练习二——线性分类

In [1]:
```python
# 导入相关的库
import pandas as pd
import numpy as np
```

In [2]:
```python
# 1. 数据加载
names = ['Sample code number', 'Clump Thickness', 'Uniformity of Cell Size',
         'Uniformity of Cell Shape', 'Marginal Adhesion', 'Single Epithelial Cel
         'Bare Nuclei', 'Bland Chromatin', 'Normal Nucleoli', 'Mitoses', 'Class'

data_path = "data/breast-cancer-wisconsin.data"
data = pd.read_csv(data_path, names=names)

# 2 数据预处理
# 2.1 数据清洗：去除含缺失值的样本
data = data.replace(to_replace="?", value=np.nan)
data = data.dropna()

# 2.2 将特征列和标签列转换为数值类型
data.iloc[:, 1:] = data.iloc[:, 1:].apply(pd.to_numeric, errors='coerce')
data = data.dropna()

x = data.iloc[:, 1:10].values.astype(np.float64)
y = data["Class"].values.astype(int)
y = np.where(y == 4, 1, 0)
```

In [3]:
```python
# 2.3 数据集划分
def train_test_split_manual(X, y, test_size=0.25, random_state=2025):
    """手动实现训练集和测试集的划分"""
    if random_state is not None:
        np.random.seed(random_state)

    n_samples = X.shape[0]
    n_test = int(n_samples * test_size)

    # 生成随机索引
    indices = np.random.permutation(n_samples)
    test_indices = indices[:n_test]
    train_indices = indices[n_test:]

    return X[train_indices], X[test_indices], y[train_indices], y[test_indices]
```

In [4]:
```python
# 2.4 特征标准化
class StandardScaler:
    # TODO
    # 实现特征标准化
    # 标准化公式: z = (x - mean) / std

    def __init__(self):
        self.mean_ = None
        self.std_ = None

    def fit(self, X):
        """计算均值和标准差"""
```

```
        # TODO: 请在此处完成代码
        self.mean_ = np.mean(X, axis=0)
        self.std_ = np.std(X, axis=0)

    def transform(self, X):
        """使用均值和标准差进行标准化"""
        # TODO: 请在此处完成代码
        return (X - self.mean_) / self.std_

    def fit_transform(self, X):
        """拟合并转换"""
        self.fit(X)
        return self.transform(X)
```

逻辑回归

模型形式

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

- x：输入特征向量
- θ：模型参数
- $h_\theta(x)$：预测结果（属于正类的概率）

$$\text{class} = \begin{cases} 1, & \text{if } h_\theta(x) \geq 0.5 \\ 0, & \text{if } h_\theta(x) < 0.5 \end{cases}$$

损失函数：

$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right]$

优化：梯度下降

$\theta := \theta - \alpha\frac{\partial J(\theta)}{\partial \theta}$

其中 α 为学习率

In [5]:
```
# 3. 实现逻辑回归
class LogisticRegression:
    """
    参数:
        learning_rate: 学习率
        n_iterations: 迭代次数
    """
    def __init__(self, learning_rate=0.1, n_iterations=2000):
        self.learning_rate = learning_rate
        self.n_iterations = n_iterations
        self.weights = None
        self.bias = None

    def sigmoid(self, z: np.ndarray) -> np.ndarray:
        # TODO: 实现sigmoid函数
        # sigmoid(z) = 1 / (1 + e^(-z))
```

```
        return 1 / (1 + np.exp(-z))

    def fit(self, X: np.ndarray, y: np.ndarray) -> None:
        # TODO: 训练模型
        # 使用批量梯度下降法（BatchGradientDescent，BGD）优化权重和偏置

        # 提示:
        # 1. 初始化权重和偏置
        # 2. 进行n_iterations次迭代
        # 3. 每次迭代:
        #     step1. 计算预测值
        #     step2. 计算梯度
        #     step3. 更新参数
        m,n = X.shape
        self.weights = np.zeros(n)
        self.bias = 0.0

        for _ in range(self.n_iterations):
            linear_model = np.dot(X, self.weights) + self.bias
            y_predicted = self.sigmoid(linear_model)

            dw = (1/m) * np.dot(X.T, (y_predicted - y))
            db = (1/m) * np.sum(y_predicted - y)

            self.weights -= self.learning_rate * dw
            self.bias -= self.learning_rate * db

    def predict(self, X: np.ndarray, threshold: float = 0.5) -> np.ndarray:
        # TODO: 预测
        # 提示:
        # 1. 进行线性运算
        # 2. 进行sigmoid计算
        # 3. 将结果与阈值进行比较
        linear_model = np.dot(X, self.weights) + self.bias
        y_predicted = self.sigmoid(linear_model)
        y_predicted_cls = [1 if i > threshold else 0 for i in y_predicted]
        return np.array(y_predicted_cls)
```

