Softmax Regression实验报告

2010239 李思凡

一、程序实现原理

Softmax函数

Softmax函数能将一个含任意实数的K维向量z映射到另一个K维实向量$\sigma(z)$中，使得每个元素的范围在(0,1)间，且所有元素的和为1。

Softmax回归用于解决多分类问题，对于样本$\{(x^1,y^1),(x^2,y^2),\ldots,(x^m,y^m)\}$，有K个类别，即$y^i\in\{1,2,\ldots,K\}$。Softmax函数将估算样本$x^i$属于每一类别的概率。函数如下：

$$h_\theta(x^{(i)})=\begin{bmatrix}p(y^{(i)}=1|x^{(i)};\theta)\\ p(y^{(i)}=2|x^{(i)};\theta)\\ \vdots\\ p(y^{(i)}=k|x^{(i)};\theta)\end{bmatrix}=\frac{1}{\sum_{j=1}^{k}e^{\theta_j^Tx^{(i)}}}\begin{bmatrix}e^{\theta_1^Tx^{(i)}}\\ e^{\theta_2^Tx^{(i)}}\\ \vdots\\ e^{\theta_k^Tx^{(i)}}\end{bmatrix}$$

样本x^i被归为类型j的概率为：

$$p(y^{(i)}=j|x^{(i)};\theta)=\frac{e^{\theta_j^Tx^{(i)}}}{\sum_{l=1}^{k}e^{\theta_l^Tx^{(i)}}}$$

Softmax回归的代价函数如下所示。若$y^i=j$则该部分为1，否则为0。

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

最小化代价函数时采用梯度下降求解，得出梯度为：

$$-\frac{1}{m}\left[\sum_{i=1}^{m}x^{(i)}(1\left\{y^{(i)}=j\right\}-p(y^{(i)}=j|x^{(i)};\theta)\right]$$

更新时采用如下方式，其中α为超参数。

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

程序设计

目标

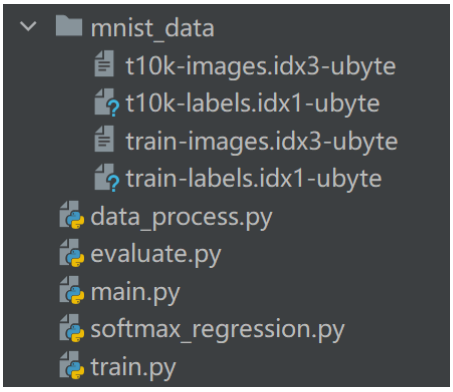
本程序要使用MNIST数据集，利用提供的训练样本训练出一个分类器，完成对数据集中0-9 10个手写数字的分类，在提供的测试集上测试并计算准确率。

经过简单分析，该问题为一个多分类问题，共有K类，适合采用Softmax回归进行分类。参照Softmax回归的数学原理，大体流程为：读取训练集和测试集数据——在训练集上多次迭代，得到合适的参数θ——使用训练得到的参数值，在测试集上测试。其中，每次训练的流程为：依次遍历各个样本，对样本x^i使用softmax函数得到其被归为各个类型的概率——计算代价函数——计算梯度——调整参数θ

。

框架

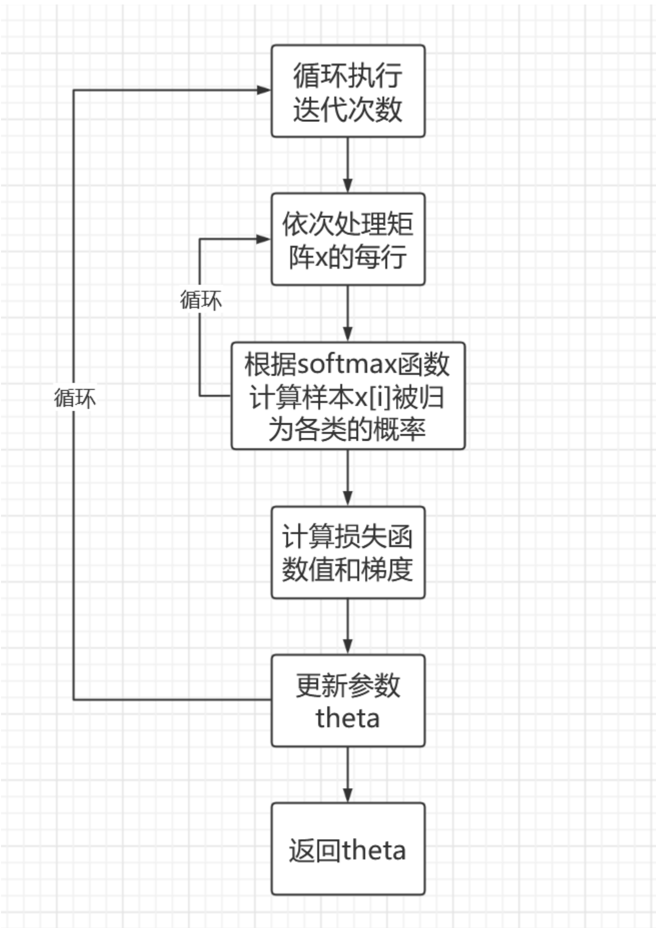
项目的框架如下图所示：



在mnist_data中存放训练数据集与测试数据集。main.py中依次实现读取数据，调用train.py中train方法训练，调用evaluate.py中predict方法预测、cal_accuracy方法计算准确率。

