Logistic 回归算法

陈 鑫

2023 年 3 月 29 日

目录

Logistic 回归算法

Logistic 回归算法是模型假设为 Sigmoid 函数，损失函数为对数损失函数的经验损失最小化算法。

Logistic 回归虽然名字中有“回归”，但却是一个分类算法，可以处理二元分类以及多元分类。算法的任务是对给定的特征 $\boldsymbol{x}$ 预测对象属于每一个类别的概率。

由于预测的对象是一个概率，所以线性回归模型就不适用了（因为预测结果可能超出 $[0,1]$ 区间）。为了能够根据特征 $\boldsymbol{x}$ 来预测概率，在线性回归的基础上，寻找一个连续函数，它既能表达特征 $\boldsymbol{x}$ 与概率之间的依存关系，又能保证在特征变动时所对应的函数值不超出 $[0,1]$ 区间，Sigmoid 函数就是满足这种要求的函数。

Logistic 回归通过对概率值设定一个阈值，将回归问题转化为一个分类问题，以阈值划分区间，落到不同范围则分成不同的类别。

Logistic 回归算法

定义

Sigmoid 函数的表达式为 $s(t) = \frac{1}{1+e^{-t}}$，$t \in \mathbb{R}$，当 $t \to -\infty$ 时，$s(t) \to 0$，当 $t \to +\infty$ 时，$s(t) \to 1$。

Sigmoid 函数的取值都在区间 $[0, 1]$ 中。其函数图像如图所示。它有一个很好的导数性质：$s'(t) = s(t)(1 - s(t))$。

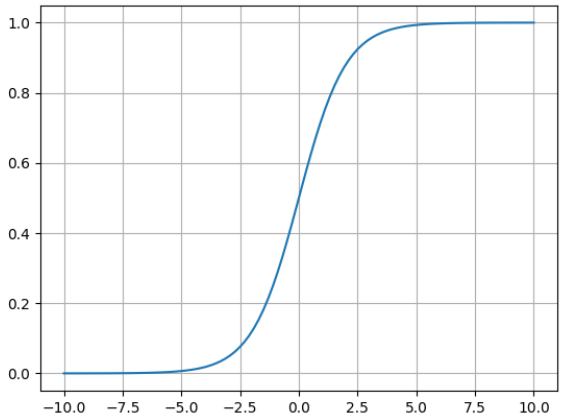

图 1: sigmoid 函数

Logistic 回归算法

用 Sigmoid 函数描述样本特征 $\boldsymbol{x}$ 与标签 y 的概率 P 之间的关系的模型就是 Logistic 模型。将 $t = \boldsymbol{x}^T\boldsymbol{w}$ 代入 Sigmoid 函数，就得到了 Logistic 模型。

定义

对于样本特征 $\boldsymbol{x} \in \mathbb{R}^n$，称模型 $h_{\boldsymbol{w}}(\boldsymbol{x}) = s(\boldsymbol{x}^T\boldsymbol{w}) = \frac{1}{1+e^{-x^T w}}$ 为一个 Logistic 模型，其中 $\boldsymbol{w} \in \mathbb{R}^n$ 为模型的参数。

给定特征 $\boldsymbol{x}$ 和标签 y，采用 Logistic 模型 $h_{\boldsymbol{w}}(\boldsymbol{x})$ 描述特征 $\boldsymbol{x}$ 与概率 P 之间的关系，其中，$h_{\boldsymbol{w}}(\boldsymbol{x}) = \frac{1}{1+e^{-x^T w}}$ 预测 $\boldsymbol{x}$ 的标签为 1 的概率即 $P(y=1)$，而 $1 - h_{\boldsymbol{w}}(\boldsymbol{x}) = \frac{1}{1+e^{x^T w}}$ 预测标签为 0 的概率 $P(y=0)$。把 $P(y=1)$ 和 $P(y=0)$ 的式子结合起来，统一写成：

$$P(y) = h_{\boldsymbol{w}}(\boldsymbol{x})^y (1 - h_{\boldsymbol{w}}(\boldsymbol{x}))^{1-y}$$

为一个样本属于其真实标签的概率分布表达式。

Logistic 回归算法

定义

对于任意 $\boldsymbol{y} = (y_1, y_2, \cdots, y_k)$，$\boldsymbol{z} = (z_1, z_2, \cdots, z_k) \in [0,1]^k$，*对数损失函数定义为*

$$\ell(\boldsymbol{y}, \boldsymbol{z}) = -\sum_{t=1}^{k} y_t \log z_t$$

在对数损失函数中，标签采用的是 k 元分类问题的向量标签形式，对于二元分类问题，如果标签采取 $0-1$ 标签的形式，则对数损失函数可以表示为：

$$\ell(y, h_{\boldsymbol{w}}(\boldsymbol{x})) = -y \log(h_{\boldsymbol{w}}(\boldsymbol{x})) - (1-y)(\log(1 - h_w(\boldsymbol{x})) \tag{1}$$

给定一条训练数据 $(\boldsymbol{x}, y)$，$\boldsymbol{x} \in \mathbb{R}^n$，$y \in \{0,1\}$。$h_{\boldsymbol{w}}$ 为 Logistic 模型，那么模型 $h_{\boldsymbol{w}}$ 在 $(\boldsymbol{x}, y)$ 上的对数损失函数为

$$\ell(y, h_{\boldsymbol{w}}(\boldsymbol{x})) = y \log(1 + e^{-\boldsymbol{x}^T \boldsymbol{w}}) + (1-y) \log(1 + e^{\boldsymbol{x}^T \boldsymbol{w}}) \tag{2}$$

Logistic 回归的目标函数是由上式 (2) 的损失函数在训练数据上的经验损失。因此其回归算法如下：

Algorithm 1: Logistic 回归算法

Input: m 个训练数据 $S = \{(\boldsymbol{x}^{(1)}, y^{(1)}), (\boldsymbol{x}^{(2)}, y^{(2)}), \cdots, (\boldsymbol{x}^{(m)}, y^{(m)})\}$

Output: Logistic 模型 $h_{\boldsymbol{w}}(\boldsymbol{x}) = \frac{1}{1+e^{-\boldsymbol{x}^T \boldsymbol{w}}}$，使得 $\boldsymbol{w}^* \in \mathbb{R}^n$ 为如下优化问题的最优解:

$$\min_{\boldsymbol{w} \in \mathbb{R}^n} \frac{1}{m} \sum_{i=1}^{m} \{y^{(i)} \log(1 + e^{-\boldsymbol{x}^{(i)T} \boldsymbol{w}}) + (1 - y^{(i)}) \log(1 + e^{\boldsymbol{x}^{(i)T} \boldsymbol{w}})\}$$

