统计学习与模式识别实验四：Softmax 回归

蔡与望　2020010801024

一　实验原理

1.1　Softmax 回归

Softmax 回归是 Logistic 回归在多分类下的泛化。

在 Logistic 回归下，我们假设标签是二元的：手写数字要么是 0，要么是 1。这是因为 Logistic 回归的假设函数的值域是[0,1]，我们无法从这样的值域中得出更多的信息。然而，现实生活中的多数问题都涉及多分类，Logistic 此时就显得有些无力应对。

因此，Softmax 回归被设计出来解决这一问题。它的核心思想是：在 K 分类下，就用 K 个权重向量分别代表输入在各个维度上的可能性。例如，在手写数字问题中，总共有 10 个数字，即 10 个维度；那么就设置 10 个权重向量，权重向量 0 指示数字 0 的可能性，权重向量 1 指示数字 1 的可能性，……最后，检查哪个权重向量指示的可能性最高，那么输入就最可能是它对应的数字。

1.2　假设函数

在之前的回归中，假设函数的输出都是一个确定的常数。然而，要在多分类下预测，很容易想到，假设函数必须要输出一个 K 维向量，每个元素都代表一个数字的可能性。

$$h_\theta(x)=\begin{bmatrix}P(y=1|x;\theta)\\P(y=2|x;\theta)\\\vdots\\P(y=K|x;\theta)\end{bmatrix}=\frac{1}{\sum_{j=1}^{K}\exp(\theta^{(j)\top}x)}\begin{bmatrix}\exp(\theta^{(1)\top}x)\\\exp(\theta^{(2)\top}x)\\\vdots\\\exp(\theta^{(K)\top}x)\end{bmatrix}$$

在$\theta^T x$外面套一层指数函数，是为了将其值域R映射到R^+；统一除以总和，是为了标准化输出——毕竟所有数字的可能性加起来必须为 1。

1.3　损失函数

损失函数是 Logistic 的损失函数的泛化。

$$J(\theta)=-\left[\sum_{i=1}^{m}\sum_{k=1}^{K}1\left\{y^{(i)}=k\right\}\log\frac{\exp(\theta^{(k)\top}x^{(i)})}{\sum_{j=1}^{K}\exp(\theta^{(j)\top}x^{(i)})}\right]$$

其中，$1\{y^{(i)}=k\}$是一个布尔函数，代表这个数字的真实值是否真的为k。这样，只有与真实值相同的那一部分才会对损失函数做出贡献。

1.4　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：

$$\nabla_{\theta^{(k)}} J(\theta) = -\sum_{i=1}^{m} \left[x^{(i)} \left(1\{y^{(i)} = k\} - P(y^{(i)} = k | x^{(i)}; \theta) \right) \right]$$

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. 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))

    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 SoftmaxRegression 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哪个数字。

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

    def _softmax(self, z):
        exp = np.exp(z)
        return exp / np.sum(exp, axis=0)

    def _loss(self, h, y):
        return -np.mean(np.sum(y * np.log(h), axis=0))

    @property
    def _record_loss_interval(self):
        return max(1, self._max_iter // 10)

    def fit(self, X, y):
        m, n = X.shape
```

```
        k = len(np.unique(y))
        self._theta = np.random.randn(m, k) * 0.01

        one_hot = np.zeros((k, n))
        one_hot[y, np.arange(n)] = 1

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

            if (iteration + 1) % self._record_loss_interval == 0:
                self._losses.append(self._loss(h, one_hot))

    def predict(self, X):
        z = np.dot(self._theta.T, X)
        h = self._softmax(z)
        return np.argmax(h)

    def draw_loss(self):
        plt.plot(
            np.arange(1, len(self._losses) + 1) * self._record_loss_interval,
            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()
```

训练模型，绘制损失函数曲线。

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

elapsed_time = -time()
model = SoftmaxRegression()
model.fit(train_images, train_labels)
elapsed_time += time()

print(f"Optimization took {elapsed_time:.2f} seconds.")
model.draw_loss()
```

