第 3 章

逻 辑 回 归

在第 2 章中，笔者详细地介绍了线性回归模型，从本章开始将继续介绍下一个经典的机器学习算法——逻辑回归(Logistic Regression)。如图 3-1 所示，此图为逻辑回归模型学习的大致路线，其同样也分为 3 个阶段。在第 1 个阶段结束后，我们也就大致掌握了逻辑回归的基本原理。下面就开始正式进入逻辑回归模型的学习。

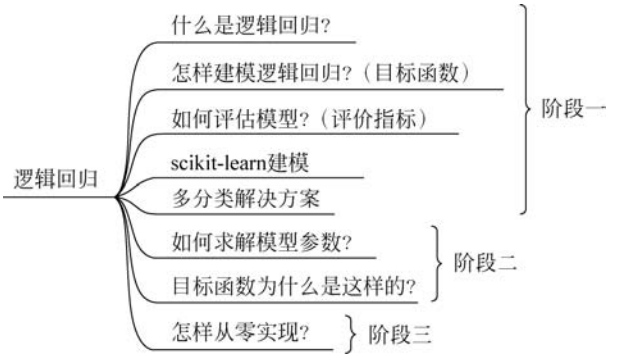

图 3-1　逻辑回归学习路线图

3.1　模型的建立与求解

3.1.1　理解逻辑回归模型

通常来讲，一个新算法的诞生要么用来改善已有的算法模型，要么就是首次提出用来解决一个新的问题，而逻辑回归模型恰恰属于后者，它是用来解决一类新的问题——分类(Classification)。什么是分类问题呢?

现在有两堆样本点，需要建立一个模型来对新输入的样本进行预测，判断其应该属于哪个类别，即二分类问题(Binary Classification)，如图 3-2 所示。对于这个问题的描述用线性回归来解决肯定是不行的，因为两者本就属于不同类型的问题。退一步讲，即使用线性回归来建模得到的估计也就是一条向右倾斜的直线，而我们这里需要的却是一条向左倾斜的且

位于两堆样本点之间的直线。同时，回归模型的预测值都位于预测曲线附近，而无法做到区分直线两边的东西。既然用已有的线性回归解决不了，那么我们可不可以在此基础上做一点改进以实现分类的目的呢？答案是当然可以。

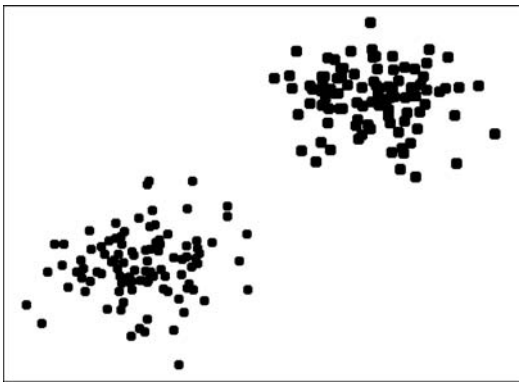

图 3-2　分类任务

3.1.2　建立逻辑回归模型

既然是解决分类问题，那么完全可以通过建立一个模型用来预测每个样本点属于其中一个类别的概率 p，如果 $p>0.5$，我们就可以认为该样本点属于这个类别，这样就能解决上述的二分类问题了。该怎样建立这个模型呢？

在前面的线性回归中，通过建模 $h(x)=wx+b$ 来对新样本进行预测，其输出值为可能的任意实数，但此处既然要得到一个样本所属类别的概率，那最直接的办法就是通过一个函数 $g(z)$，将 $h(x)$ 映射至[0,1]的范围即可。由此，便得到了逻辑回归中的预测模型

$$\hat{y}=h(x)=g(wx+b) \tag{3-1}$$

其中，w 和 b 为未知参数；$h(x)$ 称为假设函数(Hypothesis)。当 $h(x)$ 大于某个值(通常设为 0.5)时，便可以认为样本 x 属于正类，反之则认为属于负类。同时，也将 $wx+b=0$ 称为两个类别间的决策边界(Decision Boundary)。当求解得到 w 和 b 后，也就意味着得到了这个分类模型。

注意：回归模型一般来讲是指对连续值进行预测的一类模型，而分类模型则是指对离散值(类标)预测的一类模型，但是由于历史的原因虽然逻辑回归被称为回归，但它却是一个分类模型，这算是一个例外。

3.1.3　求解逻辑回归模型

当建立好模型之后就需要找到一种方法来求解模型中的未知参数。同线性回归一样，此时也需要通过一种间接的方式，即通过目标函数来刻画预测标签(Label)与真实标签之间

的差距。当最小化目标函数后，便可以得到需要求解的参数 w 和 b。

同样，笔者先给出逻辑回归中的目标函数(第二阶段再讲解来历)：

$$\begin{cases} J(w,b) = -\dfrac{1}{m}\left[\sum\limits_{i=1}^{m} y^{(i)} \log h(x^{(i)}) + (1-y^{(i)}) \log_2 (1-h(x^{(i)}))\right] \\ h(x^{(i)}) = g(wx^{(i)} + b) \end{cases} \tag{3-2}$$