Logistic 回归算法

定义

Logistic 回归问题的目标函数称为交叉熵

$$f(\boldsymbol{w}) = \frac{1}{m}\sum_{i=1}^{m}\left\{y^{(i)}\log(1+e^{-\boldsymbol{x}^{(i)T}\boldsymbol{w}}) + (1-y^{(i)})\log(1+e^{\boldsymbol{x}^{(i)T}\boldsymbol{w}})\right\}$$

因此，Logistic 的目标就是最小化交叉熵。可以使用梯度下降算法来求解。

Algorithm 2: 函数 $f(\boldsymbol{w})$ 的梯度下降算法

求函数 $f(\boldsymbol{w})$ 的极小值点 $\boldsymbol{w}$
Input: $\boldsymbol{w} = 0$
Output: $\boldsymbol{w}$
for $i = 1, 2, \cdots, N$: **do**
  $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta\nabla f(\boldsymbol{w})$
return $\boldsymbol{w}$

Logistic 回归算法

模型 $h_{\boldsymbol{w}}$ 在样本 $(\boldsymbol{x}, y)$ 上的对数损失函数为

$$\ell(y, h_{\boldsymbol{w}}(\boldsymbol{x})) = y \log(1 + e^{-\boldsymbol{x}^T \boldsymbol{w}}) + (1 - y) \log(1 + e^{\boldsymbol{x}^T \boldsymbol{w}})$$

该函数的梯度为

$$\begin{aligned} \nabla \ell\left(y, h_{\boldsymbol{w}}(\boldsymbol{x})\right) &= -y\boldsymbol{x} \frac{e^{-\boldsymbol{x}^T w}}{1 + e^{-\boldsymbol{x}^T \boldsymbol{w}}} + (1 - y)\boldsymbol{x} \frac{e^{\boldsymbol{x}^T \boldsymbol{w}}}{1 + e^{\boldsymbol{x}^T \boldsymbol{w}}} \\ &= \boldsymbol{x}\left(\frac{1}{1 + e^{-\boldsymbol{x}^T \boldsymbol{w}}} - y\right) = \boldsymbol{x}(h_{\boldsymbol{w}}(\boldsymbol{x}) - y) \end{aligned}$$

根据梯度的线性特性，交叉熵 $f(\boldsymbol{w})$ 的梯度为

$$\nabla f(\boldsymbol{w}) = \frac{1}{m} \sum_{i=1}^{m} \boldsymbol{x}^{(i)} \left(h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) - y^{(i)}\right)$$

Logistic 回归算法

为了计算方便，将其表示为矩阵的形式

$$\nabla f(\boldsymbol{w}) = \frac{1}{m} X^T (h_{\boldsymbol{w}}(X) - y)$$

其中，

$$X = \begin{pmatrix} \boldsymbol{x}^{(1)T} \\ \boldsymbol{x}^{(2)T} \\ \vdots \\ \boldsymbol{x}^{(m)T} \end{pmatrix}, \boldsymbol{y} = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{pmatrix} h_{\boldsymbol{w}}(X) = \begin{pmatrix} h_{\boldsymbol{w}}(\boldsymbol{x}^{(1)}) \\ h_{\boldsymbol{w}}(\boldsymbol{x}^{(2)}) \\ \vdots \\ h_{\boldsymbol{w}}(\boldsymbol{x}^{(m)}) \end{pmatrix}$$

分别是 $m \times n$ 的矩阵和两个 $m \times 1$ 的向量。

Logistic 回归算法

有了交叉熵的梯度表达式之后，代入算法2，得到下面的算法，其代码来实现可参考 logistic_regression_gd.py 文件。

Algorithm 3: 交叉熵的梯度下降算法

求交叉熵函数 $f(\boldsymbol{w})$ 的极小值点 $\boldsymbol{w}$

Input: $\boldsymbol{w} = 0$

Output: $\boldsymbol{w}$

for $i = 1, 2, \cdots, N$: **do**

$\quad \boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \frac{1}{m} X^T (h_{\boldsymbol{w}}(X) - y)$

return $\boldsymbol{w}$

案例 5.1 手写数字识别问题

在 sklearn 中，包含有 1797 个 $0-9$ 的数字，每个数字都是一个 8×8 的手写数字缩略图。每个数字由 64 个像素的构成，可以将每个数字看成是一个 64 维空间中的点，每个维度表示一个像素的亮度。

具体代码参考 digital_recognition.py

交叉熵的统计意义

交叉熵是一种用来衡量两个概率分布之间差异的度量方法。它可以用于评估一个概率分布是否能够很好地描述另一个概率分布。在机器学习中，交叉熵常常被用来衡量模型的预测结果和真实结果之间的差异。

统计意义上，交叉熵可以被看作是两个概率分布之间的“距离”。具体来说，如果我们有两个离散的概率分布 p 和 q，它们的取值范围分别是 $x_1, x_2, \ldots, x_n$，那么它们之间的交叉熵可以表示为：

$$H(p,q) = -\sum_{i=1}^{n} p(x_i)\log(x_i)$$

其中，$p(x_i)$ 表示真实分布中取值为 x_i 的概率，$q(x_i)$ 表示模型预测的分布中取值为 x_i 的概率，log 是自然对数。

交叉熵的统计意义在于，它衡量的是真实分布和预测分布之间的不一致性。如果两个分布非常接近，交叉熵会很小；如果两个分布差异很大，交叉熵会很大。因此，交叉熵可以用来评估模型的预测能力，即模型能否将真实分布很好地拟合。在训练模型时，我们通常会使用交叉熵作为损失函数，通过最小化交叉熵来优化模型的预测结果。

似然函数与交叉熵的关系

似然函数：在统计学中，概率描述了已知参数时的随机变量的输出结果，似然则用来描述已知随机变量输出结果时，未知参数的可能取值。似然函数就是用来求得未知参数的估计值所使用的函数。

极大似然估计：通过最大化似然函数求得未知参数的估计值。若将参数看成固定的，则极大似然函数是概率密度函数，其取得极大值，就是最有可能出现的情况；反过来说，能使似然函数取得极大值的这组参数，是最有可能的参数，因此使用极大似然作为估计。

似然函数的目标：令每个样本属于其真实标签的概率越大越好，即极大似然法。

Logistic 回归算法关于标签分布的基本假设为伯努利分布。给定 Logistic 模型 h_w 及特征样本 $\boldsymbol{x}$，Logistic 回归假设标签分布是以 $h_w(\boldsymbol{x})$ 为期望的伯努利分布。如果用 Y 表示标签随机变量，则 Y 的分布有如下参数化形式：

$$p_w(Y=y)=\begin{cases}h_w(\boldsymbol{w}), & \text{如果} y=1\\ 1-h_w(\boldsymbol{w}), & \text{如果} y=0\end{cases}$$

