逻辑回归

课前准备

- 下载Anaconda软件，请点击这里进行下载。
- 函数链式求导法则。

本节要点

- 逻辑回归算法预测原理。
- sigmoid函数与对数损失函数。
- 逻辑回归算法参数估计。
- 决策边界与模型的关系。

逻辑回归模型

分类思想

逻辑回归，我们不要被其名字所误导，实际上，逻辑回归是一个分类算法，作用是可以将样本划分为不同的类别。例如，给定如下的鸢尾花数据集：

花萼长度	花萼宽度	花瓣长度	花瓣宽度	类别
5.4	3.9	1.3	0.4	山鸢尾
5.1	3.5	1.4	0.3	山鸢尾
5.8	2.7	4.1	1.	杂色鸢尾
6.2	2.2	4.5	1.5	杂色鸢尾
6.3	2.5	5.	1.9	维吉尼亚鸢尾
6.5	3.	5.2	2.	维吉尼亚鸢尾
5.6	3.1	3.8	0.7	?

我们就可以基于已知的数据集，建立模型，从而可以对未知的样本数据进行分类预测。

逻辑回归实现分类的思想为：将每条样本进行“打分”，然后设置一个阈值，达到这个阈值的，分为一个类别，而没有达到这个阈值的，分为另外一个类别。对于阈值（临界值），没有明确的划分，即划分为哪个类别都可以，但是，要保证阈值划分的一致性。

算法模型

对于逻辑回归，模型的前面与线性回归类似：

$$
\begin{aligned}
z &= w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n \\
&= \sum_{j=1}^{n} w_jx_j + w_0 \\
&= \sum_{j=0}^{n} w_jx_j \\
&= \vec{w}^T \cdot \vec{x}
\end{aligned}
$$

不过，z的值是一个连续的值，取值范围为$(-\infty, +\infty)$，我们可以将阈值设置为中间的位置，也就是0，当$z > 0$时，模型将样本判定为一个类别（正例），当$z \leq 0$时，模型将样本判定为另外一个类别（负例）。这样，模型就实现了二分类的任务。

假设真实类别y的值为1与0，则模型的预测结果$\hat{y}$为：

$$
\hat{y} = \begin{cases} 1 & z > 0 \\ 0 & z \leqslant 0 \end{cases}
$$

说明：

- 因为模型需要数值类型，因此，我们习惯上将类别变量映射为离散变量来表示。
- 在scikit-learn中，如果z值为0，则判定为负例。

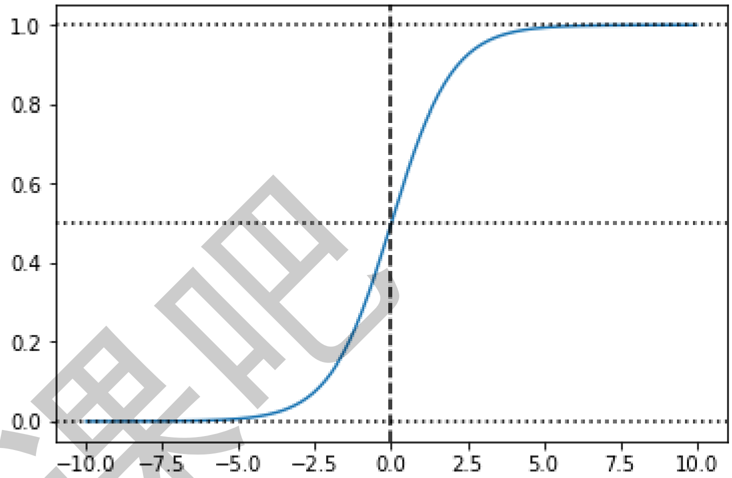
sigmoid函数

函数原型

对于分类任务来说，如果仅仅给出分类的结果，这些信息可能是不充分的。如果在分类的同时，也能够提供样本属于该类别的概率，这在现实中是非常实用的。例如，某人患病的概率，明天下雨的概率等。

因此，我们需要将z的值转换为概率值，逻辑回归使用sigmoid函数来实现转换，该函数的原型为：

$$
sigmoid(z) = \frac{1}{1+e^{-z}}
$$



当z的值从$-\infty$向$+\infty$过渡时，sigmoid函数的取值范围为(0, 1)，这正好是概率的取值范围，当$z=0$时，sigmoid(0)的值为0.5。因此，模型就可以将sigmoid(z)作为样本属于正例（1）的概率，而1 - sigmoid(z)作为样本属于负例（0）的概率。然后，根据sigmoid(z)与1 - sigmoid(z)值的大小，来决定预测结果。

$$\hat{y}=\begin{cases}1 & sigmoid(z)>1-sigmoid(z)\\ 0 & sigmoid(z)\leqslant 1-sigmoid(z)\end{cases}$$

我们也可以这样理解：

$$\hat{y}=\begin{cases}1 & sigmoid(z)>0.5\\ 0 & sigmoid(z)\leqslant 0.5\end{cases}$$

