模式识别与统计学习实验三：Softmax Regression实验报告

郭宇航 2016100104014

一　实验原理

1.1　指数分布族

指数分布族是指可以表示为指数形式的概率分布。指数分布族[1]的形式如下：

$$f_X(x|\theta) = h(x)\exp(\sum_{i=1}^{s}\eta_i(\theta)T_i(x) - A(\theta)) \tag{1}$$

其中$\eta_i(\theta)$表示为自然参数，$T_i(x)$充分统计量，$A(\theta)$对数分配函数，用于归一化。其表达式可以写为：

$$A(\eta) = \ln(\int_x h(x)\exp(\eta(\theta)\cdot T(x))\mathrm{d}x) \tag{2}$$

我们常见的伯努利分布，高斯分布均属于指数分布族，将指数分布族的表达式改写为：$P(y,\eta) = b(y)\exp(\eta^T T(y) - a(\eta))$.若以高斯分布为例，我们可以将高斯分布改写为指数分布族的标准形式：

$$\begin{cases} b(y) = \frac{1}{\sqrt{2\pi}\sigma\exp(-\frac{y^2}{2\sigma^2})} \\ \eta = \mu/\sigma \\ T(y) = y/\sigma \\ a(\eta) = \frac{\mu^2}{2\sigma^2} \end{cases} \tag{3}$$

关于指数分布族的发现和构造，大致可以描述为在无数可以对已有数据进行描述的分布中，选取一个熵最大的分布，将问题转化为一个优化问题后可以得到：

$$p^* = \underset{p\in P}{\operatorname{argmax}}\, H(p)\text{subject to } E_p[\phi_\alpha(\boldsymbol{X})] = \mu_\alpha \tag{4}$$

经过一系列的泛函求导之后可以得到指数分布族[2]。关于具体的推导难度较大，不是实验的重点这里不多赘述。

1.2　广义线性模型(GLM)

之所以引入了指数分布族是因为广义线性模型的介绍做铺垫，之前的实验一和实验二我们探究了线性回归模型和经过sigmoid函数变换过的逻辑回归。这两个模型都属于广义线性模型的特例。而建立广义线性模型我们需要考虑三个假设[3]：

- 给定x和θ之后，$P(y|x;\theta)$服从指数分布族。
- 给定的x,预测结果$T(y)$的期望值为$h_\theta(x) = E(T(y)|x)$
- η与x之间存在线性关系，即表示为：$\eta = \theta^T x$

关于广义线性模型有这样的重要结论：广义线性模型通过假设$P(y|x;\theta)$服从的分布，构造起来一个模型。如果我们假设y服从的是高斯分布，$\eta = \mu$则我们得到的是线性最小二乘模型；如果我们假设y服从伯努利分布，η与φ存在logistic函数的关系，我们得到的就是逻辑回归的模型。

1.3 Softmax Regression

引入指数分布族和广义线性模型后，下面引入Softmax Regression，Softmax是假设$P(y|x;\theta)$服从多项式分布的模型，也可以理解为由逻辑回归模型的二分类问题一般化为k分类问题，是一种更加泛化的模型。可以表示为：

$$y\ in 1, 2, \cdots, k \qquad P(y=i) = \varphi_i \qquad \sum_{i=1}^{k} \varphi_i = 1 \tag{5}$$

上述的表达式其实是有一些冗余的，可以只保留(k-1)个参数，由于$\varphi_k = 1 - \sum_{i=1}^{k-1} \varphi_i$.
为了让多项式分布能够写为指数分布族的形式，我们先引入$T(y)$：

$$T(1) = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad T(2) = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} \quad \cdots \quad T(k-1) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \quad T(k) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{6}$$

与此同时我们引入一个指示函数，用于后续的布尔矩阵的构造，这个指示函数定义为：$I(\text{True}) = 1, I(\text{False}) = 0$.因此可以得到：$T(y_i) = I(y = i)$，进一步我们又可以得到：

$$E(T(y_i)) = \sum_{i=1}^{k} T(y)_i \varphi_i = \sum_{i=1}^{k} I(y=i)\varphi_i = \varphi_i \tag{7}$$