等价于：

$$P(Y=y)=h_w(\boldsymbol{x})^y(1-h_w(\boldsymbol{x}))^{1-y}$$

给定训练数据 $S = \{(\boldsymbol{x}^{(1)}, y^{(1)}), (\boldsymbol{x}^{(2)}, y^{(2)}), \cdots, (\boldsymbol{x}^{(m)}, y^{(m)})\}$，Logistic 模型 $h_{\boldsymbol{w}}$ 的似然函数为

$$L(\boldsymbol{w}) = \prod_{i=1}^{m} P(y^{(i)} | \boldsymbol{x}^{(i)}, \boldsymbol{w}) = \prod_{i=1}^{m} h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)})^{y^{(i)}} \left(1 - h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)})\right)^{1-y^{(i)}}$$

$L(\boldsymbol{w})$ 越大越好，即求 $L(\boldsymbol{w})$ 的极大值，作为优化目标，求解对应的参数，即使用“极大似然估计”。由于似然函数取对数不改变其极值点且取对数之后，表达式更为简洁易求，所以将之转化为对数似然表达式，然后再两边取负号，可得到负对数似然表达式，那么求极大值就可转化为求极小值。于是得到

$$-\log L(\boldsymbol{w}) = -\sum_{i=1}^{m} \left(y^{(i)} \log h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) + (1 - y^{(i)}) \log \left(1 - h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)})\right)\right)$$

在乘以常数 $\frac{1}{m}$，即得交叉熵。这就是交叉熵与似然函数的关系。

将上述公式写成矩阵形式，其中 $\mathbf{1}$ 为全 1 向量，$h_{\boldsymbol{w}}(X) = \frac{1}{1+e^{-X\boldsymbol{w}}}$

$$-\log L(\boldsymbol{w}) = -\boldsymbol{y}^T \log h_{\boldsymbol{w}}(X) - (\mathbf{1} - \boldsymbol{y})^T \log \left(\mathbf{1} - h_{\boldsymbol{w}}(X)\right)$$

Logistic 回归的牛顿迭代算法

目标：求 Logistic 回归问题的目标函数为交叉熵

$$f(\boldsymbol{w}) = \frac{1}{m}\sum_{i=1}^{m}\left\{y^{(i)}\log(1+e^{-\boldsymbol{x}^{(i)T}\boldsymbol{w}}) + (1-y^{(i)})\log(1+e^{\boldsymbol{x}^{(i)T}\boldsymbol{w}})\right\}$$

给定一条训练数据 $(\boldsymbol{x}, y)$，$\boldsymbol{x} \in \mathbb{R}^n$，$y \in \{0,1\}$。$h_{\boldsymbol{w}}$ 为 Logistic 模型，那么模型 $h_{\boldsymbol{w}}$ 在 $(\boldsymbol{x}, y)$ 上的对数损失函数为

$$\ell(y, h_{\boldsymbol{w}}(\boldsymbol{x})) = y\log(1+e^{-\boldsymbol{x}^T\boldsymbol{w}}) + (1-y)\log(1+e^{\boldsymbol{x}^T\boldsymbol{w}}) \tag{3}$$

其梯度为

$$\nabla\ell(y, h_{\boldsymbol{w}}(\boldsymbol{x})) = -y\boldsymbol{x}\frac{e^{-\boldsymbol{x}^T w}}{1+e^{-\boldsymbol{x}^T\boldsymbol{w}}} + (1-y)\boldsymbol{x}\frac{e^{\boldsymbol{x}^T\boldsymbol{w}}}{1+e^{\boldsymbol{x}^T\boldsymbol{w}}} \tag{4}$$

$$= \boldsymbol{x}\left(\frac{1}{1+e^{-\boldsymbol{x}^T\boldsymbol{w}}} - y\right) = \boldsymbol{x}(h_{\boldsymbol{w}}(\boldsymbol{x}) - y) \tag{5}$$

进而可以求得其对参数 $\boldsymbol{w}$ 的二阶梯度为

$$\nabla^2\ell(y, h_{\boldsymbol{w}}(\boldsymbol{x})) = \boldsymbol{x}\frac{e^{-\boldsymbol{x}^T\boldsymbol{w}}}{(1+e^{-\boldsymbol{x}^T\boldsymbol{w}})^2}\boldsymbol{x}^T = \boldsymbol{x}\frac{1}{1+e^{-\boldsymbol{x}^T\boldsymbol{w}}}\cdot\frac{1+e^{-\boldsymbol{x}^T\boldsymbol{w}}-1}{1+e^{-\boldsymbol{x}^T\boldsymbol{w}}}\boldsymbol{x}^T \tag{6}$$

$$= \boldsymbol{x}h_{\boldsymbol{w}}(\boldsymbol{x})(1-h_{\boldsymbol{w}}(\boldsymbol{x}))\boldsymbol{x}^T \tag{7}$$

Logistic 回归的牛顿迭代算法

根据海森矩阵的线性特性，可求得

$$\nabla^2 f(\boldsymbol{w}) = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{x}^{(i)} h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)})(1 - h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}))\boldsymbol{x}^{(i)T} \tag{8}$$

使用矩阵记号表示，可将上式的海森矩阵表示为

$$\nabla^2 f(\boldsymbol{w}) = \frac{1}{m} X^T \Lambda X \tag{9}$$

其中，Λ 是一个 $m \times m$ 的对角矩阵，它的第 i 个对角元素为 $\lambda^{(i)} = h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)})(1 - h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}))$，即：

$$\nabla^2 f(\boldsymbol{w}) = \frac{1}{m} X^T \Lambda X \tag{10}$$

$$= \frac{1}{m}(\boldsymbol{x}^{(1)}\boldsymbol{x}^{(2)}\cdots\boldsymbol{x}^{(m)}) \begin{pmatrix} \lambda^{(1)} & & & \\ & \lambda^{(2)} & & \\ & & \ddots & \\ & & & \lambda^{(m)} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}^{(1)T} \\ \boldsymbol{x}^{(2)T} \\ \vdots \\ \boldsymbol{x}^{(m)T} \end{pmatrix} \tag{11}$$

Logistic 回归的牛顿迭代算法

将 $\nabla f(\boldsymbol{w}) = \frac{1}{m} X^T (h_{\boldsymbol{w}}(X) - y)$，$\nabla^2 f(\boldsymbol{w}) = \frac{1}{m} X^T \Lambda X$ 代入多元函数优化问题的牛顿迭代算法4

Algorithm 4: 多元函数优化问题的牛顿迭代算法

$\boldsymbol{w} = 0$

for $i = 0$ **to** $N - 1$ **do**

$\quad \boldsymbol{w} \leftarrow \boldsymbol{w} - \nabla^2 f(\boldsymbol{w})^{-1} \nabla f(\boldsymbol{w})$

Return $\boldsymbol{w}$

得到

Algorithm 5: 多元函数优化问题的牛顿迭代算法

$\boldsymbol{w} = 0$

for $i = 0$ **to** $N - 1$ **do**

$\quad \boldsymbol{w} \leftarrow \boldsymbol{w} - (X^T \Lambda X)^{-1} X^T (h_{\boldsymbol{w}}(X) - \boldsymbol{y})$