课堂练习

关于sigmoid函数，说法正确的是（ ）。【不定项】 ABCD

A sigmoid函数的定义域为$(-\infty,+\infty)$

B sigmoid函数的值域为(0，1)。

C sigmoid函数是非线性函数。

D sigmoid函数的优势是能够以概率的方式来呈现样本属于某个类别的可能性。

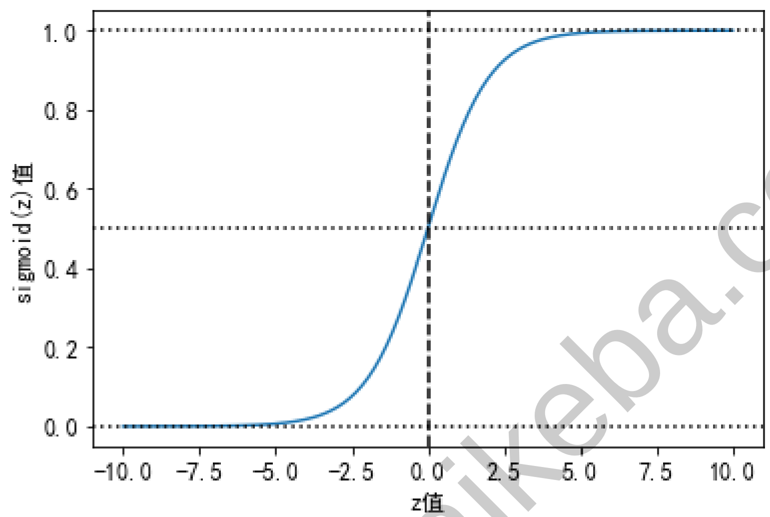
sigmoid函数图像

现在，我们通过Python程序来绘制sigmoid函数在[-10, 10]区间的图像。

```python
import numpy as np
import matplotlib.pyplot as plt

plt.rcParams["font.family"] = "SimHei"
plt.rcParams["axes.unicode_minus"] = False
plt.rcParams["font.size"] = 12
```

```python


def sigmoid(z):
    """sigmoid函数的程序实现。

    Parameters
    ----------
    z : array-like
        需要求解的数值。

    Returns
    -------
    r : array-like
        对应每个元素的sigmoid值（概率值）。

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


z = np.linspace(-10, 10, 200)
plt.plot(z, sigmoid(z))
# 绘制水平线与垂直线。
plt.axvline(x=0, ls="--", c="k")
plt.axhline(ls=":", c="k")
plt.axhline(y=0.5, ls=":", c="k")
plt.axhline(y=1, ls=":", c="k")
plt.xlabel("z值")
plt.ylabel("sigmoid(z)值")
```

```
Text(0, 0.5, 'sigmoid(z)值')
```



损失函数

损失函数分析

在逻辑回归中，使用sigmoid(z)来表示样本属于类别1的概率，1 - sigmoid(z)来表示样本属于类别0的概率。 为了方便起见，这里使用s(z)代表sigmoid(z)。

$$
p(y=1 \mid x; w) = s(z)
$$
$$
p(y=0 \mid x; w) = 1 - s(z)
$$

以上两个式子表示当真实类别为1（0）时，模型预测为1（0）的概率。但是，以上表示有些繁琐，不方便计算损失值。我们可以将以上两个式子综合表示为：

$$
p(y|x; w) = s(z)^y (1 - s(z))^{1-y}
$$

在线性回归中，我们使用最小平方损失函数，通过构建回归方程，来预测样本的标签值。因而，我们就可以考虑效仿线性回归，这样建立损失函数（最小平方和）：

$$
J(w) = \frac{1}{2} * \sum_{i=1}^{m} (y^{(i)} - s(z^{(i)}))^2
$$

我们可以这样理解，为了能够让损失函数的值最小，则：

- 如果样本的真实值为正例（1），则sigmoid(z)应该尽可能的大。
- 如果样本的真实值为负例（0），则sigmoid(z)应该尽可能的小（1 - sigmoid(z)应该尽可能的大）。

