模式识别与统计学习实验二：Logistic Regression实验报告

郭宇航 2016100104014

一 实验原理

1.1 Logistic Regression思想引入

第一次实验我们实现了最为基础的线性回归，这是一种在考虑连续型变量的预测问题（例如房价的预测）时的方法，而当我们在需要考虑离散变量时，问题就变为了一个分类而非回归。因此线性回归就不再适用。但是线性回归作为一种回归预测的算法，给我们提供了一些灵感，如果能够将连续型数据模型离散化，就可以进行分类问题的处理了。首先我们考虑一个二分类的问题，其输出变量为$y \in 0, 1$，线性回归产生的预测值为实值：$y = \boldsymbol{\omega^T x} + b$，我们将实值转化为0/1值，其中最理想的就是单位阶跃函数：

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

当预测值y大于零就判别为正例，小于零则判为反例，预测值为临界值则可以进行任意的判别。然而考虑阶跃函数，我们发现其并不连续，后续的优化无法进行，所以我们需要寻找一个有同样的功能且单调可微的函数。Logistic函数[1]就是这样一种函数。我们对线性回归模型进行变形得到这样的结果：

$$y = h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}} \tag{2}$$

其中$g(z) = 1/(1 + e^{-z})$，我们将这个函数成为Logistic函数或者sigmoid函数。这个函数的有这样的特性，当$z \to \infty$时，$g(z) \to 1$，当$z \to -\infty$时，$g(z) \to 0$，因此$h_\theta(x)$的上下限为0和1.为什么会选取这样一个函数，还有关于指数分布族的问题，在Softmax Regression中我们会展开进一步的讨论。另外我们发现这样一个sigmoid函数存在这样一个性质：

$$\begin{aligned} g'(z) &= \frac{\mathrm{d}}{\mathrm{d}z} \frac{1}{1 + e^{-z}} = \frac{1}{(1 + e^{-z})^2}(e^{-z}) \\ &= \frac{1}{(1 + e^{-z})} \cdot (1 - \frac{1}{(1 + e^{-z})}) \\ &= g(z)(1 - g(z)) \end{aligned} \tag{3}$$

建立了上述的logistic regression模型，和线性回归模型类似的我们需要对其中的参数θ进行估计，对于线性回归模型，我们直接使用最小二乘法中梯度下降法寻找全局最优解。而对于logistic regression来说，我们使用极大似然估计对参数进行估计。（实际极大似然估计与最小二乘法在某些特定情况下是一致的。）

1.2 极大似然估计

极大似然估计(Maximum Likihood Estimation)是我们在概率论与数理统计中学过的一种参数

估计（点估计）的一种方法。其主要思想是：在我已经有了样本的前提下，我们寻找参数，使得在这个参数下的概率分布和样本值最为接近。首先引入条件概率$P(\boldsymbol{D}|\boldsymbol{\theta})$，其中$\boldsymbol{D}$表示样本数据集$\boldsymbol{D} = \{x_1, x_2, \cdots, x_n\}$，接着构建似然函数$L(\boldsymbol{\theta}) = P(\boldsymbol{D}|\boldsymbol{\theta}) = P(x_1, x_2, \cdots, x_n|\boldsymbol{\theta})$.由于我们有最基本的假设，样本之间相互独立。由此我们将联合概率拆分，并对似然函数取对数，得到如下的结果：

$$
\begin{aligned}
L(\boldsymbol{\theta}) &= P(x_1, x_2, \cdots, x_n|\boldsymbol{\theta}) = \prod_{i=1}^{n} P(x_i|\boldsymbol{\theta}) \\
l(\boldsymbol{\theta}) &= \log(L(\boldsymbol{\theta})) = \sum_{i=1}^{n} \log(P(x_i|\boldsymbol{\theta}))
\end{aligned}
\tag{4}
$$

下面引入最小二乘法与极大似然估计之间的联系，首先考虑最小二乘法，回忆线性回归：$\boldsymbol{Y} = \boldsymbol{\theta}^T\boldsymbol{X} + \epsilon$，损失函数为：$J(\boldsymbol{\theta}) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2$，我们假设其中的$\epsilon$误差项服从正态分布：$\epsilon\ N(0, \sigma^2)$，由此我们可以得到概率：

$$
\begin{aligned}
P(\epsilon_i) &= \frac{1}{\sqrt{2\pi}\sigma}\exp(-\frac{(\epsilon_i)^2}{2\sigma^2}) \\
P(y_i|x_i;\theta) &= \frac{1}{\sqrt{2\pi}\sigma}\exp(-\frac{(y_i - \theta^T x_i)^2}{2\sigma^2})
\end{aligned}
$$