其中，m 表示样本总数，$x^{(i)}$ 表示第 i 个样本，$y^{(i)}$ 表示第 i 个样本的真实标签，$h(x^{(i)})$ 表示第 i 个样本为正类的预测概率。

由式(3-2)可以知道，当函数 $J(w,b)$ 取得最小值的参数 $\hat{w}$ 和 $\hat{b}$，也就是我们要求的目标参数。原因在于，当 $J(w,b)$ 取得最小值时就意味着此时所有样本的预测标签与真实标签之间的差距最小，这同时也是最小化目标函数的意义，因此，对于如何求解模型 $h(x)$ 的问题就转化为如何最小化目标函数 $J(w,b)$ 的问题。

至此，对逻辑回归算法第一阶段核心内容的学习也就只差一步之遥了，也就是评价指标及通过开源的框架来建模并进行预测。

3.1.4 逻辑回归示例代码

首先，为了便于后续的可视化过程及了解分类的实质，笔者这里首先采用人为的方式来构造一个数据集并以此进行模型的训练，然后通过 sklearn 中的 LogisticRegression 来完成模型的求解。完整代码见 Book/Chapter03/01_decision_boundary.py 文件。

1. 构造数据集

这里需要用到 sklearn 中的 make_blobs()方法来构造数据集，代码如下：

```
from sklearn.datasets import make_blobs
def make_data():
    centers = [[1, 1], [2, 2]]                          #指定中心
    x, y = make_blobs(n_samples = 200, centers = centers,
                cluster_std = 0.2, random_state = np.random.seed(10))
    index_pos, index_neg = (y == 1), (y == 0)
    x_pos, x_neg = x[index_pos], x[index_neg]
```

在上述代码中，第 2 行用来指定生成两个样本堆的中心，第 3～4 行则是根据指定的中心点生成两个不同类别的样本堆，其中 n_samples 表示样本的数量，cluster_std 表示样本间的标准差(值越小，样本点分布就越集中)，random_state 表示用来指定一个固定的随机种子，以使每次产生相同的样本点。

在这之后，就能生成一个数据集了，如图 3-3 所示。

2. 训练模型

接着通过 LogisticRegression 来完成模型的训练和预测，代码如下：

```
from sklearn.linear_model import LogisticRegression
def decision_boundary(x, y):
```

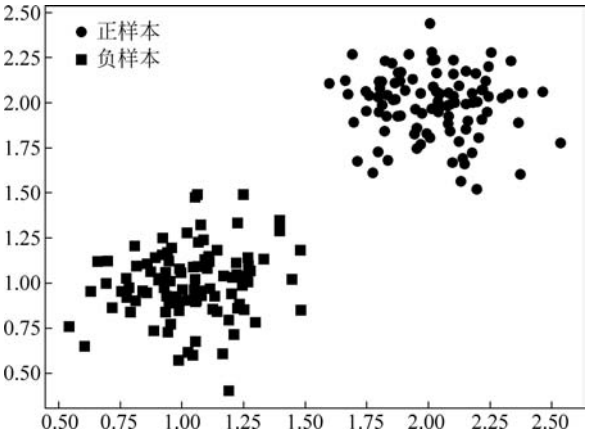

图 3-3　二分类数据集可视化

```
model = LogisticRegression()
model.fit(x, y)
pred = model.predict([[1, 0.5], [3, 1.5]])
print("样本点(1,0.5)所属的类标为{}\n"
      "样本点(3,1.5)所属的类标为{}".format(pred[0], pred[1]))
```

在完成模型的训练之后，便可以绘制出模型所训练得到的决策面，用于样本点的分类，如图 3-4 所示。

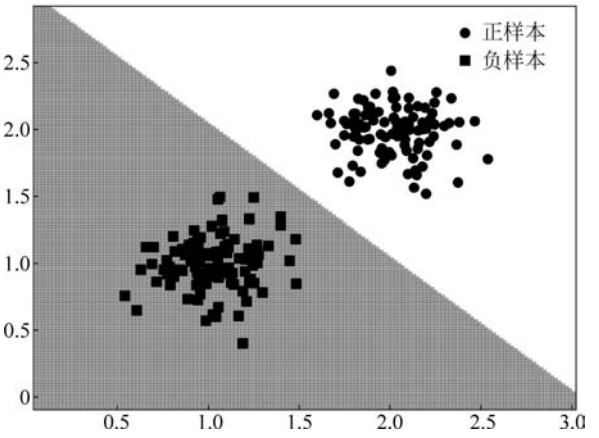

图 3-4　二分类决策边界

3.1.5　小结

在本节中，笔者首先通过一个例子引入了什么是分类，然后介绍了为什么不能用线性回归模型进行建模的原因。其次，通过对线性回归的改进得到逻辑回归模型，并直接地给出了

逻辑回归模型的目标函数。最后通过开源的 sklearn 框架搭建了一个简单的逻辑回归模型，并对决策面进行了可视化。虽然内容不多，也不复杂但却包含了逻辑回归算法的核心思想。同时，余下的内容也会在后续的章节中进行介绍。