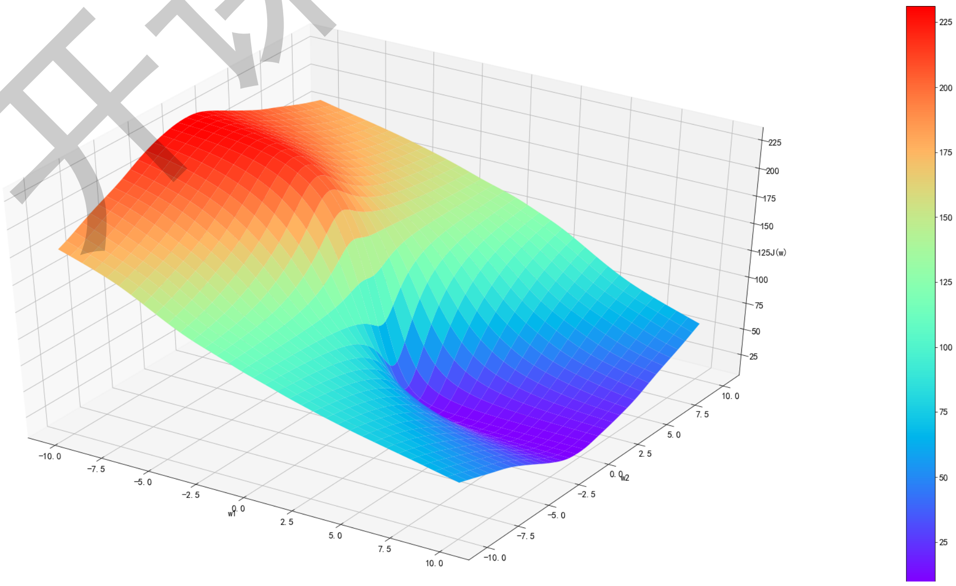
课堂练习

我们设定的损失函数合乎逻辑吗？ A

A 是。

B 否。

然而，在逻辑回归中，sigmoid函数是一个非线性函数，这会使得最小平方和的损失函数比较复杂，其不是一个凸函数，不方便优化求解。



对数损失函数

根据极大似然估计，所有样本的联合概率密度函数（即似然函数）为：

$$L(w)=\prod_{i=1}^{m}p(y^{(i)}|x^{(i)};w)$$
$$=\prod_{i=1}^{m}s(z^{(i)})^{y^{(i)}}(1-s(z^{(i)}))^{1-y^{(i)}}$$

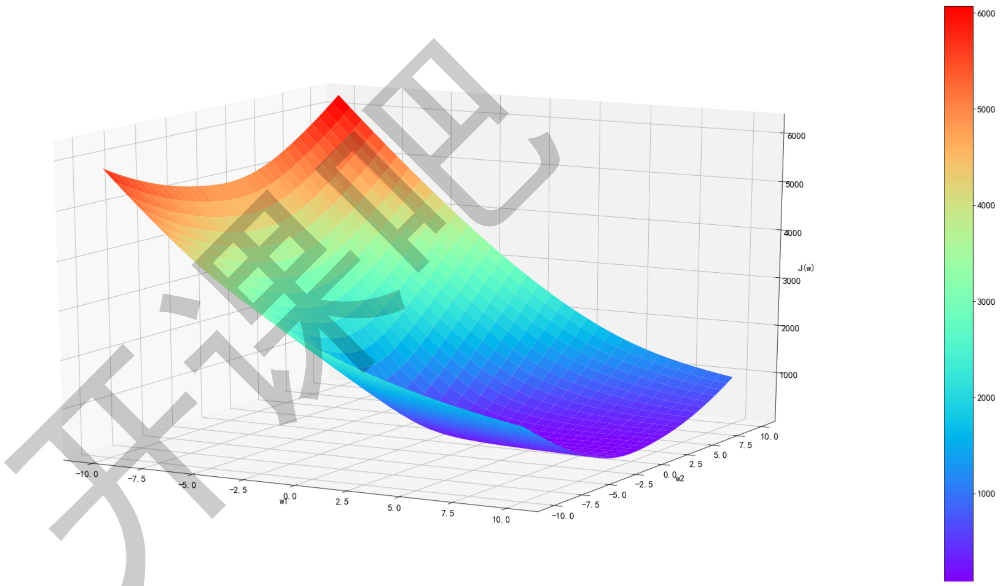
为了方便求解，我们取对数似然函数，让累积乘积变成累积求和：

$$lnL(w)=ln[\prod_{i=1}^{m}s(z^{(i)})^{y^{(i)}}(1-s(z^{(i)}))^{1-y^{(i)}}]$$
$$=\sum_{i=1}^{m}[y^{(i)}lns(z^{(i)})+(1-y^{(i)})ln(1-s(z^{(i)}))]$$

我们要使得上式的值最大，可以采用梯度上升的方式。不过，这里我们为了引入损失函数的概念，我们采用相反的方式，即只需要使得该值的相反数最小即可，因此，我们可以将上式的相反数作为逻辑回归的损失函数（对数损失函数）：

$$J(w) = -\sum_{i=1}^{m}[y^{(i)} lns(z^{(i)}) + (1 - y^{(i)})ln(1 - s(z^{(i)}))]$$

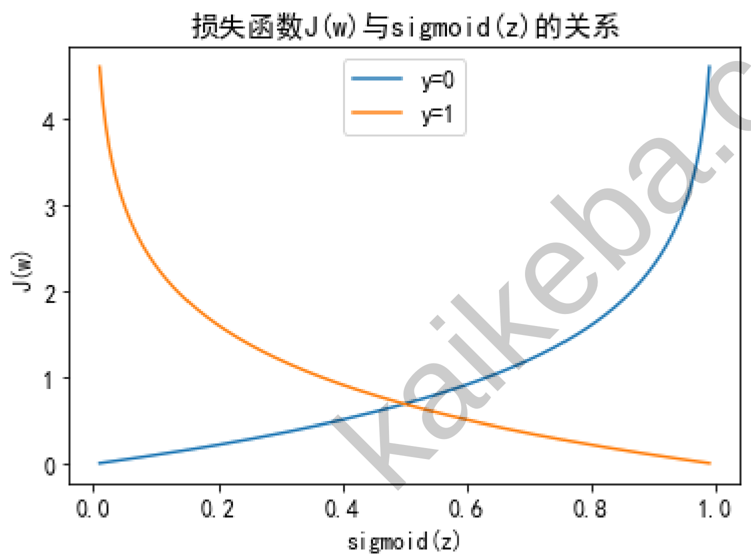
相比于最小平方和损失函数，对数损失函数简单很多，且该损失函数是凸函数，容易优化求解。



