实验三：逻辑回归实现 MINIST 手写数据集分类

2020090914019 钱琪豪

1. 实验原理

逻辑斯谛回归（logistic regression）是统计学习中的经典分类方法，属于对数线性模型，所以也被称为对数几率回归。虽然带有回归的字眼，但是该模型是一种分类算法，逻辑斯谛回归是一种线性分类器，针对的是线性可分问题。利用 logistic 回归进行分类的主要思想是：根据现有的数据对分类边界线建立回归公式，以此进行分类。这里的“回归”一词源于最佳拟合，表示要找到最佳拟合参数集，因此，logistic 训练分类器时的做法就是寻找最佳拟合参数，使用的是最优化方法。

1.1. Logistic 回归函数

事件的几率（odds），是指该事件发生的概率与该事件不发生的概率的比值。如果事件发生的概率是 p，那么该事件的几率是 p/(1-p)。取该事件发生几率的对数，定义为该事件的对数几率（log odds）或 logit 函数：

$$\mathrm{logit}(p) = \log\frac{p}{1-p}$$

将输出转换到整个实数范围内，这样就可以将对数几率记为输入特征值的线性表达式 ：

$$\mathrm{logit}(p(y=1|x)) = w_0x_0 + w_1x_1 + \ldots + w_nx_n = \sum_{i=0}^{n} w_ix_i = w^Tx$$

其中，p(y =1|x)是条件概率分布，表示当输入为 x 时，实例被分为 1 类的概率，依据此概率我们能得到事件发生的对数几率。但目标是做分类器，通过输入特征来判定图像属于几的概率。所以我们取 logit 函数的反函数，令的线性组合为输入，p 为输出，经如下推导

$$\log\frac{p}{1-p} = w^Tx$$

令$w^Tx = z$　　对上述公式取反

则有$\phi(z) = p = \frac{1}{1+e^{-z}}$　　（1）

1.2. 最佳回归系数的确定（极大似然估计+最优化方法）

先定义一个最大似然函数 L，假定数据集中的每个样本都是独立的，其计算公式如下：

$$L(w) = P(y|x;w) = \prod_{i=1}^{n} P\left(y^{(i)}|x^{(i)};w\right) = \left(\phi\left(z^{(i)}\right)\right)^{y^{(i)}}\left(1-\phi\left(z^{(i)}\right)\right)^{1-y^{(i)}}$$

L(w)就是对于损失函数的最原始的定义，还要进一步处理取对数，在进行极大似然估计的时候都要取对数。在似然函数值非常小的时候，可能出现数值溢出的情况，也就是数值在极小的时候因为无限趋近于 0 而默认其等于 0，使用对数函数降低了这种情况发生的可能性。其次，我们可以将各因子的连乘转换为和的形式，利用微积分中的方法，通过加法转换技巧可

以更容易地对函数求导。
取似然函数的对数:

$$J(w) = \log(L(w)) = \sum_{i=1}^{n} y^{(i)} \log\left(\phi\left(z^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - \phi\left(z^{(i)}\right)\right)$$

这里似然函数是取最大值的，我们可以直接将其确定为损失函数然后使用梯度上升算法求最优的回归系数。但是既然是损失函数，还是取最小值好一点，，所以我将以上公式除以-m 取反来使用梯度下降算法来求最小值
接下来就使用梯度下降求最小值。首先，计算对数似然函数对 j 个权重的偏导:

$$\frac{\partial}{\partial w_j} J(w) = \left(-y \frac{1}{\phi(z)} + (1-y) \frac{1}{1-\phi(z)}\right) \frac{\partial}{\partial w_j} \phi(z)$$

计算一下 sigmoid 函数的偏导:

$$\frac{\partial}{\partial w_j} \phi(z) = \frac{\partial}{\partial z} \frac{1}{1+e^{-z}} = \frac{1}{(1+e^{-z})^2} e^{-z} = \frac{1}{1+e^{-z}} \left(1 - \frac{1}{1+e^{-z}}\right) = \phi(z)(1-\phi(z))$$

代入（1）式中:

$$\begin{aligned} & \left(-y \tfrac{1}{\phi(z)} + (1-y) \tfrac{1}{1-\phi(z)}\right) \tfrac{\partial}{\partial w_j} \phi(z) \\ = & \left(-y \tfrac{1}{\phi(z)} + (1-y) \tfrac{1}{1-\phi(z)}\right) \phi(z)(1-\phi(z)) \tfrac{\partial}{\partial w_j} z \\ = & (-y(1-\phi(z)) + (1-y)\phi(z)) x_j \\ = & -(y - \phi(z)) x_j \end{aligned}$$

们的目标是求得使损失函数最小化的权重 w，所以按梯度下降的方向不断的更新权重:

$$w_j := w_j - \eta \sum_{i=1}^{n} \left(y^{(i)} - \phi\left(z^{(i)}\right)\right) x^{(i)}$$