3.2 多变量与多分类

3.2.1 多变量逻辑回归

如同多变量线性回归一样，所谓多变量逻辑回归其实就是一个样本点有多个特征属性，然后通过建立一个多变量的逻辑回归模型来完成分类任务。实际上在现实情况中，几乎没有一个模型是单变量的，即每个样本点都有多个特征。由于这是我们学习的第 2 个机器学习算法，所以在这里再对多变量进行一次说明。在后续的算法介绍中，将不再单独提及“多变量模型”这一叫法，所有模型的输入都是一个向量(Vector)。

$$\hat{y} = h(x) = g(w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b) \tag{3-3}$$

如式(3-3)所示，此函数为一个多变量逻辑回归模型的假设函数，其后续的所有步骤(求解、预测等)并没有任何的变化与不同，仅仅有了更多的特征属性。

3.2.2 多分类逻辑回归

在 3.1 节中对于逻辑回归的介绍都仅仅局限在二分类任务中，但是在实际任务里，更多则是多分类的任务场景，也就是说最终的分类结果中类别数会大于 2。对于这样的问题该如何解决呢？

通常情况下在用逻辑回归处理多分类任务时，都会采样一种称为 One-vs-all(也叫作 One-vs-rest)的方法，两者的缩写分别为 ova 与 ovr。这种策略的核心思想就是每次将其中一个类和剩余的其他类看作一个二分类任务进行训练，最后在预测过程中选择输出概率值最大那个类作为该样本点所属的类别。

如图 3-5 所示，此图为一个可视化的数据集，它一共包含 3 个类别。

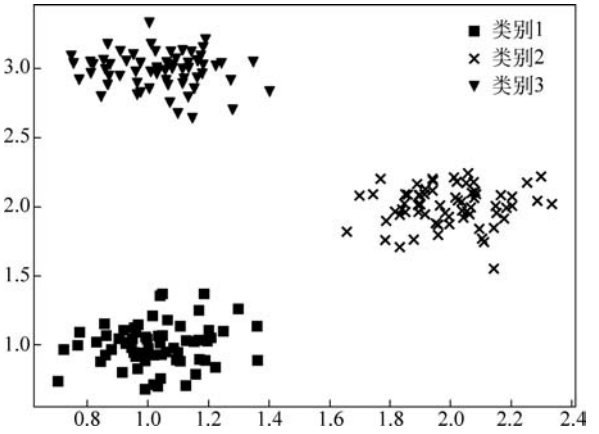

图 3-5 多分类问题

当利用 One-vs-all 的分类思想来解决图 3-5 中的多分类问题时，可以可视化成如图 3-6 所示的情况。

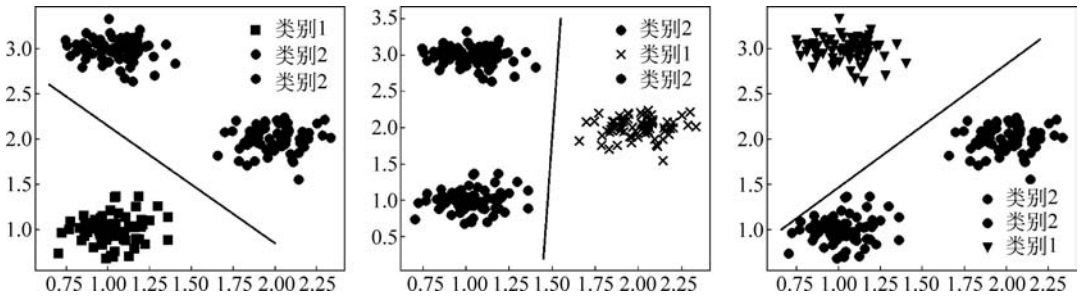

图 3-6　One-vs-all 思想

在图 3-6 中，以从左往右的划分方式划分数据集，然后分别训练 3 个二分类的逻辑回归模型 $h_1(x)$、$h_2(x)$和 $h_3(x)$，分别表示样本 x 属于第 1、第 2 和第 3 共 3 个类别的概率，最后在预测的时候只要选择概率最大时分类模型所对应的类别即可。