带入上面的式4极大似然估计，我们不难得到：

$$
\begin{aligned}
l(\boldsymbol{\theta}) &= \log(L(\boldsymbol{\theta})) \\
&= \sum_{i=1}^{m} \log\frac{1}{\sqrt{2\pi}\sigma}\exp(-\frac{(y_i - \theta^T x_i)^2}{2\sigma^2}) \\
&= m\log\frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{\sigma^2}\times\frac{1}{2}\sum_{i=1}^{m}(y_i - \theta^T x_i)^2
\end{aligned}
\tag{5}
$$

注意极大似然估计求解似然函数的最大值，也就是求解式5中的$\frac{1}{2}\sum_{i=1}^{m}(y_i - \theta^T x_i)^2$的最小值，对应于最小二乘法中的损失函数，两者是一致的。

1.3 logistic regression推导及求解

之前引入了sigmoid函数，通过引入这一函数，我们可以得到：

$$
\begin{cases}
P(y = 1|x;\theta) = h_\theta(x) \\
P(y = 0|x;\theta) = 1 - h_\theta(x)
\end{cases}
\tag{6}
$$

可以一般化的写为$P(y|x;\theta) = (h_\theta(x))^y(1 - h_\theta(x))^{1-y}$，假设我们有$m$个样本，则极大似然估计的似然函数可以写为：

$$
l(\theta) = \sum_{i=1}^{m} y^{(i)}\log h(x^{(i)}) + (1 - y^{(i)})\log(1 - h(x^{(i)}))
\tag{7}
$$

同样的问题现在转化为求解似然函数的极大值，仿照线性回归，我们使用梯度法寻找极值点。表达式可以表示为：$\theta := \theta + \alpha\nabla_\theta l(\theta)$，其中求导过程可以表示为：

$$
\begin{aligned}
\frac{\partial}{\partial\theta_j}l(\theta) &= (y\frac{1}{g(\theta^T x)} - (1 - y)\frac{1}{1 - g(\theta^T x)})\frac{\partial}{\partial\theta_j}g(\theta^T x) \\
&= (y(1 - g(\theta^T x)) - (1 - y)g(\theta^T x))x_j \\
&= (y - h_{\theta(x)})x_j
\end{aligned}
\tag{8}
$$

因此最终我们使用梯度上升法得到的结果如下：

$$
\theta_j = \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)}
\tag{9}
$$

二 python代码实现

2.1 数据读取

我们使用MINST数据集作为实验数据，MINST数据集来自美国国家标准与技术研究所，由来自250个不同的人手写的数字构成，其中50%是高中学生，50% 来自人口普查局的工作人员[2]。其中一共包含了四个数据集：

- **Training set images**:train-images-idx3-ubyte.gz（包含60000个训练样本）
- **Training set labels**:train-labels-idx1-ubyte.gz（包含60000个训练标签）
- **Test set images**:t10k-images-idx3-ubyte.gz（包含10000个测试样本）
- **Test set labels**:t10k-labels-idx1-ubyte.gz（包含10000个测试标签）

给出一组示例： 如图1所示，每一张手写数字均是28×28像素的黑白图片。我们使用python中读取

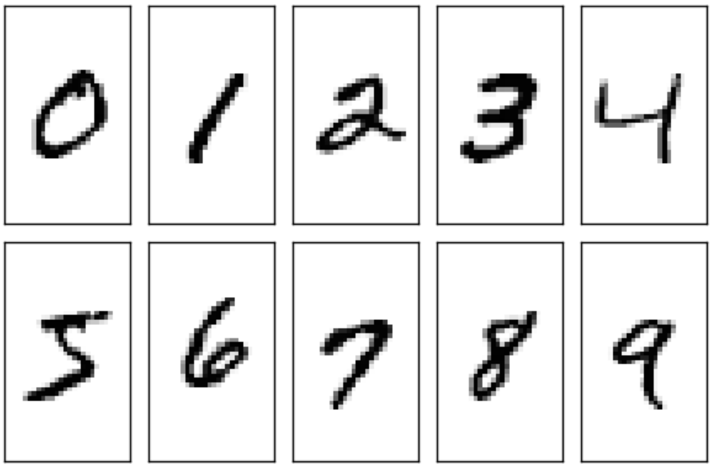

图 1: MINST手写数字示例(0-9)