Return $\boldsymbol{w}$

有了这个算法之后，我们就可以通过代码实现 Logistic 回归，具体参考 logistic_regression_nt.py。其中 fit 函数就是使用牛顿迭代去更新参数 w，predict_proba 实现对未知样本属于哪一个类别的概率预测，再由 predict 函数设置的阈值将其转换为某一类别。

Logistic 回归的牛顿迭代算法

Python 实现的代码 logistic_regression_nt.py：

```
import numpy as np
def sigmoid(x):
    return 0.5 * (1 + np.tanh(0.5 * x))
class LogisticRegression:
    def __init__(self):
        self.w = None
    def fit(self, X, y, N=1000):
        m, n = X.shape
        w = np.zeros((n, 1))
        for t in range(N):
            pred = sigmoid(X.dot(w)) # h_w(x)
            g = 1.0 / m * X.T.dot(pred - y) # F(w)的梯度
            pred = pred.reshape(-1)
            # h_w(x)(1-h_w(x))再将这些值作为矩阵的对角元素
            D = np.diag(pred * (1 - pred))
            #F(w)的Hessian方阵
            H = 1.0 / m * (X.T.dot(D)).dot(X)
            # w=w-F'(w)/F''(w) 多元牛顿迭代
            w = w - np.linalg.inv(H).dot(g)
        self.w = w
    def predict_proba(self, X):
        return sigmoid(X.dot(self.w))
    def predict(self, X):
        proba = self.predict_proba(X)
        return (proba >= 0.5).astype(np.int)
```

定义

示性函数 $I\{\}$：$\{True, False\} \to \{0, 1\}$ *定义为* $I\{True\} = 1, I\{False\} = 0$

则准确率可以定义为

定义

（准确率）在二元分类问题中，给定一组数据：
$T = \{(\boldsymbol{x}^{(1)}, y^{(1)}), (\boldsymbol{x}^{(2)}, y^{(2)}), \cdots, (\boldsymbol{x}^{(m)}, y^{(m)})\}$ *模型* h *在数据* T *上的准确率定义为*

$$Acc_T(h) = \frac{\sum_{i=1}^{m} I\{h(x^{(i)}) = y^{(i)}\}}{m}$$

准确率适合于类别预测任务的算法。是最常见的评价指标，通常来说，正确率越高，分类器越好。

在一个二元分类问题中，给定样本的特征 $\boldsymbol{x}$ 及其标签 $y \in \{0, 1\}$。将标签为 1 的样本称为正采样，标签为 0 的样本称为负采样。设 $h(\boldsymbol{x}) \in \{0, 1\}$ 为模型 h 对样本的标签预测。将 $h(\boldsymbol{x}) = 1$ 称为正预测，$h(\boldsymbol{x}) = 0$ 称为负预测。那么用 TP：预测为 1，预测正确，即实际 1；FP：预测为 1，预测错误，即实际 0；FN：预测为 0，预测错误，即实际 1；TN：预测为 0，预测正确即，实际 0。则有表1，其中 T 表示 True，F 表示 False，P 表示 Positive，N 表示 negative。

表 1: 混淆矩阵

	$y = 1$	$y = 0$
$h(\boldsymbol{x}) = 1$	真正 TP	假正 FP
$h(\boldsymbol{x}) = 0$	假负 FN	真负 TN

定义

在一个二元分类问题中，给定一组数据： $T = \{(\boldsymbol{x}^{(1)}, y^{(1)}), (\boldsymbol{x}^{(2)}, y^{(2)}), \cdots, (\boldsymbol{x}^{(m)}, y^{(m)})\}$，*用 TP、FP、TN、FN 分别表示模型* h *在数据集* T *中的真正、假正、真负、假负的预测个数，则模型* h *在数据集* T *上的精确率* $Pre_T(h)$ *和召回率* $Rec_T(h)$ *分别为*

$$Pre_T(h) = \frac{TP}{TP + FP}$$

$$Rec_T(h) = \frac{TP}{TP + FN}$$

即精确率的分子为真正的预测个数，分母是真正的预测个数和假正的预测个数之和。精确率是在预测标签为 1 的样本中，实际标签为 1 的比例。

召回率的分子是真正的预测个数，分母是真正的预测个数和假负的预测个数之和，即数据集 T 中正采样（$y = 1$）的总个数。召回率是正采样中，能被模型甄别出的比例。

案例实战 5.2：假设有 8 条标签属于 $\{0,1\}$ 的数据。他们的标签组成的向量为 $(1,1,1,1,0,0,0,0)$。假设模型 h 对这 8 条数据的标签预测为 $(1,0,1,1,0,1,1,0)$。请计算模型 h 的准确率、精确率和召回率。

案例实战 5.2：假设有 8 条标签属于 $\{0,1\}$ 的数据。他们的标签组成的向量为 $(1,1,1,1,0,0,0,0)$。假设模型 h 对这 8 条数据的标签预测为 $(1,0,1,1,0,1,1,0)$。请计算模型 h 的准确率、精确率和召回率。

表 2: 案例的混淆矩阵

	$y=1$	$y=0$
$h(x)=1$	真正 $TP=3$	假正 $FP=2$
$h(x)=0$	假负 $FN=1$	真负 $TN=2$

所以，准确率为 $Acc_T(h)=(3+2)/8=5/8$；精确率 $Pre_T(h)=3/(2+3)=3/5$；召回率为 $Rec_T(h)=3/(3+1)=3/4$。

实现交叉熵、准确率、精准率以及召回率的文件参考 metrics.py。利用前面实现的 Logistic 类和度量指标，再次使用算法进行数字 0 和 1 的二分类识别。代码参考 dig_recog10_gd.py

一般情况下，算法的精确率和召回率是不可兼得的。在要求两者都高的条件下，通常选择它们的调和平均值 F_1 作为度量的指标。

$$F_1(h) = \frac{2}{\frac{1}{Pre(h)} + \frac{1}{Rec(h)}}$$

F_1 作为指标能较好地平衡精确率和召回率的影响，是一个比较全面的评价指标。但该指标在使用时，也应当注意一些场景。在分类问题的假正预测有时会带来严重后果，例如对飞机零部件合格性的预测，要以精确率作为度量标准，否则，将不合格零件预测为合格零件，将可能带来灾难。在分类问题的假负预测有时会带来严重后果，例如对不及时治疗有较差预后的疾病做早期筛查，要以召回率作为度量标准，否则，将疾病患者预测为健康可能会错过患者的最佳治疗时机，造成严重的后果。

案例实战 5.3：山鸢尾识别问题。

Sklearn 数据库中的鸢尾花数据集每一条数据表示一株鸢尾花，包含花萼长、花萼宽、花瓣长以及花瓣宽 4 个特征，且每一条数据都对应有一个标签，表示不同的种属，通过这 4 个特征预测鸢尾花卉属于哪一个品种。代码：iris_recog_gd.py