3.2.3　多分类示例代码

在 sklearn 中，可以借助 LogisticRegression 类中的 multi_class＝'ovr'参数来完成整个多分类的建模任务，完整代码见 Book/Chapter03/02_one_vs_all_train.py 文件。

1. 载入数据集

在这里，笔者同样使用了 sklearn 中内置的一个分类数据集 iris 进行示例。首先需要载入这个数据集，代码如下：

```
from sklearn.datasets import load_iris
def load_data():
    data = load_iris()
    x, y = data.data, data.target
    return x, y
```

iris 数据集一共包含 3 个类别，每个类别中有 50 个样本，并且每个样本有 4 个特征维度。同时，sklearn 中也内置了很多丰富的其他数据集来方便初学者使用，具体信息可以参见官网[①]。

2. 训练模型

在数据集载入完成后，便可以通过 sklearn 中的 LogisticRegression 完成整个建模求解过程，代码如下：

① https://scikit-learn.org/stable.

```
def train(x, y):
    model = LogisticRegression(multi_class = 'ovr')
    model.fit(x,y)
    print("得分: ", model.score(x, y))
#得分:0.95
```

到此,对于多变量逻辑回归的分类方法与建模过程就介绍完了。不过细心的读者可能会发现,上面代码中的最后一行输出了一个 0.95 的得分,它表示什么含义呢? 这里的 0.95 其实指的模型分类的准确率,意思是有 95%的样本被模型正确分类了,具体计算原理可见 3.3 节内容。

3.2.4 小结

在本节内容中,笔者首先介绍了什么是多变量逻辑回归,同时还提到对于“多变量”这个说法在以后均不会刻意提及,因为在现实中几乎不存在一个只包含一个变量的任务场景。其次,笔者还以图示的方式介绍了如何用 One-vs-all 的思想来用逻辑回归模型解决多分类的任务场景。最后,借助开源库 sklearn 也完成了整个建模过程的示例。接下来,我们将开始学习分类模型中的常见评估指标。

3.3 常见的分类评估指标

如同回归模型一样,对于任何分类模型来讲同样需要通过一些评价指标来衡量模型的优与劣。在分类任务中,常见的评价指标有准确率(Accuracy)、精确率(Precision)、召回率(Recall)与 F 值(F_{score}),其中应用最为广泛的是准确率,接着是召回率。为了能够使读者更容易地理解与运用这 4 种评价指标,下面笔者将会由浅入深地从二分类到多分类的场景来对这 4 种指标进行介绍。

3.3.1 二分类场景

首先以一个猫狗图片识别的任务场景为例,假设现在有一个猫狗图片分类器对 100 张图片进行分类,分类结果显示有 38 张图片是猫,62 张图片是狗。经过与真实标签对比后发现,38 张猫的图片中有 20 张是分类正确的,62 张狗的图片中有 57 张是分类正确的。

根据上述这一情景,便可以得到一张如图 3-7 所示的矩阵,称为混淆矩阵(Confusion Matrix)。

真实	预测	
	猫	狗
猫	20	5
狗	18	57

真实	预测	
	P	N
P	TP	FN
N	FP	TN

图 3-7 二分类混淆矩阵

如何来读这个混淆矩阵呢？读的时候首先横向看，然后纵向看。例如读 TP 的时候，首先横向表示真实的正样本，其次是纵向表示预测的正样本，因此 TP 表示的就是将正样本预测为正样本的个数，即预测正确，因此，同理共有以下 4 种情况。

(1) True Positive(TP)：表示将正样本预测为正样本，即预测正确。

(2) False Negative(FN)：表示将正样本预测为负样本，即预测错误。

(3) False Positive(FP)：表示将负样本预测为正样本，即预测错误。

(4) True Negative(TN)：表示将负样本预测为负样本，即预测正确。

如果此时突然问 FP 表示什么含义，又该怎样迅速地反映出来呢？我们知道 FP(False Positive)从字面意思来看表示的是错误的正类，也就是说实际上它并不是正类，而是错误的正类，即实际上为负类，因此，FP 表示的就是将负样本预测为正样本的含义。再看一个 FN，其字面意思为错误的负类，也就是说实际上它表示的是正类，因此 FN 的含义就是将正样本预测为负样本。

定义完上述 4 个类别的分类情况后就能定义出各种场景下的计算指标，如式(3-4)～式(3-7)所示。

$$\text{Accuracy}=\frac{\text{TP}+\text{TN}}{\text{TP}+\text{FP}+\text{FN}+\text{TN}} \tag{3-4}$$

$$\text{Precision}=\frac{\text{TP}}{\text{TP}+\text{FP}} \tag{3-5}$$

$$\text{Recall}=\frac{\text{TP}}{\text{TP}+\text{FN}} \tag{3-6}$$

$$F_{\text{score}}=(1+\beta^2)\frac{\text{Precision}\cdot\text{Recall}}{\beta^2\cdot\text{Precision}+\text{Recall}} \tag{3-7}$$