在train.py中，先调用data_process.py中data_convert方法，将训练样本和对应标签保存在矩阵x和y中；再随机初始化参数θ矩阵；调用softmax_regression.py中softmax_regression方法，得到训练后的θ矩阵。

主要实现的代码在softmax_regression.py中。实现的流程如下图所示：



还要在evaluate.py中实现计算准确率的方法，方式为依次判断测试集的预测矩阵与标签矩阵对应元素是否相当，计算相等元素所占比例。

二、实现的代码

softmax_regression方法

完整的softmax_regression()方法如下所示:

```
def softmax_regression(theta, x, y, iters, alpha):
    count = 0
    for it in range(iters):
        loss_i = 0
        grad_i = np.zeros((theta.shape[0], theta.shape[1]))
        for i in range(x.shape[0]):
            xx = x[i].reshape(-1, 1)
            yy = y[:, i].reshape(-1, 1)
            z = np.dot(theta, xx)
            row_max = np.max(z)
            z = z - row_max
            y_pre_i = np.exp(z) / np.sum(np.exp(z))
            loss_i += np.sum(yy * np.log(y_pre_i))
            grad_i += np.dot((y_pre_i-yy), xx.T)
        f = -(1/y.shape[1]) * loss_i
        print("loss: ", f)
        count += 1
        print("count: ", count)
        g = -(1/y.shape[1]) * grad_i
        theta += alpha * g
    return theta
```

1. 根据softmax函数计算样本x[i]被归为各类的概率:

```
            # calculate theta_t * x[i]
            # theta[k,n], 每一维是[1,n], xx[n,1]
            z = np.dot(theta, xx)
            # calculate softmax[i]
            row_max = np.max(z)
            z = z - row_max
            y_pre_i = np.exp(z) / np.sum(np.exp(z))    # y_pre_i[k,1]
```

首先计算出公式中 θ 与 x^i 的乘积，保存在z中；再使用公式求出样本x^i 被归为各类别的概率，保存在y_pre_i中。由于计算exp(z)时有可能出现一溢出的情况，减去最大值进行处理。

2. 计算损失函数和梯度

```
        for i in range(x.shape[0]):

            ... # 计算softmnax

            ''' calculate loss[i] '''
            loss_i += np.sum(yy * np.log(y_pre_i))
            ''' calculate grad[i] '''
            grad_i += np.dot((y_pre_i-yy), xx.T)
        ''' calculate loss '''
        f = -(1/y.shape[1]) * loss_i
        ''' calculate grad '''
        g = -(1/y.shape[1]) * grad_i
```

按照公式进行计算，共有m个样本，需要累加m次，在每次循环中进行累加，保存在loss_i和grad_i中；在循环结束后乘上-1/m得到每一轮迭代的损失值和梯度。

3. 更新参数 θ

```
        ''' update theta '''
        theta += alpha * g
```

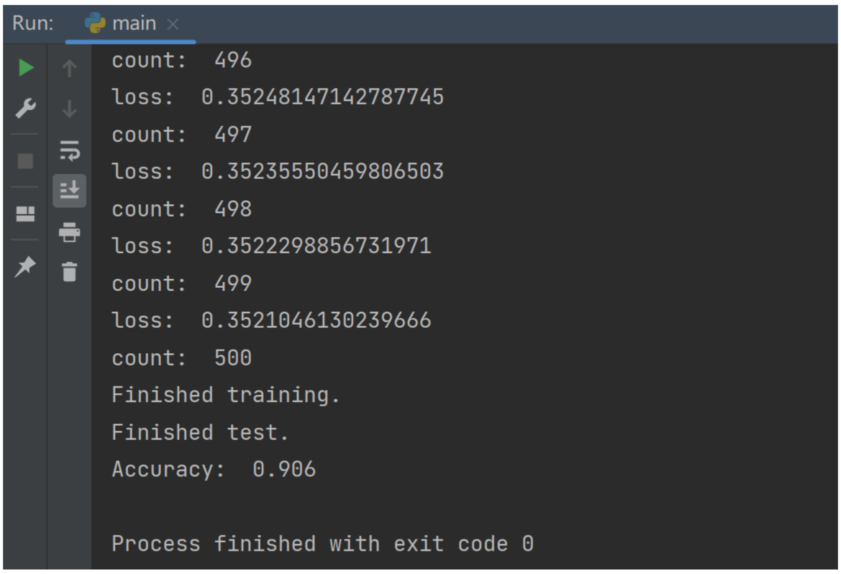
cal_accuracy方法

```
def cal_accuracy(y_pred, y):
    count = 0
    for i in range(y.shape[0]):
        if y_pred[i] == y[i]:
            count += 1
    acc = count / y.shape[0]
    return acc
```

统计预测正确的个数占总个数的比例即可。

三、实验结果及分析

实验结果如下图所示，成功完成数据读取、训练和预测，准确率为0.906。



打印了执行的次数和每次迭代后的损失值，观察可知损失值从5.1964逐渐降低到0.3521，且趋于稳定。最终结果的准确率较高。