ex1：Softmax Regression实验报告

📌 姓名：陈睿颖

学号：2013544

专业：计算机科学与技术

1. 程序实现原理

1.1 Softmax Regression

Softmax Regression是Logistic回归的推广，Logistic回归是处理二分类问题的，而Softmax Regression是处理多分类问题的。Logistic回归是处理二分类问题的比较好的算法，具有很多的应用场合，如广告计算等。Logistic回归利用的是后验概率最大化的方式去计算权重。

1.1.1 Logistic回归的推广

在Logistic回归需要求解的是两个概率：

$$P(y=1|x;\theta)$$

和

$$P(y=0|x;\theta)$$

而在Softmax Regression中将不是两个概率，而是k个概率，k表示的是分类的个数。我们需要求出以下的概率值：

$$h_\theta\left(x^{(i)}\right)=\left(\begin{array}{c}P\left(y^{(i)}=1 \mid x^{(i)} ; \theta\right) \\ P\left(y^{(i)}=2 \mid x^{(i)} ; \theta\right) \\ \cdots \\ P\left(y^{(i)}=k \mid x^{(i)} ; \theta\right)\end{array}\right)=\frac{1}{\sum_{j=1}^k e^{\theta_j^T x^{(i)}}}\left[\begin{array}{c}e^{\theta_1^T x^{(i)}} \\ e^{\theta_2^T x^{(i)}} \\ \cdots \\ e^{\theta_k^T x^{(i)}}\end{array}\right]$$

此时的损失函数为

$$J(\theta)=-\frac{1}{m}\left[\sum_{i=1}^m \sum_{j=1}^k I\left\{y^{(i)}=j\right\} \log \frac{e^{\theta^T x^{(i)}}}{\sum_{l=1}^k e^{\theta_l^T x^{(i)}}}\right]$$

1.1.2 损失函数的由来

概率函数可以表示为

$$P(y \mid x ; \theta)=\prod_{j=1}^k\left(\frac{e^{\theta_j^T x}}{\sum_{l=1}^k e^{\theta_l^T x}}\right)^{I\{y=j\}}$$

其似然函数为

$$L(\theta)=\prod_{i=1}^m \prod_{j=1}^k\left(\frac{e^{\theta_j^T x}}{\sum_{l=1}^k e^{\theta_l^T x}}\right)^{I\{y=j\}}$$

log 似然函数为

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

我们要最大化似然函数，即求

$$max\ l(\theta)$$

再转化成损失函数。

1.1.3 对损失函数求偏导

若仅取一个样本，则可表示为

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

下面对 $l(\theta)$ 求偏导：

$$\frac{\partial l(\theta)}{\partial \theta_j^{(m)}} = \sum_{j=1}^{k} I\{y=j\}\left(x^{(m)} - \frac{e^{\theta_j^T x}}{\sum_{l=1}^{k} e^{\theta_l^T x}} \cdot x^{(m)}\right) = [I\{y=j\} - P(y=j \mid x;\theta)]x^{(m)}$$

其中，m表示第m维。如Logistic回归中一样，可以使用基于梯度的方法来求解这样的最大化问题。

1.1.4 梯度下降

更新每一个 θ_j^T ,需要计算 $J(\theta)$ 对每个 θ_j^T 的梯度

$$\nabla_{\theta_j} J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right)_j - I(y_i = j)\right)x^{(i)}$$

最后更新 θ ：

$$\theta := \theta - \alpha \begin{pmatrix} \nabla_{\theta_1} J(\theta)^T \\ \nabla_{\theta_2} J(\theta)^T \\ \vdots \\ \nabla_{\theta_k} J(\theta)^T \end{pmatrix}$$

1.2 Numpy包

1.2.1 np.dot()函数

该函数的功能如下：

- 向量点积
- 矩阵乘法

1.2.2 np.exp()函数

用于计算 e^x 的值

1.2.3 np.sum()函数

函数中可接收的参数是

```
np.sum(a, axis=None, keepdims=True)
```

其中a是用于进行加法运算的数组形式的元素；axis的取值在本实验中为1，当axis为1时,是压缩列,即将每一行的元素相加,将矩阵压缩为一列； `keepdims` 表示是否保持维度。