注意：当 F_{score} 中 $\beta=1$ 时称为 F_1 值，同时 F_1 也是用得最多的 F_{score} 评价指标。

可以看出准确率是最容易理解的，即所有预测对的数量，除以总的数量。同时还可以看到，精确率计算的是预测对的正样本在整个预测为正样本中的比重，而召回率计算的是预测对的正样本在整个真实正样本中的比重，因此一般来讲，召回率越高也就意味着这个模型寻找正样本的能力越强(例如在判断是否为癌细胞的时候，寻找正样本癌细胞的能力就十分重要)，而 F_{score} 则是精确率与召回率的调和平均，但值得注意的是，通常在绝大多数任务中并不会明确哪一类别是正样本，哪一类别又是负样本，所以对于每个类别来讲都可以计算其各项指标，但是准确率只有一个。

在得到式(3-4)～式(3-7)中各项评价指标的计算公式后，便可以分别计算出 3.3.1 节一开始的示例场景中，猫狗分类模型的各项评估值。

1. 准确率

$$\text{Accuracy}=\frac{20+57}{20+18+5+57}=0.77$$

2. F 值

对于类别猫来讲，有

$$\text{Precision}=\frac{20}{20+18}=0.53$$

$$\text{Recall}=\frac{20}{20+5}=0.8$$

$$F_1=\frac{2\times 0.53\times 0.8}{0.53+0.8}=0.63$$

对于类别狗来讲，有

$$\text{Precision}=\frac{57}{57+5}=0.92$$

$$\text{Recall}=\frac{57}{57+18}=0.76$$

$$F_1=\frac{2\times 0.92\times 0.76}{0.92+0.76}=0.83$$

到这里，对于 4 种指标各自的原理及计算方式已经介绍完了，但是如果要来衡量整体的精确率、召回率或者 F 值又该怎么处理呢？对于分类结果整体的评估值，常见的做法有两种：第一种是取算术平均；第二种是加权平均[①]。

1. 算术平均

所谓算术平均也叫作宏平均(Macro Average)，也就是等权重地对各类别的评估值进行累加求和。例如对于上述两个类别来讲，其精确率、召回率和 F_1 值分别为

$$\text{Precision}=\frac{1}{2}\times 0.53+\frac{1}{2}\times 0.92=0.725$$

$$\text{Recall}=\frac{1}{2}\times 0.8+\frac{1}{2}\times 0.76=0.78$$

$$F_1=\frac{1}{2}\times 0.63+\frac{1}{2}\times 0.83=0.73$$

2. 加权平均

所谓加权平均也就是以不同的加权方式来对各类别的评估值进行累加求和。这里只介绍一种用得最多的加权方式，即按照各类别样本数在总样本中的占比进行加权。对于图 3-7 中的分类结果来讲，加权后的精确率、召回率和 F_1 值分别为

$$\text{Precision}=\frac{25}{100}\times 0.53+\frac{75}{100}\times 0.92=0.82$$

$$\text{Recall}=\frac{25}{100}\times 0.8+\frac{75}{100}\times 0.76=0.77$$

$$F_1=\frac{25}{100}\times 0.63+\frac{75}{100}\times 0.83=0.78$$

① PEDREGOSA. scikit-learn: Machine Learning in Python[J]. JMLR 12,2011: 2825-2830.

3.3.2 二分类指标示例代码

在弄清分类任务中各项指标的计算原理后，就可以选择其中的一些指标对模型的精度进行评估。从式(3-4)～式(3-7)可知，计算各项评估指标的关键就在于如何从分类结果中构造一个混淆矩阵。对于评估矩阵的构造，这里可以借助 sklearn 中提供的 confusion_matrix 方法进行实现。完整代码见 Book/Chapter03/03_confusion_matrix.py 文件。

1. 载入数据集

在这里，同样也使用 sklearn 中内置的一个分类数据集 breast_cancer 进行示例。首先需要载入这个数据集，代码如下：

```
from sklearn.datasets import load_breast_cancer
def load_data():
    data = load_breast_cancer()
    x, y = data.data, data.target
    return x, y
```