损失函数与sigmoid

模型拟合训练数据，其目标就是：当数据为类别1时，我们应当让$sigmoid(z)$的值尽可能大，反之，当数据为类别0时，我们应当让$sigmoid(z)$的值尽可能小，即1 - $sigmoid(z)$的值尽可能大。

```python
# 定义sigmoid(z)的取值。
s = np.linspace(0.01, 0.99, 200)
for y in [0, 1]:
    # 计算损失函数J(w)的值。
    loss = -y * np.log(s)-(1 - y) * np.log(1 - s)
    plt.plot(s, loss, label=f"y={y}")
plt.legend()
plt.xlabel("sigmoid(z)")
plt.ylabel("J(w)")
plt.title("损失函数J(w)与sigmoid(z)的关系")
plt.show()
```



课堂练习

线性回归与逻辑回归的比较，正确的是（ ）。 C

A 损失函数是相同的。

B 都是用于回归任务。

C 都是线性加权计算的模型。

D 都不能解决非线性问题。

参数求解

$J(w)$求导

与线性回归类似，逻辑回归模型的关键之处，就是要求解模型的参数，也就是w（包括偏置）的值，一旦参数值确定，我们就能够对未知样本数据实现预测。

我们可以使用梯度下降方法来求解w，使得损失函数的值最小。这里先考虑对一个样本中的w求偏导，对于所有样本，只需要依次将所有样本的偏导结果累加即可（a + b的导数等于各自的导数再相加）。

$$
\begin{aligned}
\frac{\partial J(w)}{\partial w_j} &= \frac{\partial}{\partial w_j}\{-[y^{(i)}lns(z^{(i)}) + (1-y^{(i)})ln(1-s(z^{(i)}))]\} \\
&= -(\frac{y^{(i)}}{s(z^{(i)})} - \frac{1-y^{(i)}}{1-s(z^{(i)})})\frac{\partial s(z^{(i)})}{\partial w_j} \\
&= -(\frac{y^{(i)}}{s(z^{(i)})} - \frac{1-y^{(i)}}{1-s(z^{(i)})})\frac{\partial s(z^{(i)})}{\partial z^{(i)}}\frac{\partial z^{(i)}}{\partial w_j} \qquad (1)
\end{aligned}
$$

sigmoid函数求导

在（1）式中：

$$\frac{\partial s(z^{(i)})}{\partial z^{(i)}}$$

就是对sigmoid函数求导，即$s'(z)$。

$$
\begin{aligned}
s'(z) &= (\frac{1}{1+e^{-z}})' \\
&= -\frac{1}{(1+e^{-z})^2}(-e^{-z}) \\
&= (\frac{1}{1+e^{-z}})(\frac{e^{-z}}{1+e^{-z}}) \\
&= (\frac{1}{1+e^{-z}})(1-\frac{1}{1+e^{-z}}) \\
&= s(z)(1-s(z)) \qquad (2)
\end{aligned}
$$

将（2）式带入（1）式：

$$
\begin{aligned}
& -(\frac{y^{(i)}}{s(z^{(i)})}-\frac{1-y^{(i)}}{1-s(z^{(i)})})\frac{\partial s(z^{(i)})}{\partial z}\frac{\partial z^{(i)}}{\partial w_j} \\
&= -(\frac{y^{(i)}}{s(z^{(i)})}-\frac{1-y^{(i)}}{1-s(z^{(i)})})s(z^{(i)})(1-s(z^{(i)}))x_j^{(i)} \\
&= -[(y^{(i)}(1-s(z^{(i)}))-(1-y^{(i)})s(z^{(i)})]x_j{}^{(i)} \\
&= -(y^{(i)}-s(z^{(i)})x_j^{(i)}
\end{aligned}
$$

因此，在随机梯度下降中，依次针对每个样本更新，我们就可以这样更新权重：

$$
w_j = w_j + \eta(y^{(i)} - s(z^{(i)})x_j^{(i)}
$$

而对于批量梯度下降或小批量梯度下降，只需将样本数量相加即可：

$$
w_j = w_j + \eta * \frac{1}{k}\sum_{i=1}^{m}(y^{(i)} - s(z^{(i)})x_j^{(i)}
$$

经过不断反复迭代，最终求得合适的w值，使得损失函数的值最小（接近最小）。

- η：学习率。
- k：用于计算梯度的样本数量。

逻辑回归实现二分类

模型训练与预测

我们以鸢尾花数据集为例，演示通过逻辑回归算法实现二分类。

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
import warnings

warnings.filterwarnings("ignore")