In [6]:

```
def get_metrics(y_true, y_pred):
    """
        获得评测指标
    """
    def recall_score(y_true, y_pred):
        """
        计算召回率
        召回率 = TP / (TP + FN)
        """
        TP = np.sum((y_true == 1) & (y_pred == 1))
        FN = np.sum((y_true == 1) & (y_pred == 0))
        return TP / (TP + FN) if (TP + FN) > 0 else 0.0

    def precision_score(y_true, y_pred):
        """精确率 = TP / (TP + FP)"""
        TP = np.sum((y_true == 1) & (y_pred == 1))
        FP = np.sum((y_true == 0) & (y_pred == 1))
        return TP / (TP + FP) if (TP + FP) > 0 else 0.0

    def accuracy_score(y_true, y_pred):
        """准确率 = (TP + TN) / 总数"""
        return np.mean(y_true == y_pred)
```

```
    def confusion_matrix(y_true, y_pred):
        """混淆矩阵"""
        TN = np.sum((y_true == 0) & (y_pred == 0))
        FP = np.sum((y_true == 0) & (y_pred == 1))
        FN = np.sum((y_true == 1) & (y_pred == 0))
        TP = np.sum((y_true == 1) & (y_pred == 1))

        return np.array([[TN, FP], [FN, TP]])
    recall = recall_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    accuracy = accuracy_score(y_true, y_pred)
    cm = confusion_matrix(y_true, y_pred)
    return recall, precision, accuracy, cm
```

In [7]:
```
if __name__=='__main__':
    print("=" * 60)
    print("乳腺癌诊断预测 - 逻辑回归实现")
    print("=" * 60)

    # 划分数据集
    X_train, X_test, y_train, y_test = train_test_split_manual(x, y, test_size=0

    print(f"\n数据集信息:")
    print(f"训练集样本数: {X_train.shape[0]}")
    print(f"测试集样本数: {X_test.shape[0]}")
    print(f"特征数: {X_train.shape[1]}")

    # 特征标准化
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    print("\n开始训练模型...")
    # 训练逻辑回归模型
    model = LogisticRegression(learning_rate=0.01, n_iterations=1000)
    model.fit(X_train, y_train)

    # 预测
    y_pred = model.predict(X_test)

    # 评估
    print("\n" + "=" * 60)
    print("模型评估结果")
    print("=" * 60)

    recall, precision, accuracy, cm = get_metrics(y_test, y_pred)

    print(f"\n召回率 (Recall):    {recall:.4f} ({recall*100:.2f}%)")
    print(f"精确率 (Precision): {precision:.4f} ({precision*100:.2f}%)")
    print(f"准确率 (Accuracy):  {accuracy:.4f} ({accuracy*100:.2f}%)")

    print(f"\n混淆矩阵:")
    print(f"                预测负类  预测正类")
    print(f"实际负类        {cm[0,0]:4d}     {cm[0,1]:4d}")
    print(f"实际正类        {cm[1,0]:4d}     {cm[1,1]:4d}")
```

```
============================================================
乳腺癌诊断预测 - 逻辑回归实现
============================================================

数据集信息:
训练集样本数: 513
测试集样本数: 170
特征数: 9

开始训练模型...

============================================================
模型评估结果
============================================================

召回率 (Recall):    0.9831 (98.31%)
精确率 (Precision): 0.9667 (96.67%)
准确率 (Accuracy):  0.9824 (98.24%)

混淆矩阵:
              预测负类  预测正类
实际负类          109         2
实际正类            1        58
```

相关问题

问题一：逻辑回归的数学原理

1. sigmoid函数有什么重要的数学性质?

   sigmoid函数的输出范围严格在(0,1)之间，可以被解释为概率；单调递增；处处可导且导数有较好的形式。

2. 逻辑回归使用什么损失函数，为什么不能使用均方误差（MSE）？

   使用交叉熵损失函数。因为MSE在此场景下是非凸的，存在大量局部最小值，难以找到全局最优解；且在概率逼近0或1时梯度消失，优化困难。

问题二：召回率的理解

在本次作业中，我们引入召回率（Recall）作为模型评估的重要指标。请简单阐明在实验数据集上使用召回率的意义。

本实验数据集为乳腺癌数据集，医疗诊断中误判的后果十分严重，而高召回率保证了模型捕获绝大多数真正恶性样本，避免误诊。

问题三：softmax回归的基础概念

1. softmax函数的核心作用是什么?

   是将任意实数向量转换为概率分布，输出各类别的相对概率。

2. softmax与普通的归一化（如除以总和）有什么本质区别?

softmax函数使用指数函数，能放大类别间的差异，更加适合分类决策。

3. softmax的计算公式如下，但在实现时可能面临指数过大带来的溢出问题，可以怎么处理?

$$\mathrm{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}, \quad i = 1, \ldots, n$$

可以通过减去向量最大值来进行数值稳定，即计算 softmax(x_i - max(x)) = e^{x_i - max(x)} / \sum e^{x_j - max(x)}。