breast_cancer 数据集一共包含 2 个类别(正样本与负样本)，其中负样本 212 个，正样本 357 个，并且每个样本有 30 个特征维度。

2. 指标计算

根据前面介绍的计算原理，可以实现各项指标的计算过程，代码如下：

```
def get_acc_rec_pre_f(y_true, y_pred, beta=1.0):
    (tn, fp), (fn, tp) = confusion_matrix(y_true, y_pred)
    p1, p2 = tp / (tp + fp), tn / (tn + fn)
    r1, r2 = tp / (tp + fn), tn / (tn + fp)
    f_beta1 = (1 + beta ** 2) * p1 * r1 / (beta ** 2 * p1 + r1)
    f_beta2 = (1 + beta ** 2) * p2 * r2 / (beta ** 2 * p2 + r2)
    m_p,m_r, m_f = 0.5 * (p1+p2),0.5 * (r1+r2),0.5 * (f_beta1 + f_beta2)
    count = np.bincount(y_true)
    w1, w2 = count[1]/sum(count), count[0]/sum(count)                    #计算加权平均
    w_p, w_r,w_f=w1 * p1+w2 * p2, w1 * r1+w2 * r2, w1 * f_beta1+w2 * f_beta2
    print(f"算术平均:精确率为{m_p},召回率为{m_r},F值为{m_f}")
    print(f"加权平均:精确率为{w_p},召回率为{w_r},F值为{w_f}")
```

在上述代码中，第 2 行用来构造混淆矩阵，第 3～4 行分别用来计算两种类别各自的精确率和召回率，第 5～6 行分别用来计算两种类别的 F 值，第 7 行用来计算各个指标的算术平均，第 8～10 行用来计算各个指标的加权平均。

3. 训练模型

在完成上面两个步骤后，便可以通过 LogisticRegression 来完成整个建模的求解过程，并输出对应的评价指标，代码如下：

```
def train(x, y):
    model = LogisticRegression()
    model.fit(x, y)
    y_pred = model.predict(x)
    print("准确率: ", model.score(x, y))
    get_acc_rec_pre_f(y, y_pred)
```

运行上述代码后，便能够得到如下所示的评估结果，如下：

```
准确率: 0.95
算术平均: 精确率为 0.95,召回率为 0.94,F 值为 0.94
加权平均: 精确率为 0.95,召回率为 0.95,F 值为 0.95
```

3.3.3 多分类场景

在 3.3.1 节中，笔者详细地介绍了在二分类场景下各种评价指标的计算方法，但是在现实场景情况里更多的场景便是多分类场景，并且通常也会采用召回率、精确率或者 F 值作为评价指标。此时各个指标又该怎么计算呢？在接下来的这节内容中，笔者将会针对多分类的任务场景来介绍这些指标的计算方法。

假设有以下三分类任务的预测值与真实值，代码如下：

```
y_true = [1, 1, 1, 0, 0, 0, 2, 2, 2, 2]
y_pred = [1, 0, 0, 0, 2, 1, 0, 0, 2, 2]
```

根据这一结果，便可以得到一个混淆矩阵，如图 3-8 所示。

真实	预测		
	0	1	2
0	1	1	1
1	2	1	0
2	2	0	2

图 3-8 多分类混淆矩阵

如图 3-8 所示，由于是多分类，所以也就不止正样本和负样本两个类别，此时这个表该怎么读呢？方法还是同 3.3.1 节中的一样，先横向看，再纵向看。例如第 1 行灰色单元格中的 1 表示的就是将真实值 0 预测为 0 的个数(预测正确)，接着右边的 1 表示的就是将真实值 0 预测为 1 的个数，第 2 行灰色单元格中的 1 表示的就是将真实值 1 预测为 1 的个数，第 3 行灰色单元格中的 2 表示的就是将真实值 2 预测为 2 的个数。也就是说只有这个对角线上的值才表示模型预测正确的样本的数量。接下来开始对每个类别的各项指标进行计算。

1. 对于类别 0 来讲

在上面笔者介绍过，精确率计算的是预测对的正样本在整个预测为正样本中的比重。根据图 3-8 可知，对于类别 0 来讲，预测对的正样本(类别 0)的数量为 1，而整个预测为正样本的数量为 5，因此，类别 0 对应的精确率为

$$\text{Precision}=\frac{1}{1+2+2}=0.2$$

同时，召回率计算的是预测对的正样本在整个真实正样本中的比重。根据图 3-8 可知，对于类别 0 来讲，预测对的正样本(类别 0)的数量为 1，而整个真实正样本 0 的个数为 3(图 3-8 中第 2 行的 3 个 1)，因此，对于类别 0 来讲其召回率为