通过上式我们可以将$P(y;\theta)$写为如下连乘的形式：

$$\begin{aligned} P(y;\theta) &= \prod_{i=1}^{k} \varphi_i^{I(y=i)} \\ &= \varphi_1^{I(y=1)} \varphi_2^{I(y=2)} \cdots \varphi_k^{1-\sum_{i=1}^{k-1} I(y=i)} \\ &= \exp(I(y=1)\log\varphi_1 + I(y=2)\log\varphi_2 + \cdots + (1 - \sum_{i=1}^{k-1} I(y=i))\log\varphi_k) \\ &= \exp(I(y=1)\log(\varphi_1/\varphi_k) + I(y=2)\log(\varphi_2/\varphi_k) + \cdots + I(y=k-1)\log(\varphi_{k-1}/\varphi_k) + \log\varphi_k) \\ &= b(y)\exp(\eta T(y) - a(\eta)) \end{aligned} \tag{8}$$

其中：

$$\eta = \begin{bmatrix} \log(\varphi_1/\varphi_k) \\ \log(\varphi_2/\varphi_k) \\ \vdots \\ \log(\varphi_{k-1}/\varphi_k) \end{bmatrix} \quad b(y) = 1 \quad a(\eta) = -\log\varphi_k$$

又由于我们之前得到：$\sum_{i=1}^{k}\varphi_i=1$,，因此改写后不难发现：

$$\begin{aligned}\eta_i&=\log(\varphi_i/\varphi_k)\Rightarrow\varphi_i=\varphi_k e^{\eta_i}\\ \sum_{i=1}^{k}\varphi_i&=\sum_{i=1}^{k}=\varphi_k e^{\eta_i}=1\Rightarrow\varphi_k=\frac{1}{\sum_{i=1}^{k}e^{\eta_i}}\\ \varphi_i&=\frac{e^{\eta_i}}{\sum_{j=1}^{k}e^{\eta_j}}\end{aligned}\tag{9}$$

导出对于每一类别对应的概率φ_i的表达式后，我们假设$h_\theta(x)=E[T(y)|x;\theta]$，得到如下的表达式：

$$\begin{aligned}h_\theta(x)&=E[T(y)|x;\theta]\\ &=E\left[\begin{array}{c}I(y=1)\\ I(y=2)\\ \vdots\\ I(y=k-1)\end{array}\middle|x;\theta\right]=\begin{bmatrix}\varphi_1\\ \varphi_2\\ \vdots\\ \varphi_{k-1}\end{bmatrix}=\begin{bmatrix}\frac{e^{\theta_1^T x}}{\sum_{j=1}^{k}e^{\theta_j^T x}}\\ \vdots\\ \frac{e^{\theta_{k-1}^T x}}{\sum_{j=1}^{k}e^{\theta_j^T x}}\end{bmatrix}\end{aligned}\tag{10}$$

再将样本点都纳入考虑，这里我们仍然采用的是MINST数据集，与逻辑回归不同，这一次我们将所有的数据全部纳入考虑，手写数字共有10类不同的标签数据。适用于Softmax Regression的多分类问题。这里对参数θ的估计我们仍然采用的是极大似然估计。基于之前的标签分布，给出似然函数：

$$L(\theta)=\prod_{i=1}^{m}P(y^{(i)}|x^{(i)};\theta)=\prod_{i=1}^{m}\prod_{j=1}^{k}\varphi_j^{I(y^{(i)}=j)}\tag{11}$$

进行对数变换后：

$$l(\theta)=\sum_{i=1}^{m}\sum_{j=1}^{k}I(y^{(i)}=j)\log\frac{e^{\theta_j^T x^{(i)}}}{\sum_{j=1}^{k}e^{\theta_j^T x^{(i)}}}\tag{12}$$

取Loss function为：

$$J(\theta)=-\frac{1}{m}\left(\sum_{i=1}^{m}\sum_{j=1}^{k}I(y^{(i)}=j)\log\frac{e^{\theta_j^T x^{(i)}}}{\sum_{j=1}^{k}e^{\theta_j^T x^{(i)}}}\right)\tag{13}$$