iris = load_iris()
X, y = iris.data, iris.target
# 因为鸢尾花具有三个类别，4个特征，此处仅使用其中两个特征，并且移除一个类别（类别0）。
X = X[y != 0, 2:]
y = y[y != 0]
# 此时，y的标签为1与2，我们这里将其改成0与1。（仅仅是为了习惯而已）
y[y == 1] = 0
y[y == 2] = 1
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=2)
# penalty：正则化方式。可选值为：
#      l1：L1正则化。
#      l2（默认值）：L2正则化。
#      elasticnet：弹性网络正则化。
#      none：不使用正则化，在这种情况下，会使用L2正则化，并将C值设置为无穷大。
# C：正则化强度，类似于线性回归中的alpha参数值。可以看做C为alpha的倒数，默认值为1.0。
# solver：优化求解算法。可选值为：
#      liblinear：使用C++的liblinear库，支持ovr，不支持multinomial。支持L1，L2正则化。
#      newton-cg：牛顿法，使用海森矩阵（损失函数的二阶偏导）的逆矩阵来更新权重。支持L2正则化
#                 与不使用正则化。
#      lbfgs（默认值）：拟牛顿法，牛顿发的一种变体，不去计算海森矩阵，而是近似去构造海森矩阵。
#                       支持L2正则化与不使用正则化。
```

```
#     sag：平均随机梯度下降法（Stochastic Average Gradient Descent），类似于梯度下降，只是
#           在更新权重时，考虑样本旧的梯度值。支持L2正则化与不使用正则化。
#     saga：sag的一种变体（改进），理论上具有更好的收敛速度，支持所有类型正则化。
# multi_class：多分类的实现方式，可选值为：
#     auto（默认值）：如果二分类，或者sover为liblinear，使用ovr，其他情况使用multinomial。
#     ovr：one-versus-rest实现方式。
#     multinomial：多项式实现方式。
lr = LogisticRegression()
lr.fit(X_train, y_train)
y_hat = lr.predict(X_test)
print("权重：", lr.coef_)
print("偏置：", lr.intercept_)
print("真实值：", y_test)
print("预测值：", y_hat)
```

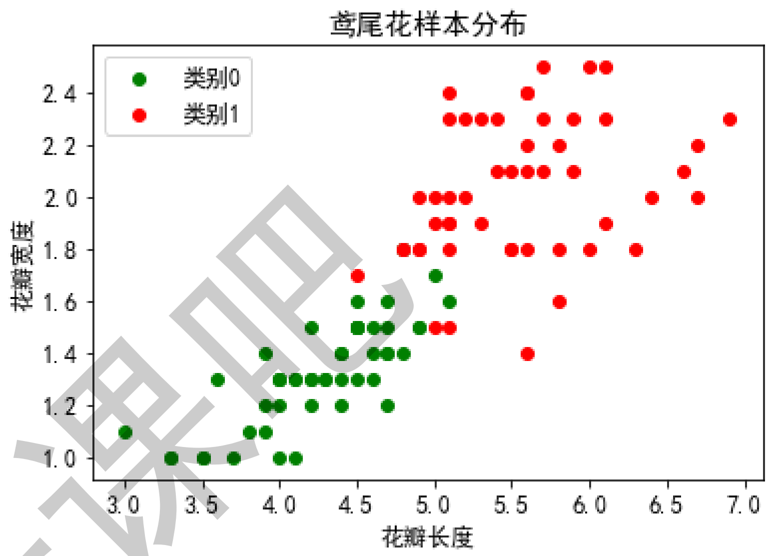
```
权重：  [[2.54536368 2.15257324]]
偏置：  [-16.08741502]
真实值：  [1 0 1 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 0 0 1 0 1]
预测值：  [1 0 0 0 0 0 0 1 0 0 1 1 1 0 0 0 0 0 1 1 0 0 1 0 1]
```

结果可视化

现在，我们对分类的结果进行可视化显示。首先，我们来绘制下鸢尾花数据的分布图。

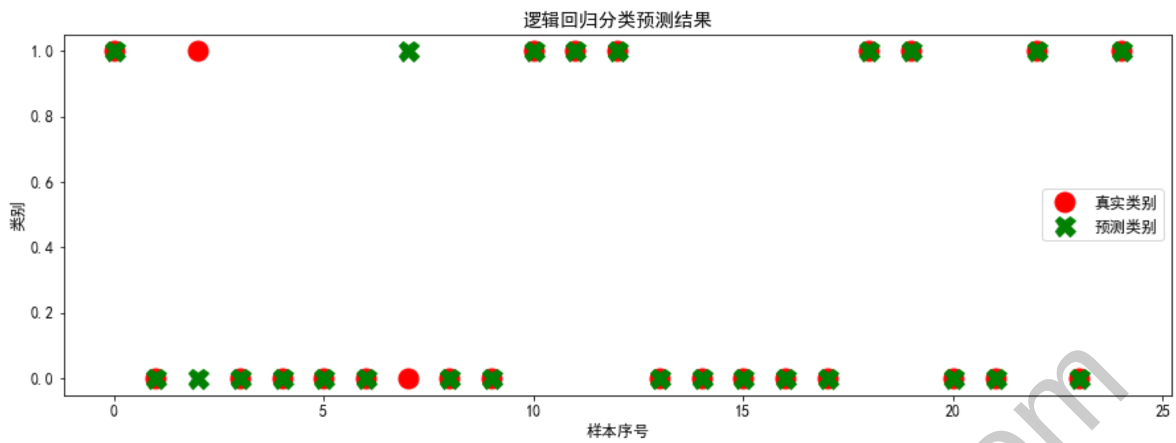
```
c1 = X[y == 0]
c2 = X[y == 1]
plt.scatter(x=c1[:, 0], y=c1[:, 1], c="g", label="类别0")
plt.scatter(x=c2[:, 0], y=c2[:, 1], c="r", label="类别1")
plt.xlabel("花瓣长度")
plt.ylabel("花瓣宽度")
plt.title("鸢尾花样本分布")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x1dccb887108>
```



接下来，我们来绘制在测试集中，样本的真实类别与预测类别。