1.2.4 np.multiply()函数

数组的乘法运算。

1.2.5 np.log()函数

数组的log运算

2. 代码细节

2.1 softmax_regression函数

首先，先命名变量m，其含义对应1.1节公式中的m，为样本个数：

```
m=x.shape[0] #x是m*n矩阵，x.shape[0]即为m的值
```

在softmax regression 函数中，进行迭代的循环如下：

```
    for i in range(iters):
        theta_xi=np.dot(x,theta.T)
        fenzi=np.exp(theta_xi) #分子
        fenmu=np.sum(np.exp(theta_xi),axis=1,keepdims=True)#分母
        h_theta_xi=fenzi/fenmu
        J_theta=-np.sum(np.multiply(np.log(h_theta_xi),y.T).reshape(y.T.size,-1)
        f=J_theta
        print("i=",i,"loss=",f)
        g=np.dot((h_theta_xi-y.T).T,x)/m #下降的梯度
        theta=theta-alpha*g
```

下面对代码进行逐句分析

2.1.1 计算概率值

该部分计算公式为

$$h_\theta\left(x^{(i)}\right)=\begin{pmatrix}P\left(y^{(i)}=1 \mid x^{(i)};\theta\right)\\ P\left(y^{(i)}=2 \mid x^{(i)};\theta\right)\\ \cdots\\ P\left(y^{(i)}=k \mid x^{(i)};\theta\right)\end{pmatrix}=\frac{1}{\sum_{j=1}^{k} e^{\theta_j^T x^{(i)}}}\begin{bmatrix}e^{\theta_1^T x^{(i)}}\\ e^{\theta_2^T x^{(i)}}\\ \cdots\\ e^{\theta_k^T x^{(i)}}\end{bmatrix}$$

首先计算 $e^{\theta_j^T x^i}$ ，即为公式中的分子：

```
theta_xi=np.dot(x,theta.T)
fenzi=np.exp(theta_xi) #分子
```

接着计算 $\sum_{j=1}^{k} e^{\theta_j^T x^{(i)}}$ ，即分母部分：

```
fenmu=np.sum(np.exp(theta_xi),axis=1,keepdims=True)
```

概率值 $h_\theta\left(x^{(i)}\right)$ 为分子分母之比：

```
h_theta_xi=fenzi/fenmu
```

2.1.2 计算损失函数

有如下公式：

$$J(\theta)=-\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{j=1}^{k} I\left\{y^{(i)}=j\right\}\log\frac{e^{\theta^T x^{(i)}}}{\sum_{l=1}^{k} e^{\theta_l^T x^{(i)}}}\right]$$

在代码中实现如下：

```
J_theta=-np.sum(np.multiply(np.log(h_theta_xi),y.T).reshape(y.T.size,-1))/m #reshape控制为y.T.size=m行，列数系统自己定
```

上面所使用的值在2.1节中均有注解。至此损失函数值计算完成，将其放入变量f

```
f=J_theta
```

2.1.3 计算下降的梯度并更新 θ 值

使用的公式为：

$$\nabla_{\theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right)_j - I(y_i = j) \right) x^{(i)}$$

同样使用前面所计算好的变量，将下降的梯度值存入变量g

```
g=np.dot((h_theta_xi-y.T).T,x)/m #下降的梯度
```

根据公式

$$\theta := \theta - \alpha \begin{pmatrix} \nabla_{\theta_1} J(\theta)^T \\ \nabla_{\theta_2} J(\theta)^T \\ \vdots \\ \nabla_{\theta_k} J(\theta)^T \end{pmatrix}$$

更新 θ 值：

```
theta=theta-alpha*g
```

2.2 cal_accuracy函数

该部分完成了模型分类的正确率检验，使用的代码如下：

```
def cal_accuracy(y_pred, y):
    # TODO: Compute the accuracy among the test set and store it in acc
    m,n =y.shape
    ace=0
    for i in range(m):
        if y_pred[i]==y[i]:
            ace+=1
    accuracy = ace/m
    return accuracy
```