由此关于Softmax Regression的求解问题转换为了一个优化问题，我们希望寻找这个损失函数的极小值，因此继续沿用之前的梯度下降法（或者也可以求解极大似然估计的极大值，使用梯度上升法等）。下面给出梯度的求解公式：

$$\begin{aligned}\nabla_{\theta_j}J(\theta)&=-\frac{1}{m}\sum_{i=1}^{m}\left[\nabla_{\theta_j}\sum_{j=1}^{k}I(y^{(i)}=j)\log\frac{e^{\theta_j^T x^{(i)}}}{\sum_{j=1}^{k}e^{\theta_j^T x^{(i)}}}\right]\\ &=-\frac{1}{m}\sum_{i=1}^{m}\left[I(y^{(i)}=j)\cdot\left(1-\frac{e^{\theta_j^T x^{(i)}}}{\sum_{j=1}^{k}e^{\theta_j^T x^{(i)}}}\right)\boldsymbol{X}^{(i)}\right]\\ &=-\frac{1}{m}\sum_{i=1}^{m}\left[\boldsymbol{X}^{(i)}\left(I(y^{(i)}=j)-P(y^{(i)}=j|\boldsymbol{X}^{(i)};\theta)\right)\right]\end{aligned}\tag{14}$$

需要注意的是这里的θ_j表示的是一个向量，需要通过梯度下降法进行迭代更新：

$$\theta_j=\theta_j+\alpha\nabla_{\theta_j}J(\theta)\tag{15}$$

1.4 注意事项

Softmax Regression中的参数是冗余的，或者说是被过度参数化[4]，即会存在多组满足我们设定的条件的系数矩阵。若我们采用牛顿法去迭代，可能会出现数值计算的问题：Hessian矩阵是不可逆的。为了解决这个问题我们在损失函数中添加了正则化项：

$$J(\theta) = -\frac{1}{m}\left(\sum_{i=1}^{m}\sum_{j=1}^{k} I(y^{(i)}=j)\log\frac{e^{\theta_j^T x^{(i)}}}{\sum_{j=1}^{k} e^{\theta_j^T x^{(i)}}}\right) + \frac{\lambda}{2}\sum_{i=1}^{k}\sum_{j=0}^{n}\theta_{ij}^2 \tag{16}$$

加入正则化后的损失函数变为了严格的凸函数，因此不会出现之前的Hessian矩阵奇异的情况，此时的偏导数为：

$$\nabla_{\theta_j} J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[\boldsymbol{X}^{(i)}\left(I(y^{(i)}=j) - P(y^{(i)}=j|\boldsymbol{X}^{(i)};\theta)\right)\right] + \lambda\theta_j \tag{17}$$

二 Python代码实现

2.1 读取数据

读取数据的过程在逻辑回归中已经进行了讲解，这里就不再赘述，由于之前在逻辑回归的代码中将这个读取数据的过程写成了一个类中的方法，这里直接对这个方法进行调用即可读取数据。读取的代码如下：

```
    def load_data(self,filepath):
        '''
        :param filepath: one list,Store four file paths
        :return: four array or matrix
        '''
        idx3_data_train = logistic_regression.Logistic_regression.decode_idx3(
                                                    logistic_regression,filepath[0])
        idx3_data_test = logistic_regression.Logistic_regression.decode_idx3(
                                                    logistic_regression,filepath[1])
        idx1_data_train = logistic_regression.Logistic_regression.decode_idx1(
                                                    logistic_regression,filepath[2])
        idx1_data_test = logistic_regression.Logistic_regression.decode_idx1(
                                                    logistic_regression,filepath[3])
        return idx3_data_train,idx1_data_train,idx3_data_test,idx1_data_test

    #rebuild matrix
    def rebulit_data(self,idx3_data):
        shape_number = idx3_data.shape
        #print(shape_number)
        length = shape_number[0]
        width = shape_number[1] * shape_number[2]
        data_rebuilt = np.empty([length,width])
        for k in range(length):
            data_rebuilt[k] = np.ravel(idx3_data[k])
        data_one = np.ones(length)
        #print(np.c_[data_rebuilt,data_one])
        return np.c_[data_rebuilt,data_one]
```

