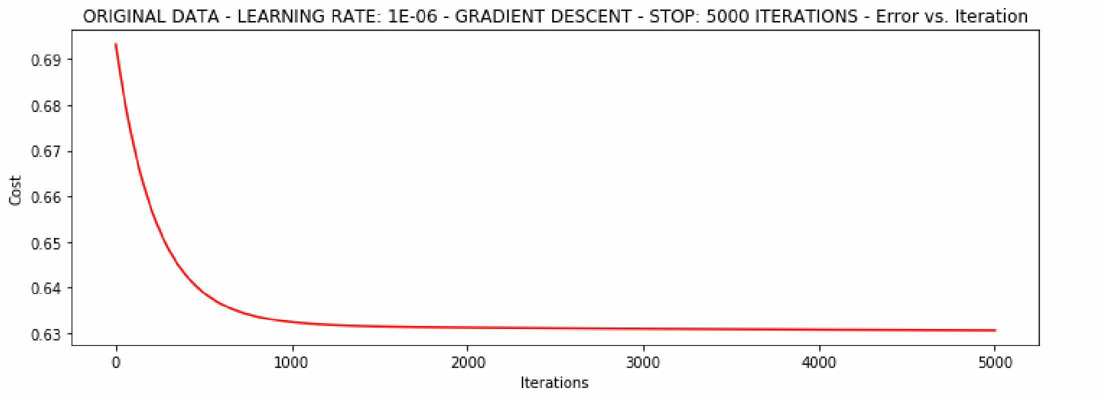
```
#设定阈值

def predict(X, theta):

    return [1 if x >= 0.5 else 0 for x in model(X, theta)]
```

5. 精确度计算

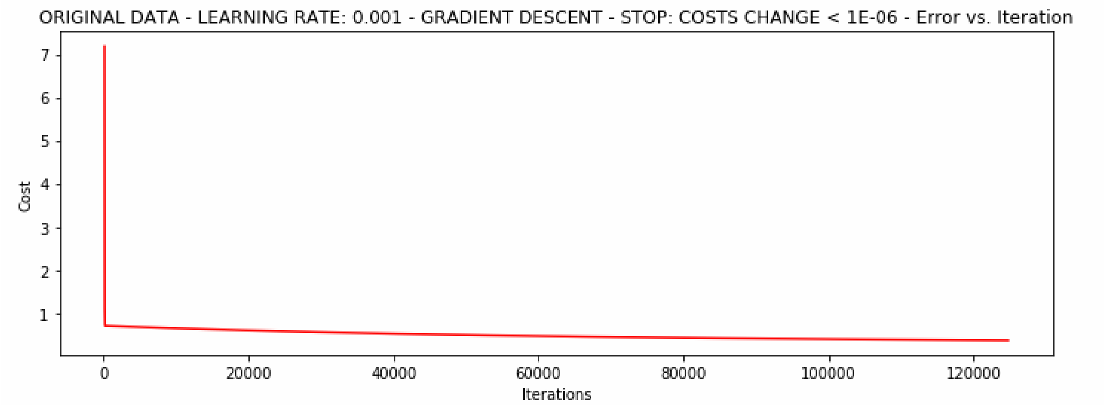
```
scaled_X = scaled_data[:, :3]

y = scaled_data[:, 3]

predictions = predict(scaled_X, theta)

correct = [1 if ((a == 1 and b == 1) or (a == 0 and b == 0)) else 0 for (a, b) in
zip(predictions, y)]

accuracy = (sum(map(int, correct)) % len(correct))

print('accuracy = {0}%'.format(accuracy))
```