2. Python 代码实现

2.1. 从零实现

2.1.1. 导入数据集

```
def read_image(file_name):
    # 先用二进制方式把文件都读进来
    file_handle = open(file_name, "rb")  # 以二进制打开文档
    file_content = file_handle.read()  # 读取到缓冲区中
    offset = 0
    head = struct.unpack_from('>IIII', file_content, offset)  # 取前 4 个整数，返回一个元组
    offset += struct.calcsize('>IIII')
    imgNum = head[1]  # 图片数
    rows = head[2]  # 宽度
    cols = head[3]  # 高度
```

```
    images = np.empty((imgNum, 784))  # empty，是它所常见的数组内的所有元素均为空，没有实际意义，它是创建数组最快的方法
    image_size = rows * cols  # 单个图片的大小
    fmt = '>' + str(image_size) + 'B'  # 单个图片的 format

    for i in range(imgNum):
        images[i] = np.array(struct.unpack_from(fmt, file_content, offset))
        offset += struct.calcsize(fmt)
    return images
```

```
def read_label(file_name):
    file_handle = open(file_name, "rb")  # 以二进制打开文档
    file_content = file_handle.read()  # 读取到缓冲区中

    head = struct.unpack_from('>II', file_content, 0)  # 取前 2 个整数，返回一个元组
    offset = struct.calcsize('>II')

    labelNum = head[1]  # label 数
    bitsString = '>' + str(labelNum) + 'B'  # fmt 格式：'>47040000B'
    label = struct.unpack_from(bitsString, file_content, offset)  # 取 data 数据，返回一个元组
    return np.array(label)
```

```
def loadDataSet():
    train_x_filename = "train-images-idx3-ubyte"
    train_y_filename = "train-labels-idx1-ubyte"
    test_x_filename = "t10k-images-idx3-ubyte"
    test_y_filename = "t10k-labels-idx1-ubyte"
    train_x = read_image(train_x_filename)
    train_y = read_label(train_y_filename)
    test_x = read_image(test_x_filename)
    test_y = read_label(test_y_filename)

    return train_x, test_x, train_y, test_y
```

2.1.2. 训练模型

```
def train_model(train_x, train_y, theta, learning_rate, numClass):  # theta 是 n+1 行的列向量
    m = train_x.shape[0]
    train_x = np.insert(train_x, 0, values=1, axis=1)

    for k in range(numClass):
        real_y = np.zeros((m, 1))
        index = train_y == k  # index 中存放的是 train_y 中等于 0 的索引
        real_y[index] = 1  # 在 real_y 中修改相应的 index 对应的值为 1，先分类 0 和非 0

        temp_theta = theta[:, k].reshape((785, 1))
        # 求概率
        h_theta = expit(np.dot(train_x, temp_theta)).reshape((60000, 1))
        # 似然函数（取对数版）为了梯度下降而不是上升而取负
        # J_theta[j, k] = (np.dot(np.log(h_theta).T, real_y) + np.dot((1 - real_y).T, np.log(1 - h_theta))) / (-m)
        # 梯度下降
        temp_theta = temp_theta + learning_rate * np.dot(train_x.T, (real_y - h_theta))

        theta[:, k] = temp_theta.reshape((785,))

    error.append(predict(test_x, test_y, theta))

    return theta  # 返回的 theta 是 n*numClass 矩阵
```

2.1.3. 计算错误率

```
def predict(test_x, test_y, theta):  # 这里的 theta 是学习得来的最好的 theta，是 n*numClass 的矩阵
    errorCount = 0
    test_x = np.insert(test_x, 0, values=1, axis=1)
    m = test_x.shape[0]

    h_theta = np.dot(test_x, theta)
    h_theta_max_postion = h_theta.argmax(axis=1)  # 获得每行的最大值的 label
    for i in range(m):
        if test_y[i] != h_theta_max_postion[i]:
            errorCount += 1

    error_rate = float(errorCount) / m
    return error_rate
```

2.1.4. 展示错误率与迭代次数的关系

```
def show(error, iteration):
    x = range(0, iteration, 1)
    plt.xlabel('Number of iterations')
    plt.ylabel('error rate')
    plt.plot(x, error, color='k')
    plt.show()
    plt.savefig('./iteration.png')
```