2.2 梯度下降求解系数

在理论推导的最后部分式14我们给出对于Softmax Regression的极大似然函数构造出的损失函数的梯度公式，基于这个公式以及MINST数据集我们进行代码端的实现：

```
    def bool_function(self,y,label):
        return y == label
    def main_softmax_gradient(self,data_x,data_y,theta,alpha,lambda_regular):
        '''
        :param data_x:60000 * 785
        :param data_y: 60000 * 1
        :param theta: 10 * 785
        :param maxiteration: maximum iteration times
        :param alpha: learning rate
        :return:
        '''
        data_x = data_x
        #data_y = data_y + 1
        length,width = data_x.shape
        #print('data_dimension:',length,width)
        label_number = len(np.unique(data_y))
        label = np.arange(1,label_number+1)
        bool_matrix = label_binarize(data_y, classes=np.unique(data_y).tolist()).reshape(
                                                    length,label_number)
        #print('bool_matrix dimension:',bool_matrix.shape,bool_matrix)# binary (m*k)
        #print('biaoqian shuliang',label_number)
        data_x_new = data_x.transpose() # n+1 * m
        #print('new data dimension:',data_x_new.shape)
        theta_data_x_new = theta.dot(data_x_new) # k * m
        theta_data_x_new = theta_data_x_new - np.max(theta_data_x_new)
        #print(np.exp(theta_data_x_new))
        possible_theta_data = np.exp(theta_data_x_new) / np.sum(np.exp(theta_data_x_new),axis=
                                                    0)
        #print(possible_theta_data)
        cost_value = (-1 / length) * np.sum(np.multiply(bool_matrix,np.log(possible_theta_data
                                                    ).T)) + (lambda_regular / 2) * np.sum(
                                                    np.square(theta))   #1 * 1
        gradient = (-1 / length) * (data_x_new.dot(bool_matrix - possible_theta_data.T)).T +
                                                    lambda_regular * theta   # k * n+1
        return cost_value,gradient
```

2.3 数据训练

训练数据为60000张标签为0-9的手写数字，输入参数为：训练集数据（数据维度：60000*785），标签训练集数据（60000*1），学习率，最大的迭代次数以及退出迭代的参数。输出值为θ系数矩阵（10*785）实现代码如下：

```
    #train data
    def data_training(self,data_x,data_y,max_iteration,alpha,lambda_regular,epsilon):
        m,n = data_x.shape
        label_number = len(np.unique(data_y))
        #define one coefficients matrix,softmax: k kinds labels,every label has (width)
                                                    coefficients
        theta = np.random.randn(label_number,n)
        print(theta)
        counter = 0
        error_0 = 0
        while counter < max_iteration:
            counter += 1
            error_1 = 0
            cost_value,gradient = Softmax_regression.main_softmax_gradient(self,data_x,data_y,
                                                        theta,alpha,lambda_regular)
```

```
            print('Error value: %f' %(cost_value))
            #print(gradient)
            theta = theta - alpha * gradient
            error_1 = cost_value
            #print(theta)
            if abs(error_1 - error_0) < epsilon:
                break
            else:
                error_0 = error_1
            #print(error_0)
        return theta
```

2.4 数据测试

测试数据集为10000张标签为0-9的手写数字，输入参数为：测试集数据（数据维度：10000*785），标签测试数据（10000*1）以及通过数据训练得到的θ系数矩阵，输出参数为每个样本的预测值以及所有测试样本的预测准确率。实现代码如下：

```
    #testing data
    def predict(self,theta,testdata_x, testdata_y):  # testdata (10000 * 785)
        #testdata_y = testdata_y + 1
        prod = theta.dot(testdata_x.T)
        pred = np.exp(prod) / np.sum(np.exp(prod), axis=0)
        pred = pred.argmax(axis=0)
        accuracy = 0.0
        for i in range(len(testdata_y)):
            if testdata_y[i] == pred[i]:
                accuracy += 1
        return pred, float(accuracy / len(testdata_y))
```

三 结果及图形展示

给出主函数的代码并加以解释：

```
if __name__ == '__main__':
    softmax = Softmax_regression
    filepath = ['train-images-idx3-ubyte','t10k-images-idx3-ubyte','train-labels-idx1-ubyte','
                                                t10k-labels-idx1-ubyte']
    data_trainX,data_trainY,data_testX,data_testY = softmax.load_data(softmax,filepath)
    #print(data_trainX.shape,data_trainY.shape,data_testX.shape,data_testY.shape)
    data_trainX = softmax.rebulit_data(softmax,data_trainX)
    data_trainX = data_trainX / 255
    data_testX = softmax.rebulit_data(softmax,data_testX)
    data_testX = data_testX / 255
    #print(data_trainX.shape,data_testX.shape)
    max_iteration = 1000
    alpha = 0.1
    lambda_regular = 0.001
    epsilon = 0.001
    theta_result = softmax.data_training(softmax,data_trainX,data_trainY,max_iteration,alpha,
                                                lambda_regular,epsilon)
    print(theta_result)
    prediction,accuracy = softmax.predict(softmax,theta_result,data_testX,data_testY)
    print('Softmax_Regression Prediction Accuracy is percentage:%f.' %(accuracy * 100))
```