案例实战 5.4：mnist 数据集数字识别。

MNIST（Mixed National Institute of Standards and Technology database）数据集是美国国家标准与技术研究院收集整理的大型手写数字数据库，包含 $60,000$ 个示例的训练集以及 $10,000$ 个示例的测试集，每个样本都是一张 28×28 像素的灰度手写数字图片。代码：mnist_recog_gd.py

给定一条训练数据 $(\boldsymbol{x}, y)$，$\boldsymbol{x} \in \mathbb{R}^n$，$y \in \{0, 1\}$。$h_{\boldsymbol{w}}$ 为 Logistic 模型，那么模型 $h_{\boldsymbol{w}}$ 在 $(\boldsymbol{x}, y)$ 上的对数损失函数为

$$\ell(y, h_{\boldsymbol{w}}(\boldsymbol{x})) = y \log(1 + e^{-\boldsymbol{x}^T \boldsymbol{w}}) + (1 - y) \log(1 + e^{\boldsymbol{x}^T \boldsymbol{w}})$$

其梯度为

$$\begin{aligned} \nabla \ell\left(y, h_{\boldsymbol{w}}(\boldsymbol{x})\right) &= -y\boldsymbol{x} \frac{e^{-\boldsymbol{x}^T w}}{1 + e^{-\boldsymbol{x}^T \boldsymbol{w}}} + (1 - y)\boldsymbol{x} \frac{e^{\boldsymbol{x}^T \boldsymbol{w}}}{1 + e^{\boldsymbol{x}^T \boldsymbol{w}}} \\ &= \boldsymbol{x} \left(\frac{1}{1 + e^{-\boldsymbol{x}^T \boldsymbol{w}}} - y \right) = \boldsymbol{x}(h_{\boldsymbol{w}}(\boldsymbol{x}) - y) \end{aligned}$$

进而可以求得其对参数 $\boldsymbol{w}$ 的二阶梯度为

$$\nabla^2 \ell(y, h_{\boldsymbol{w}}(\boldsymbol{x})) = \boldsymbol{x} \frac{e^{-\boldsymbol{x}^T \boldsymbol{w}}}{1 + e^{-\boldsymbol{x}^T \boldsymbol{w}}} \boldsymbol{x}^T = \boldsymbol{x} h_{\boldsymbol{w}}(\boldsymbol{x})(1 - h_{\boldsymbol{w}}(\boldsymbol{x})) \boldsymbol{x}^T$$

根据海森矩阵的线性特性，可求得

$$\nabla^2 f(\boldsymbol{w}) = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{x}^{(i)} h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)})(1 - h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}))\boldsymbol{x}^{(i)T}$$

使用矩阵记号表示，可将上式的海森矩阵表示为

$$\nabla^2 f(\boldsymbol{w}) = \frac{1}{m} X^T \Lambda X$$

其中，Λ 是一个 $m \times m$ 的对角矩阵，它的第 i 个对角元素为 $\lambda^{(i)} = h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)})(1 - h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}))$，即：

$$\begin{aligned}
\nabla^2 f(\boldsymbol{w}) &= \frac{1}{m} X^T \Lambda X \\
&= \frac{1}{m} (\boldsymbol{x}^{(1)} \boldsymbol{x}^{(2)} \cdots \boldsymbol{x}^{(m)}) \begin{pmatrix} \lambda^{(1)} & & & \\ & \lambda^{(2)} & & \\ & & \ddots & \\ & & & \lambda^{(m)} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}^{(1)T} \\ \boldsymbol{x}^{(2)T} \\ \vdots \\ \boldsymbol{x}^{(m)T} \end{pmatrix}
\end{aligned}$$

Logistic 回归的牛顿迭代算法

前面 Logistic 回归的类中的参数拟合 fit 函数，采用的是梯度下降算法，求得了海森矩阵之后，fit 函数也可以使用牛顿迭代去更新参数 w。具体参考 logistic_regression_nt.py。

```
import numpy as np
def sigmoid(x):
    return 0.5 * (1 + np.tanh(0.5 * x))
class LogisticRegression:
    def __init__(self):
        self.w = None
    def fit(self, X, y, N=1000):
        m, n = X.shape
        w = np.zeros((n, 1))
        for t in range(N):
            pred = sigmoid(X.dot(w)) # h_w(x)
            g = 1.0 / m * X.T.dot(pred - y) # F(w)的梯度
            pred = pred.reshape(-1)
            # h_w(x)(1-h_w(x))再将这些值作为矩阵的对角元素
            D = np.diag(pred * (1 - pred))
            #F(w)的Hessian方阵
            H = 1.0 / m * (X.T.dot(D)).dot(X)
            # w=w-F'(w)/F''(w) 多元牛顿迭代
            w = w - np.linalg.inv(H).dot(g)
        self.w = w
    def predict_proba(self, X):
        return sigmoid(X.dot(self.w))
    def predict(self, X):
        proba = self.predict_proba(X)
        return (proba >= 0.5).astype(np.int)
```

案例实战 5.5：变色鸢尾花的识别。具体代码参见 iris_recog_nt.py

ROC 曲线和 AUC 度量

ROC 曲线中的主要两个指标就是真正率和假正率。其中横坐标为假正率（FPR），纵坐标为真正率（TPR），下面给出它们的定义。在一个二元分类问题中，给定一组数据：$T = \{(\boldsymbol{x}^{(1)}, y^{(1)}), (\boldsymbol{x}^{(2)}, y^{(2)}), \cdots, (\boldsymbol{x}^{(m)}, y^{(m)})\}$，并给定概率预测模型 h。用 TP(t)、FP(t)、TN(t)、FN(t) 分别表示以 t 为阈值的阈值分类函数 $T_{h,t}(x)$ 在数据集 T 上的真正，假正，真负，假负的预测数。将

$$TPR(t) = \frac{TP(t)}{TP(t) + FN(t)}$$

称为真正率，将

$$FPR(t) = \frac{FP(t)}{FP(t) + TN(t)}$$

称为假正率。

定义

ROC 曲线以假正率为横坐标，以真正率为纵坐标构成的二维空间称为 ROC 空间，给定二元分类问题的概率预测模型 h 和阈值 $t \in [0,1]$。将阈值分类函数 $T_{h,t}(x)$ 的坐标 $(TPR(t), FPR(t))$ 都画在 ROC 空间，就构成了模型 h 的 ROC 曲线。

