统计学习与模式识别实验三：Logistic 回归

蔡与望　2020010801024

一　实验原理

1.1　Logistic 回归

在线性回归中，我们使用一个线性函数，预测了连续的输出。但很多时候，我们需要预测的是离散的输出，例如一个手写的数字是“0”还是“1”。在这种分类问题中，Logistic 回归是很好的解决方案。

Logistic 回归的核心是一个假设函数$y = h(x)$，它对于每条已知数据$\left(x^{(i)}, y^{(i)}\right)$，都有$y^{(i)} \approx h\left(x^{(i)}\right)$。如果我们能够找到这样的函数，并且已知的数据足够多，那么即使碰到了新的手写数字，我们也相信这个函数能够预测出它是“0”还是“1”。

为了找到这样的函数，我们先要确定$h(x)$的表达形式。在线性回归中，我们假定它是一个线性函数$\theta^T x$；然而这个函数的值域是整个实数域，显然不适合我们“0”或“1”的离散输出。所以，我们可以在$\theta^T x$外面再套一层函数，令假设函数为：

$$h_\theta(x) = \frac{1}{1 + \exp(-\theta^T x)}$$

如此，整个实数域的输出就被“压缩”到了$(0,1)$。如果$h_\theta(x)$更接近 1，那么手写数字就是“1”；反之，如果$h_\theta(x)$更接近 0，那么手写数字就是“0”。

在刻画$h_\theta\left(x^{(i)}\right)$与$y^{(i)}$的相近程度时，使用下面的损失函数：

$$J(\theta) = -\sum_i \left(y^{(i)} \log h_\theta\left(x^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - h_\theta\left(x^{(i)}\right)\right)\right)^2$$

1.2　梯度下降

现在，目标就转化为找到一个θ，使得$J(\theta)$最小。梯度下降法能够很好的解决这一问题。我们首先计算出$J(\theta)$在当前θ值下的梯度：

$$\nabla_\theta J(\theta) = \begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \frac{\partial J(\theta)}{\partial \theta_2} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{bmatrix}$$

其中，

$$\frac{\partial J(\theta)}{\partial \theta_j} = \sum_i x_j^{(i)}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)$$

由于梯度的方向是函数值增加最快的方向，所以只要沿着梯度的反方向步进，就能够迅速地找到该函数的极小值。也即，每次迭代都有：

$$\theta' = \theta - \alpha \nabla_\theta J(\theta), \alpha > 0$$

其中的α是每次递进的步长。如果步长太长，就有可能在极小值周围“徘徊”；如果步长太短，则迭代次数可能过多。

二 代码实现

加载 MNIST 数据集的函数。在加载图像时，我们需要做一系列预处理：

1. 将 1D 数据集变换为像素数*图像数的 2D 矩阵；
2. 加上一行全为 1 的偏置；
3. 筛选出“0”和“1”的图像；
4. 随机打乱矩阵各列。

```
def load_dataset(kind):
    labels_path = f"{kind}-labels-idx1-ubyte.gz"
    with gzip.open(labels_path, "rb") as fr:
        labels = np.frombuffer(fr.read(), dtype=np.uint8, offset=8)

    images_path = f"{kind}-images-idx3-ubyte.gz"
    with gzip.open(images_path, "rb") as fr:
        images = np.frombuffer(fr.read(), dtype=np.uint8, offset=16) / 255
        images = images.reshape(labels.shape[0], 784).T
        images = np.vstack((np.ones(labels.shape[0]), images))

    filter_indices = np.where((labels == 0) | (labels == 1))[0]
    labels = labels[filter_indices]
    images = images[:, filter_indices]

    shuffle_indices = np.random.permutation(labels.shape[0])
    labels = labels[shuffle_indices]
    images = images[:, shuffle_indices]

    return images, labels
```