```
plt.figure(figsize=(15, 5))
plt.plot(y_test, marker="o", ls="", ms=15, c="r", label="真实类别")
plt.plot(y_hat, marker="X", ls="", ms=15, c="g", label="预测类别")
plt.legend()
plt.xlabel("样本序号")
plt.ylabel("类别")
plt.title("逻辑回归分类预测结果")
plt.show()
```



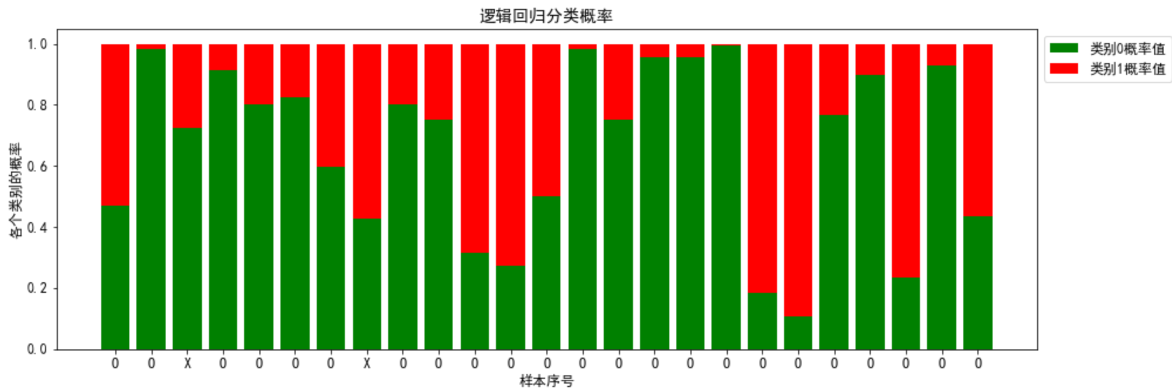
计算概率值

之前我们讲过，作为分类模型，不仅能够预测样本所属的类别，而且，还可以预测属于各个类别的概率。这在实践中是非常有意义的。接下来，我们就来求解逻辑回归预测的概率值。

```
# 获取预测的概率值，包含数据属于每个类别的概率。
probability = lr.predict_proba(X_test)
print(probability[:5])
# 获取每一行最大值的索引，就等价于预测结果。
print(np.argmax(probability, axis=1))
# 产生序号，用于可视化的横坐标。
index = np.arange(len(X_test))
# 分别获取类别0与类别1的概率。
pro_0 = probability[:, 0]
pro_1 = probability[:, 1]
tick_label = np.where(y_test == y_hat, "O", "X")
```

```
plt.figure(figsize=(15, 5))
# 绘制堆叠图
plt.bar(index, height=pro_0, color="g", label="类别0概率值")
# bottom=x，表示从x的值开始堆叠上去。
# tick_label 设置标签刻度的文本内容。
plt.bar(index, height=pro_1, color='r', bottom=pro_0, label="类别1概率值",
tick_label=tick_label)
plt.legend(loc="best", bbox_to_anchor=(1, 1))
plt.xlabel("样本序号")
plt.ylabel("各个类别的概率")
plt.title("逻辑回归分类概率")
plt.show()
```

```
[[0.46933862 0.53066138]
 [0.98282882 0.01717118]
 [0.72589695 0.27410305]
 [0.91245661 0.08754339]
 [0.80288412 0.19711588]]
[1 0 0 0 0 0 0 1 0 0 1 1 1 0 0 0 0 0 1 1 0 0 1 0 1]
```



课堂练习

关于模型predict_proba方法返回的概率，说法正确的是（ ）。【不定项】 AC

A 该方法会返回样本属于每个类别的概率。

B 该方法返回的概率值越大越好。

C 该方法返回的概率是通过sigmoid函数计算的（二分类下）。

D 该方法返回的概率值与决策函数的输出值（z）是没有关系的。

绘制决策边界

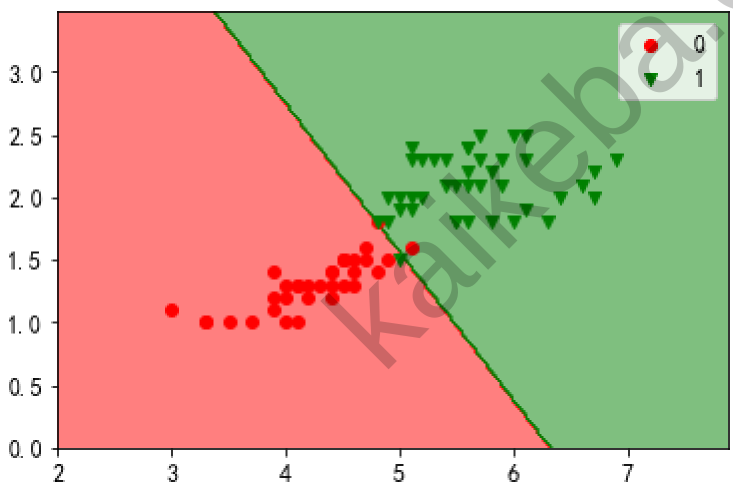
我们可以绘制决策边界，将分类效果进行可视化显示。首先，我们来定义绘制决策边界的函数。