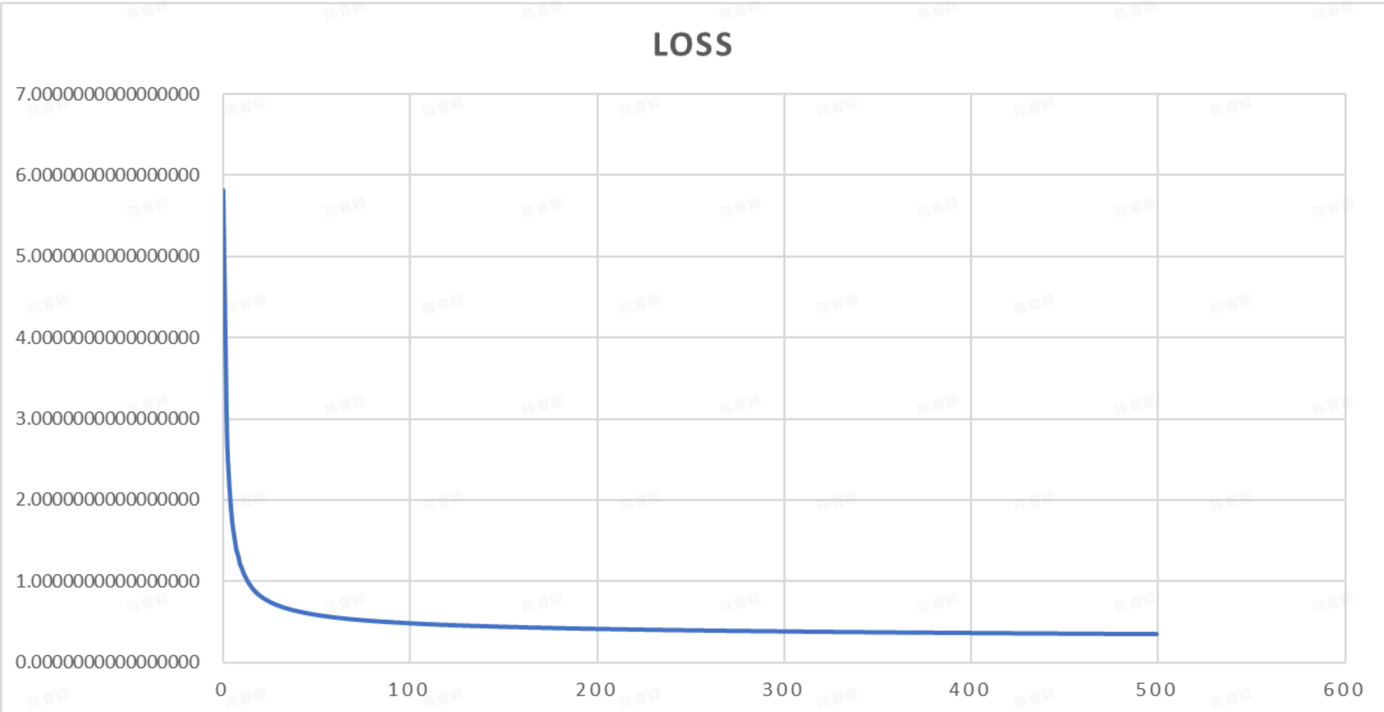
每次预测类别与实际值相等则正确，统计所有正确个数并计算正确率。

3. 实验结果及其分析

3.1 分析

经过多次实验，发现本实验模型准确率稳定在90%左右。某次实验的结果如下图：

对于每一次迭代的损失值进行了统计，并绘制了图表：



可以看到损失值在前期下降非常明显，随着迭代次数增多下降逐渐缓慢。对应了前面所介绍的损失函数公式。

3.2 总结

每一次实验所用的时间较长，搜集相关资料后尝试使用@jit的方法对程序进行提速，但效果并不是很明显。具体操作为：

- 在softmax_regression.py文件开头加入库函数 `from numba import jit`
- 在需要softmax_regression函数定义前上加上 `@jit(nopython=True)`

在以后的学习过程中，将注意解决这个问题