6.3 关于梯度下降法

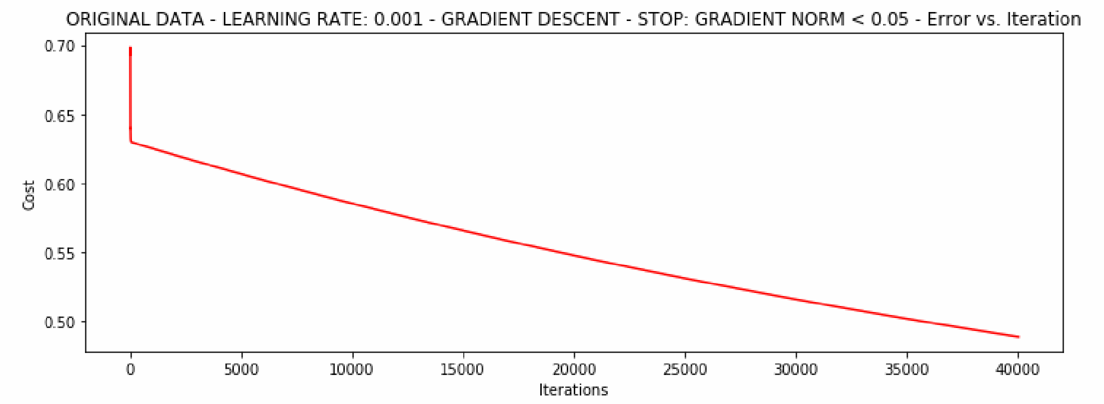
我们可以选择多种条件控制迭代，比如设定迭代次数，超出则停止：



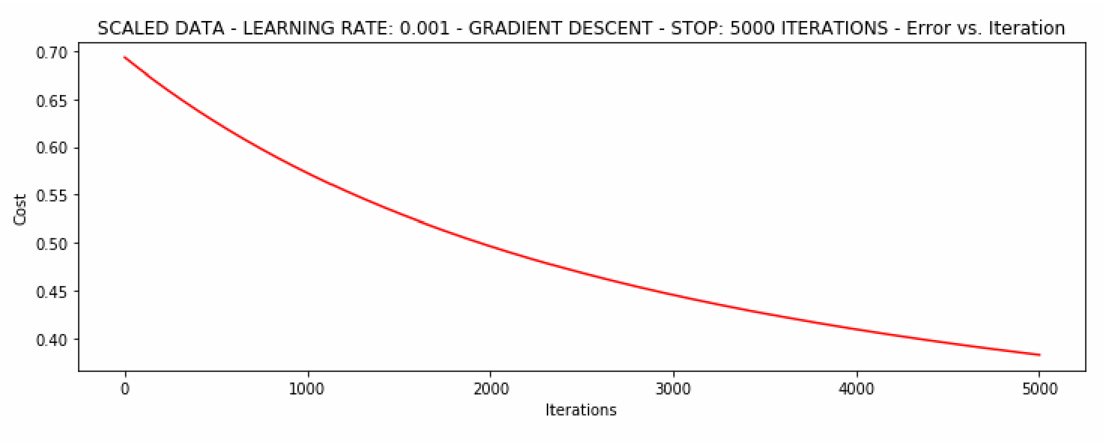
也可以根据损失值停止：



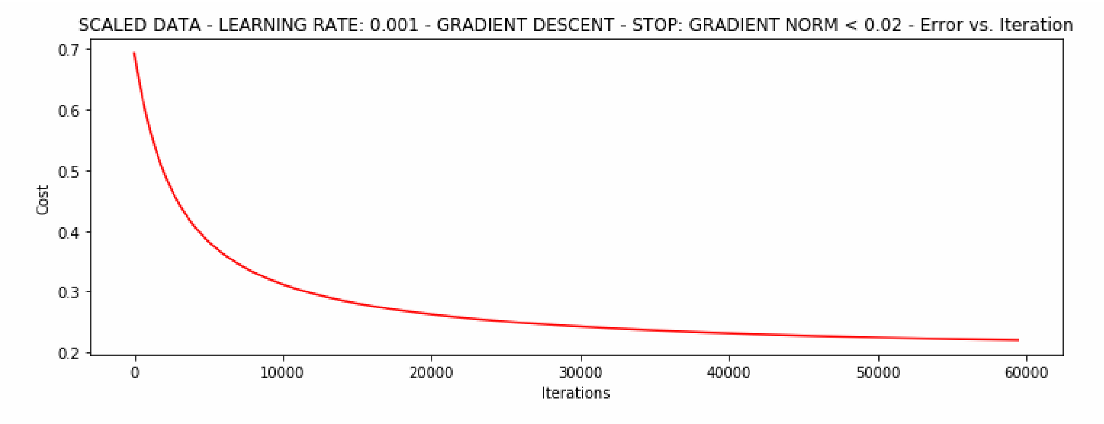
还可以根据梯度变化停止：



同时也可以选用不同的梯度下降法，比如 Mini-batch descent



另外调整学习率也是很关键的:



最后，综合得出识别精确率 85%。

七、总结

回顾前些日子，感触颇多。熬了好几个晚上，起初准备做支持向量机，但真的好难，啃了几天没啃下来，在用了 OpenCV 的库函数进行了手写数字识别之后，就放弃了。转做 Logistic Regression，做这个也很不容易。刚开始是选的 Lentcode 上商品分类的那道题，但是那个题是 9 类分类，而我对 LG 算法的了解还处于二值分类问题上。查了些资料，感觉有点眉目了就开始做，但类别太多实现起来还

是力不从心。就查阅各类测试网站，看到了经典的葡萄酒分类问题，只有三类，查阅了相关资料后，我进行了实现，在数据集偏斜不太明显的情况下，得到了 80% 的正确率。除此之外，我还选做了预测考试录取的问题，实现起来到达了 85%的正确率。每一行代码都是在自己深刻理解了理论推导之后，敲出来。有 jupyter notebook 的帮助真的是事半功倍，感谢赵老师的倾情分享！

总的来说，这次的大作业完成得很艰难，并且印象深刻，收获很多。我深知“欲速则不达”的道理，这次却还是陷了进去。前边的几天真的是浪费在发愁和焦虑上了，根本没有静下心来去思考理论，一直在找例程，也一直被网上的良莠不齐的信息所干扰，是真真的事倍功半，一无所获。直到最后，花了大半天时间推导代价函数和正则项构成的目标函数之后，又复习了梯度下降法，才算把理论搞清楚，心里踏踏实实的。虽然 numpy 依然有很多坑，又花费了我大半天时间去学习，但效率的确非常高。磨刀不误砍柴工，道理都懂，做到贼难。

还有就是在我查阅资料的同时，发现了很多各式算法，有 K-means 的、决策树的、CNN 的，都是我不曾接触的，看到这类算法的识别率，再看看我自己算法的效果极大地增强了我对模式识别、机器学习的兴趣，我一定会坚持不懈，继续走下去的。最后，再次感谢赵老师课上课下、线上线下的指导和分享！