```
from matplotlib.colors import ListedColormap

def plot_decision_boundary(model, X, y):
    """绘制model模型的决策边界。

    绘制决策边界，同时绘制样本数据X与对应的类别y,
```

```
    用于可视化模型的分类效果。

    Parameters
    ----------
    model : object
        模型对象。
    X : array-like
        需要绘制的样本数据。
    y : array-like
        每个样本数据对应的类别（标签）。

    """
    # 定义不同类别的颜色与符号。可以用于二分类与三分类。
    color = ["r", "g", "b"]
    marker = ["o", "v", "x"]
    # 获取数据中不重复的标签。
    class_label = np.unique(y)
    # 定义颜色图，在绘制等高线的时候使用，不同的值使用不同的颜色来绘制。
    cmap = ListedColormap(color[: len(class_label)])
    # 获取每个特征的最小值与最大值。
    x1_min, x2_min = np.min(X, axis=0)
    x1_max, x2_max = np.max(X, axis=0)
    # 定义每个特征的取值范围。
    x1 = np.arange(x1_min - 1, x1_max + 1, 0.02)
    x2 = np.arange(x2_min - 1, x2_max + 1, 0.02)
    # 对数组x1,x2进行扩展，获取二者的笛卡尔积组合，用于送入模型中，进行预测。
    X1, X2 = np.meshgrid(x1, x2)
    # 将之前两个特征的笛卡尔积组合送入模型中，预测结果。
    Z = model.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape)
    # 根据Z值的不同，绘制等高线（不同的值使用不同的颜色表示）。
    plt.contourf(X1, X2, Z, cmap=cmap, alpha=0.5)
    # 绘制样本数据X。
    for i, class_ in enumerate(class_label):
        plt.scatter(x=X[y == class_, 0], y=X[y == class_, 1],
                c=cmap.colors[i], label=class_, marker=marker[i])
    plt.legend()
```

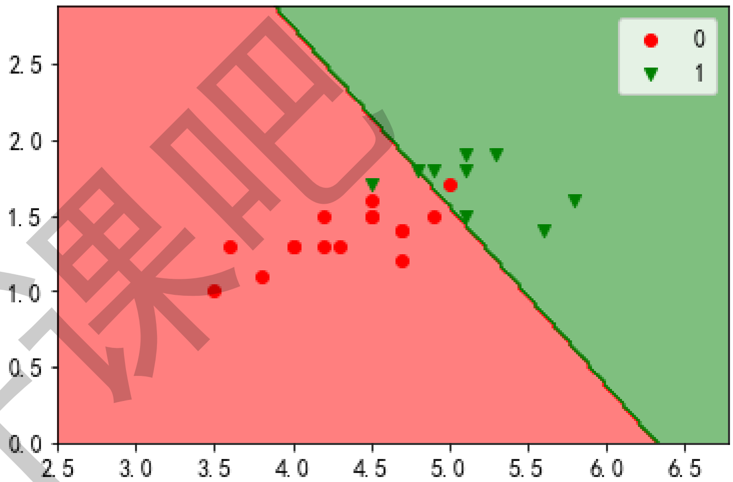
然后，就可以调用该函数绘制决策边界。我们绘制模型在训练集上的划分效果。