二进制文件的方式读取MINST数据集，读取进来的图像像素值为0-255的数字，标签为0-9的数值。主要用到python中的open()和struct.unpack_from()函数。流程为：首先打开二进制文件，然后解析数据（大端存储）：魔数，维度，接下来解析为像素值并转换为numpy类型的数组。实现代码如下所示：

```
def decode_idx3(self,idx3_filepath):
    #read binary data
    binar_data = open(idx3_filepath,'rb').read()
    #magic num,dimension
    offset = 0
    fmt_header = ">iiii" # >:Big end storage i:Analytic type int
    magic_num, number_images,num_rows,num_cols = struct.unpack_from(fmt_header,binar_data,
                                                 offset)
    #Parse to get the data set
    image_size = num_rows * num_cols
    offset += struct.calcsize(fmt_header)
    #print(offset)
    fmt_image = '>' + str(image_size) + 'B'
    #print(fmt_image,offset,struct.calcsize(fmt_image))
    images = np.empty((number_images,num_rows,num_cols))
```

```
        #plt.figure
        for i in range(number_images):
            if (i + 1) % 10000 == 0:
                print('Have parsed %d photos' %(i+1))
                #print(offset)
            images[i] = np.array(struct.unpack_from(fmt_image,binar_data,offset)).reshape((
                                                    num_rows, num_cols))
            #print(images[i])
            offset += struct.calcsize(fmt_image)
        return images

    #decode the idx1 files
    def decode_idx1(self,idx1_filepath):
        #read binary data
        binar_data = open(idx1_filepath,'rb').read()
        #Parse header file
        offset = 0
        fmt_header = '>ii'
        magic_number, num_images = struct.unpack_from(fmt_header, binar_data, offset)
        #Parse data set
        offset += struct.calcsize(fmt_header)
        fmt_image = '>B'
        labels = np.empty(num_images)
        for i in range(num_images):
            if (i + 1) % 10000 == 0:
                print('Have parsed %d photos' %(i + 1))
            labels[i] = struct.unpack_from(fmt_image,binar_data,offset)[0]
            offset += struct.calcsize(fmt_image)
        return labels

    #divide the data
```

2.2 sigmoid函数及极大似然估计实现

定义sigmoid函数，似然函数以及梯度上升法求解使得似然函数取得最大值的参数$\boldsymbol{\theta}$，在求解梯度上升法的时候，使用矩阵进行运算加速运算速度，减少循环遍历提高效率。代码实现如下：

```
        return 1.0 / (1 + np.exp(-z))

    def max_likelihood_estimate(self,theta,x,y):
        z = theta * x
        h_x = Logistic_regression.sigmoid_function(self,z)
        likelihood_function = np.sum(y.T * np.log(h_x) + (1 - y).T * np.log(1 - h_x))
        return likelihood_function
    #logistic main part
    def main_logistic_gradient_ascent(self,training_data_x,training_data_y,max_iteration,alpha
                                      ,epsilon):
        training_X = np.array(training_data_x)
        training_Y = np.array(training_data_y)
        print(training_Y.shape)
        # test_array = np.array([[1,2,3],[1,5,6]])
        # test_array_theta = np.array([0.2,0.3,0.6])
        # result = np.dot(test_array,test_array_theta)
        # print(result)
        # z = Logistic.sigmoid_function(self,result)
        # print(z)
        m,n = training_X.shape
        counter = 0
```

```
        theta = np.ones(n)  #init
        #print(training_X.shape,theta.shape)
        error_0 = 0
        while counter < max_iteration:
            counter += 1
            error_1 = 0
            predict_Y = Logistic_regression.sigmoid_function(self, np.dot(training_X,theta))
            #print(predict_Y.shape)
            error = training_Y - predict_Y
            #print(error.shape)
            #calculate the gradient
            gradient = np.dot(training_X.T,error.T)
            #print(gradient.shape)
            theta = theta + alpha * gradient
            error_1 = np.sum(error)
            if abs(error_1 - error_0) < epsilon:
```

2.3 数据预测及正确率计算

之前按照1:4的比例划分了数据，48000个样本训练得到了参数$\boldsymbol{\theta}$，12000个样本作为测试集，用来进行预测，并对照实际数据，求解准确率。实现代码如下：

```
                break
            else:
                error_0 = error_1
            print(error_0)
        return error_0,theta,counter
    def predicting_data(self,theta, testing_data):
        row, col = testing_data.shape
        predict_label = np.zeros((row,1))
        for i in range(row):
            predict_label[i] = np.dot(theta,testing_data[i])
            if predict_label[i] >= 0.5:
                predict_label[i] = 1
            else:
                predict_label[i] = 0
        return predict_label
    def accurate_value(self,real_label,predict_label):
        total = len(real_label)
```