2.2. 调用 sklearn 实现

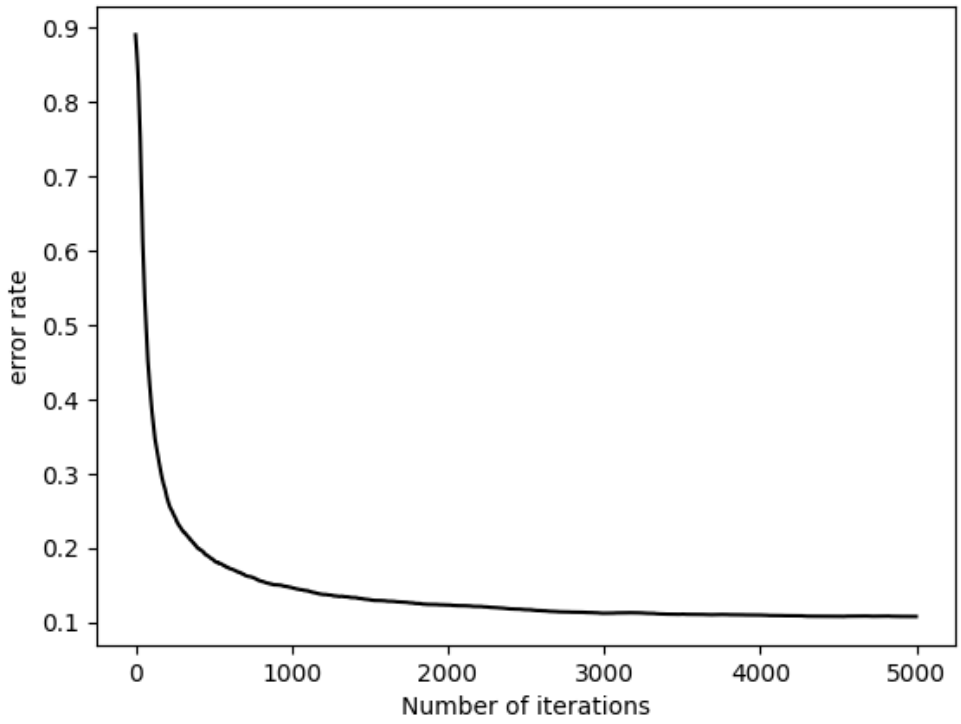
```
def logisticRegression(n):
    # 调用逻辑回归模型
    lr_clf = LogisticRegression(max_iter=n)
    warnings.filterwarnings("ignore")
    # 用逻辑回归模型拟合构造的数据集
    lr_clf = lr_clf.fit(train_x, train_y)  # 其拟合方程为 y=w0+w1*x1+w2*x2
    predict_y = lr_clf.predict(test_x)
    m = test_x.shape[0]
    errorCount = 0
    for i in range(m):
        if test_y[i] != predict_y[i]:
            errorCount += 1

    error_rate = float(errorCount) / m
    return error_rate
```

3. 结果展示

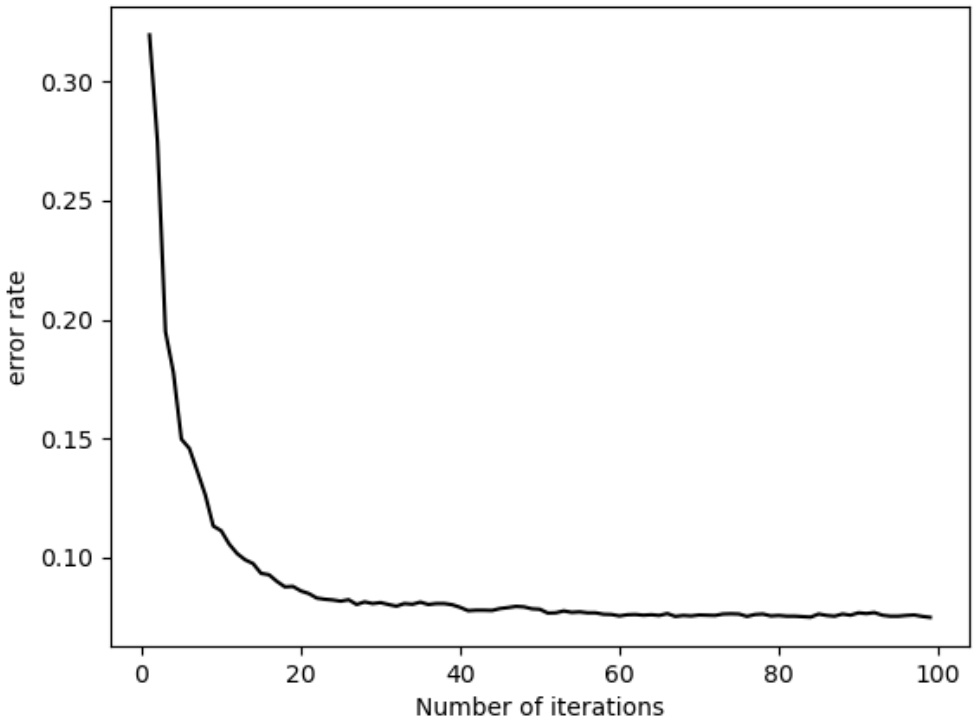
3.1. 从零实现

错误率与迭代次数的关系



3.2. Sklearn 实现

错误率与迭代次数的关系



4. 实验心得

一开始时因为学习率的设置过大，预测错误率随着迭代次数的波动很大，怀疑是自己的梯度下降的学习算法出了问题，调用 sklearn 实现却没有出现此问题，最后将学习率设置为 3×10^{-9}后才解决此问题。理解了逻辑回归分类的算法，加深了对梯度下降的理解。