```
plot_decision_boundary(lr, X_train, y_train)
```



我们再来绘制模型在测试集上的划分效果。

```
plot_decision_boundary(lr, X_test, y_test)
```



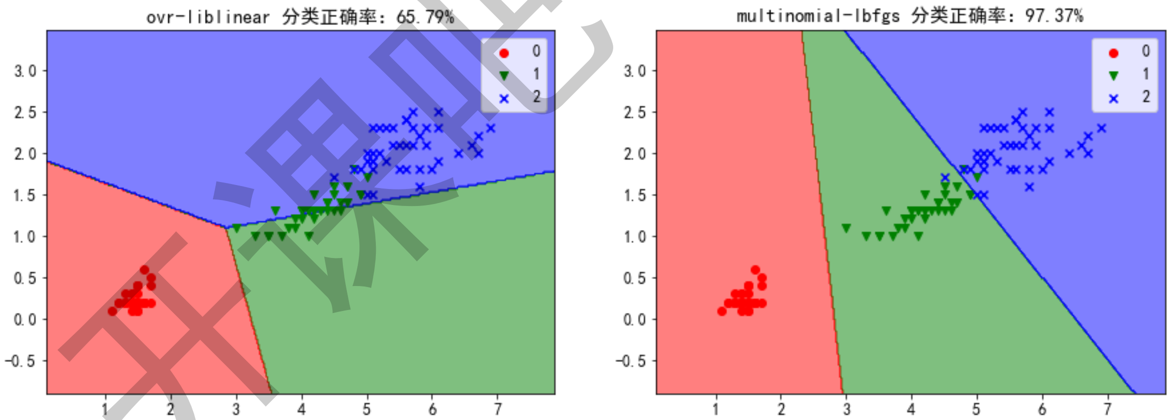
逻辑回归实现多分类

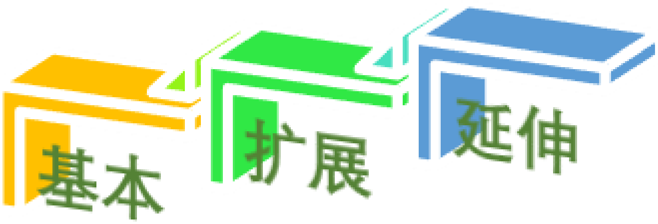
建模与可视化

逻辑回归算法也可以实现多分类任务。在不同的分类方式下，可能会得到不同的结果，我们可以通过决策边界来清晰的查看这一点。

```
iris = load_iris()
X, y = iris.data, iris.target
# 仅使用其中的两个特征。
X = X[:, 2:]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)
plt.figure(figsize=(15, 5))
# 定义不同的参数组合，用于循环。
params = [("ovr", "liblinear"), ("multinomial", "lbfgs")]
for index, (mc, s) in enumerate(params, start=1):
    lr = LogisticRegression(multi_class=mc, solver=s)
    lr.fit(X_train, y_train)
```

```python
    y_hat = lr.predict(X_test)
    plt.subplot(1, 2, index)
    # 计算分类正确率。
    accuracy = np.sum(y_test == y_hat) / len(y_test)
    plt.title(f"{mc}-{s} 分类正确率：{accuracy * 100:.2f}%")
    plot_decision_boundary(lr, X_train, y_train)
```





多分类实现细节（扩展）

我们刚才使用逻辑回归模型成功实现了多分类，不过，我们可能会有如下疑问：
在我们之前的讲解中，逻辑回归模型是依靠sigmoid函数的值来判别数据的类别的，判别的依据是：

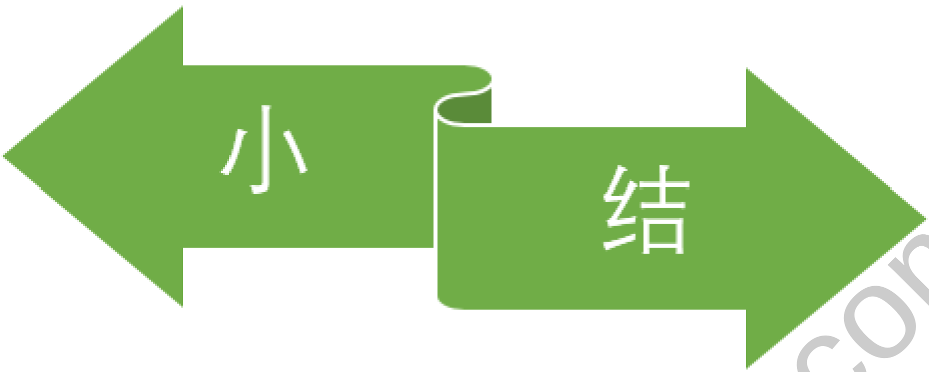
$$\hat{y} = \begin{cases} 1 & sigmoid(z) > 0.5 \\ 0 & sigmoid(z) \le 0.5 \end{cases}$$

这就是说，逻辑回归应该只能实现二分类，可是，刚才的程序中，也完全实现了多分类的效果，而且在程序中，并没有提现出明显的不同，那么，多分类是怎样做到的呢？

多分类在实现上，可以采用两种方式：

- one versus rest（一对其他）
- multinomial（多项式）

关于具体的实现细节与代码演示，老梁提供辅助视频，供大家学习。



拓展点

- 逻辑回归实现多分类的细节。

作业

1. 手动编写逻辑回归类，能够实现二分类任务，要求实现如下的功能：
    - 编写fit方法，能够训练模型。

- 在训练模型之后，模型具有coef与*intercept*属性，能够返回权重（w）与偏置（b）。
- 编写predict方法，能够对未知样本实现预测。
- 编写decision_function方法，用于计算z值（样本的得分值）。
- 编写predict_proba方法，用于计算样本属于每个类别的概率值。
- 不允许借助于scikit-learn中的逻辑回归类。

2. 对泰坦尼克号数据集上，使用逻辑回归进行分类，并预测乘客是否生还。
3. 在scikit-learn中，LogisticRegression为什么默认就带有正则化，而不是像LinearRegression那样，没有加入正则化?