三 结果及图形展示

首先给出主函数代码：

```
        for i in range(total):
            if real_label[i] == predict_label[i]:
                counter += 1
        return counter / total

if __name__ == '__main__':
    Logistic = Logistic_regression
    binar_x = Logistic.decode_idx3(Logistic,'train-images-idx3-ubyte')
    shape_number = binar_x.shape  #three dimensions
    #print(shape_number[0],shape_number[1],shape_number[2])
    length = shape_number[0]
    width = shape_number[1] * shape_number[2]
```

```python
    #print(width)
    data_resolved = np.empty([length,width])
    for k in range(length):
        data_resolved[k] = np.ravel(binar_x[k])  #change two dimension to one dimension as
                                                  data_resolved
    #print(data_resolved.shape)
    binar_y = Logistic.decode_idx1(Logistic,'train-labels-idx1-ubyte')
    shape_number_y = binar_y.shape
    #print(shape_number_y)
    #print(binar_y)
    training_set = np.c_[data_resolved,binar_y]  #array joining
    #print(training_set.shape)
    training_set_new = training_set[np.where((binar_y == 0) | (binar_y == 1))] #we only need
                                            the '0' and '1' samples in minst
    #print(training_set_new.shape)
    x_train,x_test,y_train,y_test = Logistic.data_divide(Logistic,training_set_new)
    #print(x_train.shape,x_test.shape,y_train.shape,y_test.shape)
    alpha = 0.001
    max_iteration = 1000
    epsilon = 0.0001
    error_result,theta,counter = Logistic.main_logistic_gradient_ascent(Logistic,x_train,
                                            y_train,max_iteration,alpha,epsilon)
    print('iteration number: %d'%(counter))
```

参数设置如下：学习率：0.001；最大迭代次数：1000次；循环退出的条件：前后两次训练误差小于0.0001即退出迭代。运行程序我们得到的结果如下（给出部分θ的数值）：

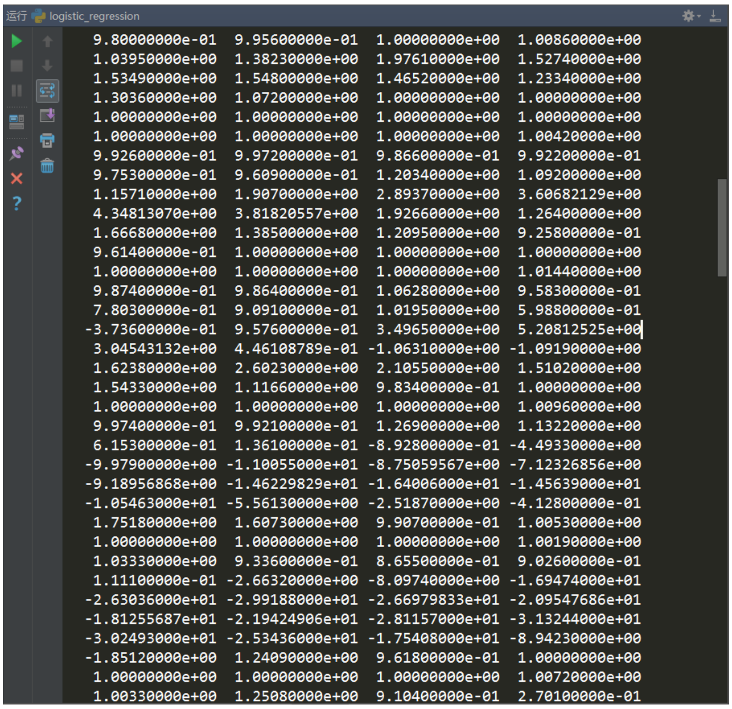

图 2: 部分LogisticRegression系数

最终预测的准确率为：99.48%

图 3: LogisticRegression Accuracy

四　总结体会

Logistic回归的主要思想还是来自于线性回归，如果说线性回归是对于特征的线性组合来拟合真实标记的话($y = \omega x + b$)，那么Logistic回归就是对于特征的线性组合来拟合真实标记的对数几率，($\ln(y/(1-y)) = \omega x + b$).另外在阅读文献的时候，我发现对于二分类问题，有两种不同形式的损失函数。一种是交叉熵损失，另一种log损失，而实际上这两种损失函数实际表达的意思是一致的。
交叉熵损失：

$$L(\omega) = \sum_{i=1}^{m} -y_i \log(h(x_i)) - (1-y_i)\log(1-h(x_i)) \quad y_i = 0 \,\mathrm{or}\, 1 \tag{10}$$

log损失：

$$L(\omega) = \sum_{i=1}^{m} \log(1 + e^{-\omega x_i t_i}) \quad t_i = -1 \,\mathrm{or}\, 1 \tag{11}$$

而实际上，无论是交叉熵还是log损失，虽然形式上不同，但意义上是相同的，只不过对应的标签一个是0和1，还有一个是-1和1.（其中交叉熵中的0标签与log损失中的-1标签）

在进行数值计算时，使用Matrix进行计算会比使用使用循环遍历计算来的快，这次的Logistic Regression相比之前的线性回归，我使用到了更多的矩阵运算，整体性的对数据集进行变换计算，在计算机中运行的速度更快，算是一个小的tip.

参考文献

[1] https://see.stanford.edu/materials/aimlcs229/cs229-notes1.pdf.

[2] https://blog.csdn.net/panrenlong/article/details/81736754.