随着搜索步数 N 值的增大，ROC 曲线的形状从近似一条直线开始变化，其左上角区域的曲率不断增大，逐渐呈现出左上角近似于直角的曲线。搜索步数 N 值越大，得到的预测模型精确度也越高。ROC 曲线就越“弯曲”。这是因为用降低阈值来增大真正率的代价是同时增大了假正率。因为降低阈值 t 就会使得更多的 z 为 1，tp 和 fp 都会增大，从而 tpr 和 fpr 都增大。

```
z = threshold(t, proba)
tp = (y * z).sum()
fp = ((1 - y) * z).sum()
fpr.append(1.0 * fp / (fp + tn))
tpr.append(1.0 * tp / (tp + fn))
```

mnist_roc.py

因此，如果模型预测不精确，则假正率的增长将近似于真正率的增长，从而对应的 ROC 曲线就比较“平直”，而如果模型预测精确，即正负标签对应的概率两极分化，则在真正率增长的同时不会给假正率带来太大的变化，从而对应的 ROC 曲线就比较“弯曲”。

为了计算 ROC 曲线上的点，我们可以使用不同的分类阈值多次评估逻辑回归模型，但这样做效率非常低。幸运的是，有一种基于排序的高效算法可以为我们提供此类信息，这种算法称为曲线下面积（Area Under Curve）ROC 曲线越陡越好，所以理想值就是 1，一个正方形，而最差的随机判断都有 0.5，所以一般 AUC 的值是介于 0.5 到 1 之间的。

AUC 的一般判断标准：

- 0.5–0.7：效果较低
- 0.7–0.85：效果一般
- 0.85–0.95：效果很好
- 0.95–1：效果非常好，但一般不太可能

AUC 的物理意义：曲线下面积对所有可能的分类阈值的效果进行综合衡量。曲线下面积的一种解读方式是看作模型将某个随机正类别样本排列在某个随机负类别样本之上的概率。

案例实战 5.6：Mnist 数据集中手写数字识别 5 的 ROC 曲线和对应的 AUC 值。参考 mnist_roc.py

程序绘制了搜索步数为 $2, 10, 20, 100$ 时得到的 4 个模型的 ROC 曲线，分别计算他们的 AUC 测度。从程序运行的结果看，搜索步数 N 值的越大，得到的预测模型精确度越高。模型的概率预测越精确，ROC 曲线就越弯曲，即 ROC 曲线的形状反映出概率预测模型的预测效果。曲线越弯曲，曲线下方的面积越大，因此可以将面积作为曲线 “变曲” 程度的有效度量。

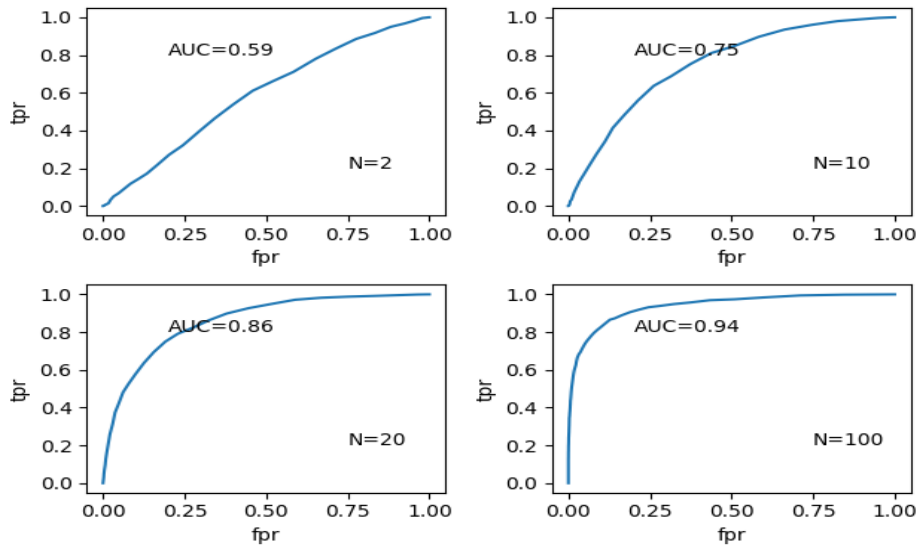

图 2: 4 个 N 值所对应的模型的 ROC 曲线的 AUC 测度

SoftMax 回归基本概念

SoftMax 函数，柔性最大值函数，又称归一化指数函数。它是二分类 Sigmoid 函数在多分类上的推广，目的是将多分类的结果以概率的形式展现出来，适用于求解 k 元分类问题。概率有两个性质：

1. 预测的概率为非负数；
2. 各种预测结果概率之和等于 1。

SoftMax 就是将在 $(-\infty, +\infty)$ 的预测结果按照以下两步转换为概率的。

1. 将预测结果转化为非负数。指数函数 $\exp(x)$ 的值域是 $(0, +\infty)$。SoftMax 第一步就是将模型的预测结果转化到指数函数上，这样保证了概率的非负性。
2. 各种预测结果概率之和等于 1。为了确保各个预测结果的概率之和等于 1。只需要将转换后的结果进行归一化处理。方法就是将转化后的结果除以所有转化后结果之和，可以理解为转化后结果占总数的百分比。这样就得到近似的概率。

SoftMax 回归基本概念

实例：假如模型对一个三分类问题的预测结果为 -2、0、1。我们要用 SoftMax 将模型结果转为概率。步骤如下：

1. 将预测结果转化为非负数
   $y_1 = \exp\ (x_1) = \exp\ (-2) = 0.14$
   $y_2 = \exp\ (x_2) = \exp\ (0) = 1$
   $y_3 = \exp\ (x_3) = \exp\ (1) = 2.72$
2. 各种预测结果概率之和等于 1
   $z_1 = y_1/(y_1 + y_2 + y_3) = 0.14/(0.14 + 1 + 2.72) = 0.036$
   $z_2 = y_2/(y_1 + y_2 + y_3) = 1/(0.14 + 1 + 2.72) = 0.259$
   $z_3 = y_3/(y_1 + y_2 + y_3) = 2.72/(0.14 + 1 + 2.72) = 0.705$

总结一下 SoftMax 如何将多分类输出转换为概率，可以分为两步：

1. 分子：通过指数函数，将实数输出映射到零到正无穷。
2. 分母：将所有结果相加，进行归一化。

在 k 元分类问题中，标签是一个 k 维的 $0-1$ 向量，相应的监督学习模型预测的是给定对象属于每一个分类的概率，因此预测模型输出的是 k 个概率值。一般情况下，可以采用 SoftMax 回归建立 k 元分类的预测模型。

多元回归算法 SoftMax 回归

定义 (SoftMax 模型)