$$\text{Recall}=\frac{1}{1+1+1}=0.33$$

因此，其 F_1 值为

$$F_1=\frac{2\times 0.2\times 0.33}{0.2+0.33}=0.25$$

2. 对于类别 1 来讲

对于类别 1 来讲，预测对的正样本(类别 1)的数量为 1，而整个预测为类别 1 的样本数量为 2，因此，其精确率为

$$\text{Precision}=\frac{1}{1+1+0}=0.5$$

同理，其召回率和 F_1 值分别为

$$\text{Recall}=\frac{1}{1+2}=0.33$$

$$F_1=\frac{2\times 0.5\times 0.33}{0.5+0.33}=0.40$$

3. 对于类别 2 来讲

$$\text{Precision}=\frac{2}{1+0+2}=0.67$$

$$\text{Recall}=\frac{2}{2+0+2}=0.50$$

$$F_1=\frac{2\times 0.67\times 0.50}{0.67+0.50}=0.57$$

最后，对于 3 个类别来讲其加权后的准确率、召回率和 F_1 值分别为

1. 算术平均

$$\text{Precision}=\frac{1}{3}\times(0.20+0.50+0.67)=0.46$$

$$\text{Recall}=\frac{1}{3}\times(0.33+0.33+0.50)=0.39$$

$$F_1=\frac{1}{3}\times(0.25+0.40+0.57)=0.41$$

2. 加权平均

$$\text{Precision}=\frac{3}{10}\times 0.2+\frac{3}{10}\times 0.50+\frac{4}{10}\times 0.67=0.48$$

$$\text{Recall}=\frac{3}{10}\times 0.33+\frac{3}{10}\times 0.33+\frac{4}{10}\times 0.50=0.40$$

$$F_1=\frac{3}{10}\times 0.25+\frac{3}{10}\times 0.40+\frac{4}{10}\times 0.57=0.42$$

3.3.4 多分类指标示例代码

对于这部分代码的实现，从上面的计算过程可以发现，最重要的是要计算并得到这个混淆矩阵，但是对于多分类任务来讲，后续实现代码也略显繁杂，不过各位读者可以仿照 3.3.2 节中的代码自己编码实现。不过这里可以直接借助 sklearn 中的 classification_report 模块完成所有的计算过程，代码如下：

```
from sklearn.metrics import classification_report
y_true = [1, 1, 1, 0, 0, 0, 2, 2, 2, 2]
y_pred = [1, 0, 0, 0, 2, 1, 0, 0, 2, 2]
print(classification_report(y_true, y_pred))
```

在运行上述代码后，便可以得到如下所示的结果：

```
                precision    recall  f1-score   support      #分别表示精确率、召回率、F值

           0       0.20      0.33      0.25         3
           1       0.50      0.33      0.40         3
           2       0.67      0.50      0.57         4

    accuracy                           0.40        10      #模型准确率
   macro avg       0.46      0.39      0.41        10      #算术平均(宏平均)
weighted avg       0.48      0.40      0.42        10      #加权平均
```

3.3.5 小结

在本节中，笔者首先介绍了二分类任务场景下混淆矩阵的构造及对应的含义，接着介绍了如何通过混淆矩阵来计算分类模型中的各项评估指标，然后介绍了在多分类任务下混淆矩阵的构造及对应各项指标的计算方法，最后还通过 sklearn 中的 confusion_matrix 和 classification_report 模块来示例完成了所有的计算过程。接下来让我们开始学习逻辑回归模型中最后两个阶段的内容。

3.4 目标函数推导

在前面3节的内容中，笔者详细地介绍了什么是逻辑回归、如何进行多分类及分类任务对应的评价指标等，即完成了前面阶段一的学习，但是到目前为止仍旧有一些问题没有解决，映射函数 $g(z)$是什么样的？逻辑回归的目标函数是怎么来的？如何自己求解并实现逻辑回归？只有在这3个问题得到解决后，整个逻辑回归算法的主要内容才算学习完了。

3.4.1 映射函数

前面笔者只是介绍了通过一个函数 $g(z)$将特征的线性组合 $z=Wx+b$ 映射到区间[0,1]，那么这个 $g(z)$是什么样的呢？如图3-9所示，这便是 $g(z)$的函数图形，其也被称为Sigmoid()函数。

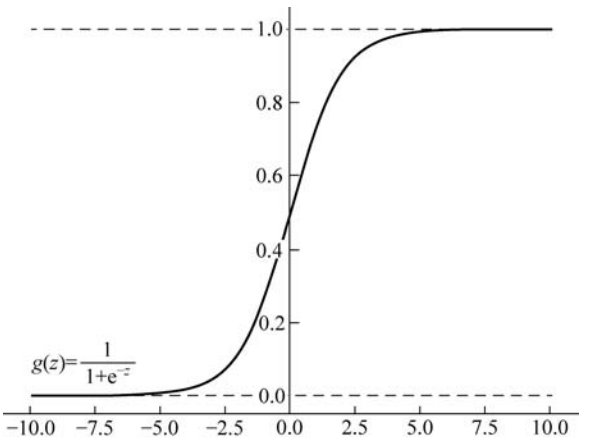

图3-9 Sigmoid()函数图形

Sigmoid()函数的数学定义如下

$$g(z)=\frac{1}{1+e^{-z}} \tag{3-8}$$

其中 $z\in[-\infty,+\infty]$，而之所以选择Sigmoid()的原因在于：①其连续光滑且处处可导；②Sigmoid()函数关于点(0,0.5)中心对称；③Sigmoid()函数的求导过程简单，其最后的求导结果为 $g'(z)=g(z)(1-g(z))$。