手写实现一个 LogisticRegression 类。fit 函数通过最小化损失函数，训练出最佳的θ。predict 函数使用训练好的θ预测新的手写数字是“0”还是“1”。

```
class LogisticRegression:
    def __init__(self, learning_rate=0.1, max_iter=500, record_loss=True):
        self._learning_rate = learning_rate
        self._max_iter = max_iter
        self._record_loss = record_loss
        self._losses = []

    def _logistic(self, z):
        return 1 / (1 + np.exp(-z))

    def _loss(self, h, y):
        return -np.mean(y * np.log(h) + (1 - y) * np.log(1 - h))

    def fit(self, X, y):
        m, n = X.shape
        self.theta = np.random.randn(m, 1) * 0.01

        for iteration in range(self._max_iter):
            z = np.dot(self.theta.T, X)
            h = self._logistic(z)
            gradient = np.dot(X, (h - y).T) / n
            self.theta -= self._learning_rate * gradient

            if self._record_loss and iteration % 100 == 0:
                self._losses.append(self._loss(h, y))
```

```
    def predict(self, X):
        z = np.dot(self.theta.T, X)
        h = self._logistic(z)
        return int(h > 0.5)

    def draw_loss(self):
        plt.plot(np.arange(len(self._losses)) * 100, self._losses)
        plt.xlabel("迭代次数")
        plt.ylabel("损失函数值")
        plt.title("损失函数曲线")
    plt.show()
```

使用训练集训练模型。

```
train_images, train_labels = load_dataset("train")
test_images, test_labels = load_dataset("t10k")

start_time = time()
model = LogisticRegression(learning_rate=0.1, max_iter=500)
model.fit(train_images, train_labels)
end_time = time()
print(f"Optimization took {end_time - start_time:.2f} seconds.")
```

绘制损失函数曲线，并在测试集上测试。

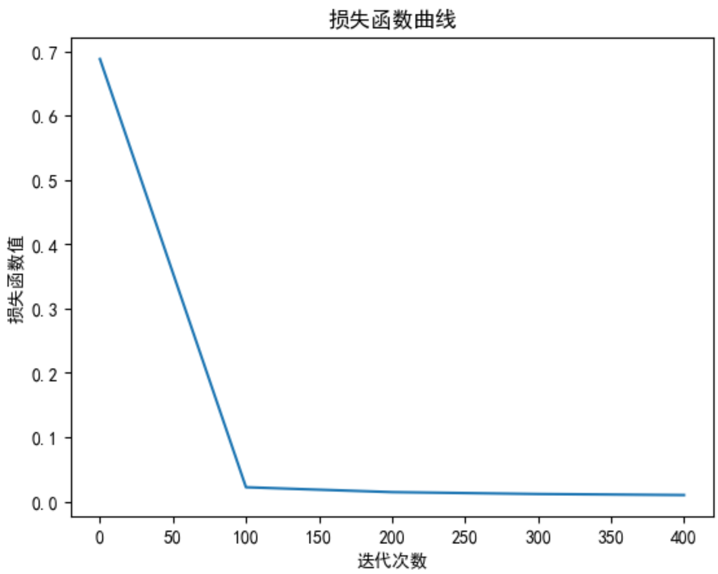
```
model.draw_loss()

train_predictions = [model.predict(image) for image in train_images.T]
train_accuracy = np.mean(train_predictions == train_labels)
print(f"Training accuracy: {round(train_accuracy * 100, 2)}%")

test_predictions = [model.predict(image) for image in test_images.T]
test_accuracy = np.mean(test_predictions == test_labels)
print(f"Test accuracy: {round(test_accuracy * 100, 2)}%")
```

三 结果分析

损失函数曲线如下图所示：



可以看到，经过 100 次迭代训练，模型的损失函数值已经基本接近于 0，并且逐渐趋于平缓。

模型在训练集和测试集上的准确率分别为 99.79%和 99.91%，说明模型训练成功。

四　总结体会

在本实验中，我了解了 Logistic 回归的模型，即它的假设函数和损失函数。

同时，我也通过将其和线性回归对比，对两种回归的原理有了更深的理解。线性回归适合预测连续值，Logistic 回归适合预测离散值（分类）；而产生这一差别的根本原因，是 Logistic 函数将实数域的输出“压缩”到了(0,1)上。尽管有此差别，它们同样能够使用梯度下降法来最小化损失函数。