给定一个 $n \times k$ 矩阵：$W = (\boldsymbol{w}_1, \boldsymbol{w}_2, \cdots, \boldsymbol{w}_k)$，其中，每个 $\boldsymbol{w}_j \in \mathbb{R}^n$ 为 $n \times 1$ 列向量 $(1 \leq j \leq k)$。*SoftMax* 模型 $\boldsymbol{h}_w : \mathbb{R}^n \to \mathbb{R}^k$ 为

$$\boldsymbol{h}_w(\boldsymbol{x}) = \left(\frac{e^{\boldsymbol{x}^T \boldsymbol{w}_1}}{\sum_{t=1}^{k} e^{\boldsymbol{x}^T \boldsymbol{w}_t}}, \frac{e^{\boldsymbol{x}^T \boldsymbol{w}_2}}{\sum_{t=1}^{k} e^{\boldsymbol{x}^T \boldsymbol{w}_t}}, \cdots, \frac{e^{\boldsymbol{x}^T \boldsymbol{w}_k}}{\sum_{t=1}^{k} e^{\boldsymbol{x}^T \boldsymbol{w}_t}} \right) \tag{12}$$

$h_w(\boldsymbol{x})$ 是一个 k 维向量，用 $h_{w_j}(\boldsymbol{x})$ 表示 $h_w(\boldsymbol{x})$ 的第 j 个分量，它是模型预测 $\boldsymbol{x}$ 属于第 j 类的概率，即 $Pr(\boldsymbol{x} \in \text{类} j) = h_{w_j}(\boldsymbol{x}) = \frac{e^{\boldsymbol{x}^T \boldsymbol{w}_j}}{\sum_{t=1}^{k} e^{\boldsymbol{x}^T \boldsymbol{w}_t}}$

式 (12) 中的 $h_{w_j}(\boldsymbol{x})$ 取值属于 $[0, 1]$ 区间，并且有

$$\sum_{j=1}^{k} \frac{e^{\boldsymbol{x}^T \boldsymbol{w}_j}}{\sum_{t=1}^{k} e^{\boldsymbol{x}^T \boldsymbol{w}_t}} = 1$$

因此是一个合法的概率分布。

定义 (SoftMax 回归算法的目标函数)

对于 $\boldsymbol{x}^{(i)} \in \mathbb{R}^n, \boldsymbol{y}^{(i)} \in \mathbb{R}^k, i = 1, 2, \cdots, m$，*给定* $n \times k$ *矩阵* $W = (\boldsymbol{w_1}, \boldsymbol{w_2}, \cdots, \boldsymbol{w_k})$，*定义 SoftMax 回归的目标函数为模型* $h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) = \left(h_{\boldsymbol{w_1}}(\boldsymbol{x}^{(i)}), h_{\boldsymbol{w_2}}(\boldsymbol{x}^{(i)}), \cdots, h_{\boldsymbol{w_k}}(\boldsymbol{x}^{(i)})\right)$ *的* k *元交叉熵。其表达式为*

$$f(\boldsymbol{w}) = -\frac{1}{m}\sum_{i=1}^{m} \boldsymbol{y}^{(i)T} \log h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)})$$

SoftMax 回归算法是以 SoftMax 函数作为模型假设，以 k 元交叉熵为目标函数的经验损失最小化算法

Algorithm 6: SoftMax 回归算法

Input: m 个训练数据 $S = \{(\boldsymbol{x}^{(1)}, y^{(1)}), (\boldsymbol{x}^{(2)}, y^{(2)}), \cdots, (\boldsymbol{x}^{(m)}, y^{(m)})\}$

Output: SoftMax 模型 $h_{\boldsymbol{W}}(\boldsymbol{x})$，使得 $W^* = (\boldsymbol{w}_1^*, \boldsymbol{w}_2^*, \cdots, \boldsymbol{w}_k^*)$ 为如下优化问题的最优解: $\min_{\boldsymbol{W} \in \mathbb{R}^{n \times k}} \frac{1}{m}\sum_{i=1}^{m} y^{(i)T} \log h_{\boldsymbol{W}}(\boldsymbol{x}^{(i)})$

SoftMax 回归优化算法使用 k 维的 $0-1$ 标签，若 $\boldsymbol{x}$ 属于 j，则相应的标签向量 $\boldsymbol{y}$ 在其第 j 个分量上取值为 1，其余分量上均为 0。

在多分类问题的 Softmax 回归算法中，通过最小化

$$f(W)=\frac{1}{m}\sum_{i=1}^{m}\boldsymbol{y}^{(i)T}\log h_{W}(\boldsymbol{x}^{(i)})$$

的 k 元交叉熵来求解预测模型的参数。由于 k 元交叉熵是一个凸函数，因而可以用梯度下降算法来优化 Softmax 回归算法，为此，需要计算 k 元交叉熵的梯度。

设 $(\boldsymbol{x},\boldsymbol{y})$ 为任意取定的一条训练数据。用 j 表示 $\boldsymbol{x}$ 所属的类，则模型在 $(\boldsymbol{x},\boldsymbol{y})$ 上的经验损失为

$$\ell(\boldsymbol{y},h_{W}(\boldsymbol{x}))=-\log(h_{w_j}(\boldsymbol{x}))=-\log\frac{e^{\boldsymbol{x}^{T}w}}{\sum_{t=1}^{k}e^{\boldsymbol{x}^{T}w_t}} \tag{13}$$

根据链式法则计算 $\ell(\boldsymbol{y},h_{W}(\boldsymbol{x}))$ 关于 $\boldsymbol{w}_j$ 的偏导数：

$$\frac{\partial\ell(\boldsymbol{y},\boldsymbol{h}_{W}(\boldsymbol{x}))}{\partial\boldsymbol{w}_j}=-\frac{\sum_{t=1}^{k}e^{\boldsymbol{x}^{T}w_t}}{e^{\boldsymbol{x}^{T}w_j}}\cdot\frac{\boldsymbol{x}e^{\boldsymbol{x}^{T}w_j}\sum_{t=1}^{k}e^{\boldsymbol{x}^{T}w_t}-\boldsymbol{x}e^{2\boldsymbol{x}^{T}w_j}}{(\sum_{t=1}^{k}e^{\boldsymbol{x}^{T}w_t})^{2}}=\boldsymbol{x}(h_{w_j}(\boldsymbol{x})-1) \tag{14}$$

对于任意 $r\neq j$ 有

$$\frac{\partial\ell(\boldsymbol{y},\boldsymbol{h}_{W}(\boldsymbol{x}))}{\partial\boldsymbol{w}_r}=\frac{\sum_{t=1}^{k}e^{\boldsymbol{x}^{T}w_t}}{e^{\boldsymbol{x}^{T}w_j}}\cdot\frac{\boldsymbol{x}e^{\boldsymbol{x}^{T}w_j}e^{\boldsymbol{x}^{T}w_r}}{(\sum_{t=1}^{k}e^{\boldsymbol{x}^{T}w_t})^{2}}=\boldsymbol{x}h_{w_r}(\boldsymbol{x}) \tag{15}$$