首先我们对所有的数据进行标准化，之所以标准化是因为若不进行标准化操作则在取指数的时候会出现数据值过大，极大值无法求解的问题，因此需要对数据进行标准化，这里的标准化比较简单，由于我们每个像素点介于0-255之间，所以我们对每个像素点进行除以255以标准化。关于参数的设置：最大的迭代次数设为1000次，学习率（步长）为0.1，正则化系数为0.001，迭代退出的参数为：0.001.最终给出损失函数的变化结果和预测的准确率的结果展示（图1和图2），我们看到最终的预测准确率大概在80.59%.

```
已解析10000张                                                    Error value: 21.645921
已解析10000张                                                    Error value: 20.875787
已解析20000张                                                    Error value: 20.214436
已解析30000张                                                    Error value: 19.643993
已解析40000张                                                    Error value: 19.147745
已解析50000张                                                    Error value: 18.711029
已解析60000张                                                    Error value: 18.321664
已解析10000张                                                    Error value: 17.969682
[[ 1.68013087 -0.52506151 -0.57455001 ...  2.09077929  0.59632505  Error value: 17.647140
   0.60000042]                                                   Error value: 17.347822
 [ 0.40976199 -0.82842269 -0.96323407 ...  1.83774383 -0.45630534  Error value: 17.066945
   1.33317758]                                                   Error value: 16.800862
 [ 0.6556405   0.2909807  -0.04549001 ... -1.94325725 -0.27849877  Error value: 16.546789
  -0.40607183]                                                   Error value: 16.302603
 ...                                                             Error value: 16.066695
 [ 0.51695169  1.0571826   1.95729638 ... -0.94950589  0.02340949  Error value: 15.837846
  -0.7977949 ]                                                   Error value: 15.615135
 [-0.24914637 -1.74755889  0.17011915 ... -0.25212765 -0.98284025  Error value: 15.397861
   0.28491945]                                                   Error value: 15.185495
 [ 1.42665175  0.91633986 -2.42947895 ...  1.35097885  0.78320249  Error value: 14.977635
   0.9052986 ]]                                                  Error value: 14.773989
```

图 1: 读取图片，初始化系数矩阵以及初识的损失函数

```
Error value: 3.928410
Error value: 3.927279
Error value: 3.926149
Error value: 3.925020
Error value: 3.923892
Error value: 3.922765
Error value: 3.921639
[[ 1.52023768 -0.47509293 -0.51987174 ...  1.89180587  0.53957452
   0.5439702 ]
 [ 0.37076613 -0.7495841  -0.87156587 ...  1.66285107 -0.41288008
   1.207518  ]
 [ 0.59324509  0.26328891 -0.04116085 ... -1.75832308 -0.25199485
  -0.36710116]
 ...
 [ 0.46775489  0.95657359  1.77102615 ... -0.85914416  0.02118168
  -0.72091046]
 [-0.22543583 -1.58124876  0.1539294  ... -0.22813339 -0.88930619
   0.25499534]
 [ 1.29088143  0.82913445 -2.19827247 ...  1.2224101   0.70866738
   0.81839364]]
Softmax_Regression Prediction Accuracy is percentage:80.590000.
```

图 2: 最终的损失函数与预测准确率

四　总结体会

Softmax Regression的引入过程其实相比实现过程更加吸引我的关注，因为从指数分布族到广义线性模型再到Softmax，我发现在我们看完广义线性模型之后我会回过头来再去看前面的Linear

Regression和Logistic Regression.从更加宏观的角度去看这两个模型可以从标签概率分布的假设以及中间函数的变换过程去理解。其实线性回归，逻辑回归，Softmax回归三者之间存在相同的本质，内核是一致的。这样去理解模型可能会有更深刻的印象。

另外的话，在进行数值求解的时候沿用了矩阵运算的方式，但是还是采用了梯度上升的方法，对于MINST数据集上进行Softmax Regression的计算，还是比较耗费时间，运行一次大概2,3分钟的样子，在数值求解的计算上还需要进一步的优化。

参考文献

[1] Wikipedia, "Exponential family," https://en.wikipedia.org/wiki/Exponential_family.

[2] 民科智能, "指数分布族的构造," https://www.zhihu.com/question/52240493/answer/282469915.

[3] A. Ng, "Cs229 lecture notes," https://see.stanford.edu/materials/aimlcs229/cs229-notes1.pdf.

[4] 田湾第一帅, "Softmax regression模型," May 2018, https://www.cnblogs.com/wanshuai/p/9104518.html.