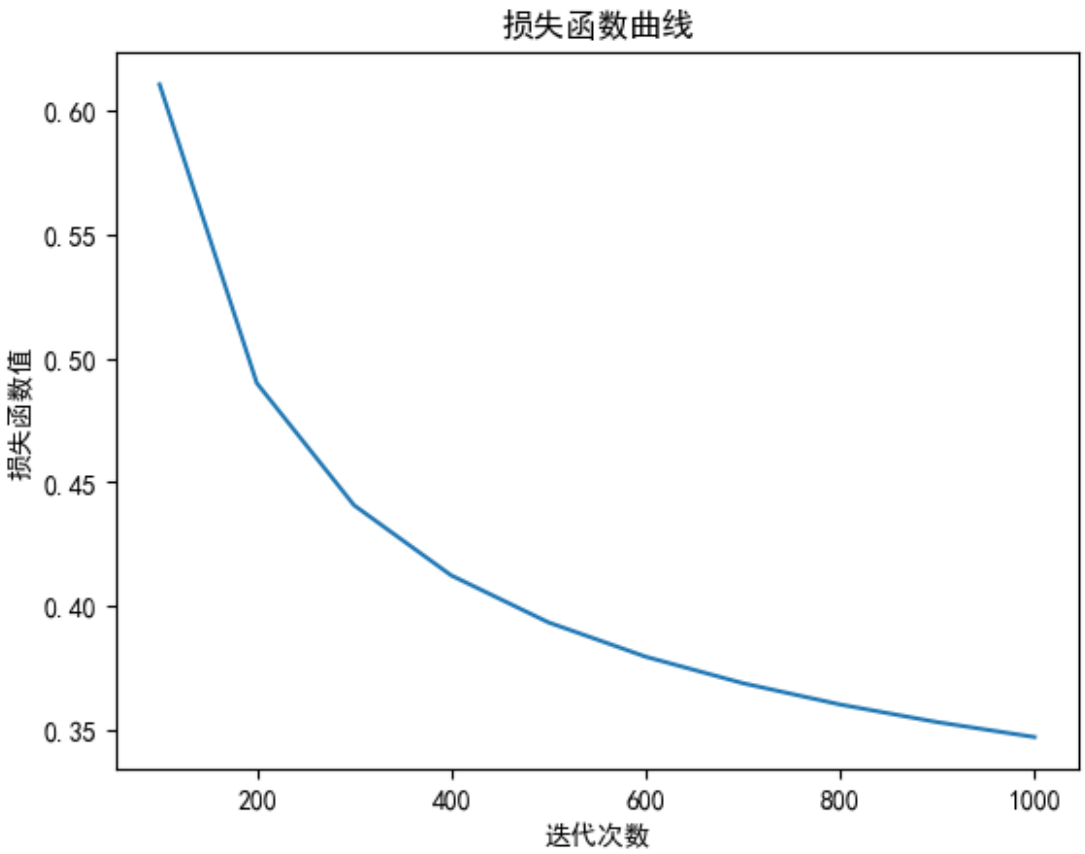
在训练集和测试集上分别测试准确率。

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

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

三 结果分析

损失函数曲线如下图所示：



可以看到，在学习率 0.1 时，经过 1000 次迭代训练，模型的损失函数曲线逐渐趋于平缓。

模型在训练集和测试集上的准确率分别为 90.42%和 90.93%，说明模型训练成功。

```
Training accuracy: 90.42%
Test accuracy: 90.93%
```

四 总结体会

在本实验中，我了解了 Softmax 回归的模型，即它的假设函数、损失函数。通过和 Logistic 回归的对比，我理解了“Softmax 是 Logistic 的泛化”的含义，也因此明白了 Softmax 的一些底层原理，比如为什么 Softmax 回归的权重矩阵要这样设置、它的假设函数要这样计算，等等。

同时，训练的结果其实还没有达到最优，但由于个人笔记本性能原因，没有继续训练下去。从损失函数曲线可以看到，实际上损失值离收敛还有一段距离；如果增加迭代次数，训练的效果会更好，准确率会提高——但提高得有限，因为损失函数已经低于 0.5 了，准确率不会有太大的提升。