因为

$$
\begin{aligned}
&\nabla \ell(\boldsymbol{y}, h_{\boldsymbol{W}}(\boldsymbol{x})) \\
&= \left(\frac{\partial \ell(\boldsymbol{y}, \boldsymbol{h_W}(\boldsymbol{x}))}{\partial \boldsymbol{w}_1}, \frac{\partial \ell(\boldsymbol{y}, \boldsymbol{h_W}(\boldsymbol{x}))}{\partial \boldsymbol{w}_2}, \cdots, \frac{\partial \ell(\boldsymbol{y}, \boldsymbol{h_W}(\boldsymbol{x}))}{\partial \boldsymbol{w}_j}, \cdots, \frac{\partial \ell(\boldsymbol{y}, \boldsymbol{h_W}(\boldsymbol{x}))}{\partial \boldsymbol{w}_k} \right) \\
&= \boldsymbol{x} \left(h_{\boldsymbol{w}_1}(\boldsymbol{x}), h_{\boldsymbol{w}_2}(\boldsymbol{x}), \cdots, (h_{\boldsymbol{w}_j}(\boldsymbol{x}) - 1), \cdots, h_{\boldsymbol{w}_k}(\boldsymbol{x}) \right) \\
&= \boldsymbol{x} \left(h_{\boldsymbol{w}_1}(\boldsymbol{x}) - 0, h_{\boldsymbol{w}_2}(\boldsymbol{x}) - 0, \cdots, (h_{\boldsymbol{w}_j}(\boldsymbol{x}) - 1), \cdots, h_{\boldsymbol{w}_k}(\boldsymbol{x}) - 0 \right) \\
&= \boldsymbol{x}(h_{\boldsymbol{W}}(\boldsymbol{x}) - \boldsymbol{y})^T
\end{aligned}
\tag{16}
$$

其中 $h_{\boldsymbol{W}}(x) = (h_{\boldsymbol{w_1}}(\boldsymbol{x}) h_{\boldsymbol{w_2}}(\boldsymbol{x}) \cdots h_{\boldsymbol{w}_k}(\boldsymbol{x}))^T$, $\boldsymbol{y} = (0 \cdots 010 \cdots 0)^T$ 是 k 维列向量。由梯度的线性特性可知，

$$
\nabla f(W) = \frac{1}{m} \sum_{i=1}^{m} \boldsymbol{x}^{(i)} (h_{\boldsymbol{W}}(\boldsymbol{x}^{(i)}) - \boldsymbol{y}^{(i)})^T
$$

令 X 和 Y 分别为 $m \times n$ 矩阵和 $m \times k$ 矩阵，$h_W(X)$ 为 $m \times k$ 的矩阵：

$$
X = \begin{bmatrix} \boldsymbol{x}^{(1)T} \\ \boldsymbol{x}^{(2)T} \\ \vdots \\ \boldsymbol{x}^{(m)T} \end{bmatrix}, Y = \begin{bmatrix} \boldsymbol{y}^{(1)T} \\ \boldsymbol{y}^{(2)T} \\ \vdots \\ \boldsymbol{y}^{(m)T} \end{bmatrix}, h_W(X) = \begin{bmatrix} h_W(\boldsymbol{x}^{(1)})^T \\ h_W(\boldsymbol{x}^{(2)})^T \\ \vdots \\ h_W(\boldsymbol{x}^{(m)})^T \end{bmatrix}
$$

Softmax 回归的梯度下降实现

则可以将 $\nabla f(W)$ 表示为矩阵的形式：

$$\nabla f(W) = \frac{1}{m} X^T (h_W(X) - Y)$$

$$= \frac{1}{m} (\boldsymbol{x}^{(1)} \boldsymbol{x}^{(2)} \cdots \boldsymbol{x}^{(m)}) \left(\begin{bmatrix} h_W(\boldsymbol{x}^{(1)})^T \\ h_W(\boldsymbol{x}^{(2)})^T \\ \vdots \\ h_W(\boldsymbol{x}^{(m)})^T \end{bmatrix} - \begin{bmatrix} \boldsymbol{y}^{(1)T} \\ \boldsymbol{y}^{(2)T} \\ \vdots \\ \boldsymbol{y}^{(m)T} \end{bmatrix} \right)$$

$$= \frac{1}{m} (\boldsymbol{x}^{(1)} \boldsymbol{x}^{(2)} \cdots \boldsymbol{x}^{(m)}) \begin{bmatrix} h_W(\boldsymbol{x}^{(1)})^T - \boldsymbol{y}^{(1)T} \\ h_W(\boldsymbol{x}^{(2)})^T - \boldsymbol{y}^{(2)T} \\ \vdots \\ h_W(\boldsymbol{x}^{(m)})^T - \boldsymbol{y}^{(m)T} \end{bmatrix}$$

SoftMax 回归算法的实现，参考 softmax_regression_gd.py。

Python 实现的代码 softmax_regression_gd.py：

```
import numpy as np
class SoftmaxRegression:
    def __init__(self):
        self.w = None
    def fit(self, X, y, eta=0.1, N=5000):
        m, n = X.shape # m,n分别是样本的数量和每个样本的特征数
        m, k = y.shape # m,k是分别标签的数量和标签的k维向量取值形式
        w = np.zeros([n, k])
        for t in range(N):
            proba = self.softmax(X.dot(w)) # proba 就是h_{w}(x)
            g = X.T.dot(proba - y) / m # g就是f(w)的梯度
            w = w - eta * g # 进行梯度下降算法的迭代
        self.w = w
    @staticmethod
    def softmax(x): # x会传入X@w,返回值e即为归一化后的h_{w}(x)
        e = np.exp(x)
        s = e.sum(axis=1)
        for i in range(len(s)):
            e[i] /= s[i]
        return e
    def predict_proba(self, X): # 进行概率值预测
        return self.softmax(X.dot(self.w))
    def predict(self, X): # 获取概率值最大的分量下标，将其作为判别的类别
        proba = self.predict_proba(X)
        return np.argmax(proba, axis=1)
```

因为 SoftMax 回归优化算法使用 k 维的 $0-1$ 标签，而有时候数据集提供的数据类型是 $0,1,2$ 等整数，所以需要先将标签转化为 k 维 $0-1$ 标签 $\{0,1\}^k$，该过程称为 onehot 编码。

案例实战 5.7：将一维数组进行 onehot 编码。参考 one_hot.py

```
import numpy as np
def one_hot_encoding(y):
    m = len(y)
    k = np.max(y) + 1
    result = np.zeros([m, k])
    for i in range(m):
        result[i, y[i]] = 1
    return result
```

案例实战 5.8：鸢尾花预测问题。(对鸢尾花进行 3 分类) iris_softmax_gd.py
随机梯度下降算法实现 softmax 回归的类：softmax_regression_sgd.py
案例实战 5.9：手写数字识别问题。
mnist_regression_sgd.py

Sklearn 的 Logistic 回归算法

Sklearn 中的 LogisticRegression 类的用法。class sklearn.linear_model.LogisticRegression(penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None)

案例实战 5.10 利用鸢尾花花瓣的长宽，对比使用不同的分类方式进行分类。

iris_logistic_regression_sklearn.py