根据式(3-8)可以得出其实现代码如下：

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

可以看到对于Sigmoid()的实现也非常简单，1行代码就能完成。

3.4.2 概率表示

在介绍完 Sigmoid()函数后就需要弄清楚逻辑回归中的目标函数到底是怎么得来的。此时,可以设

$$\begin{cases} P(y=1 \mid x;\boldsymbol{W},b)=h(\boldsymbol{x}) \\ P(y=0 \mid x;\boldsymbol{W},b)=1-h(\boldsymbol{x}) \\ h(\boldsymbol{x})=g(z)=g(\boldsymbol{W}^{\mathrm{T}}\boldsymbol{x}+b) \end{cases} \tag{3-9}$$

其中 $\boldsymbol{W}$ 和 $\boldsymbol{x}$ 均为一个列向量,$P(y=1|\boldsymbol{x};\boldsymbol{W},b)=h(\boldsymbol{x})$的含义为当给定参数 $\boldsymbol{W}$ 和 b 时,样本 $\boldsymbol{x}$ 属于 $y=1$ 这个类别的概率为 $h(\boldsymbol{x})$。此时可以发现,对于每个样本来讲都需要前面两个等式来衡量每个样本所属类别的概率,为了更加方便地表示每个样本所属类别的概率,可以改写为如下形式:

$$p(y \mid \boldsymbol{x};\boldsymbol{W},b)=(h(\boldsymbol{x}))^{y}(1-h(\boldsymbol{x}))^{1-y} \tag{3-10}$$

这样一来,不管样本 $\boldsymbol{x}$ 属于哪个类别,都可以通过式(3-10)进行概率计算。

进一步,我们知道在机器学习中是通过给定训练集,即$(\boldsymbol{x}^{(i)},y^{(i)})$来求得其中的未知参数 $\boldsymbol{W}$ 和 b。换句话说,对于每个给定的 $\boldsymbol{x}^{(i)}$,我们已经知道了其所属的类别 $y^{(i)}$,即 $y^{(i)}$的这样一个分布结果我们是知道的。那么什么样的参数 $\boldsymbol{W}$ 和 b 能够使已知的 $y^{(1)},y^{(2)},\cdots,y^{(m)}$ 这样一个结果(分布)最容易出现呢?也就是说给定什么样的参数 $\boldsymbol{W}$ 和 b,使当输入 $\boldsymbol{x}^{(1)}$,$\boldsymbol{x}^{(2)},\cdots,\boldsymbol{x}^{(m)}$ 这 m 个样本时,最能够产生已知类别标签 $y^{(1)},y^{(2)},\cdots,y^{(m)}$ 这一结果呢?

3.4.3 极大似然估计

上面绕来绕去说了这么多,其目的只有一个,即为什么要用似然函数进行下一步计算。由 3.4.2 节内容分析可知,为了能够使 $y^{(1)},y^{(2)},\cdots,y^{(m)}$ 这样一个结果最容易出现,应该最大化如下似然函数①

$$L(\boldsymbol{W},b)=\prod_{i=1}^{m}p(y^{(i)} \mid \boldsymbol{x}^{(i)};\boldsymbol{W},b)=\prod_{i=1}^{m}(h(\boldsymbol{x}^{(i)}))^{y^{(i)}}(1-h(\boldsymbol{x}))^{1-y^{(i)}} \tag{3-11}$$

对式(3-11)两边同时取自然对数有

$$\ell(\boldsymbol{W},b)=\log L(\boldsymbol{W},b)=\sum_{i=1}^{m}[y^{(i)}\log h(\boldsymbol{x}^{(i)})+(1-y^{(i)})\log(1-h(\boldsymbol{x}^{(i)}))] \tag{3-12}$$

注意:$\log a^{b}c^{d}=\log a^{b}+\log c^{d}=b\log a+d\log c$

由于我们的目标是最大化式(3-11),也就等价于最大化式(3-12),因此当式(3-12)取得最大值时,其所对应的参数 $\boldsymbol{W}$ 和 b 就是逻辑回归模型所需要求解的参数。由此便得到了逻辑回归算法的目标函数

① Andrew Ng,Machine Learning,Stanford University,CS229,Spring 2019.

$$J(\boldsymbol{W},b)=-\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log h(\boldsymbol{x}^{(i)})+(1-y^{(i)})\log(1-h(\boldsymbol{x}^{(i)}))\right] \tag{3-13}$$

从式(3-13)可以发现我们在前面加了一个负号，因此求解逻辑回归的最终目的就变成了最小化式(3-13)。

3.4.4　求解梯度

在求解线性回归中，笔者首次引入并讲解了梯度下降算法，知道可以通过梯度下降算法来最小化某个目标函数。当目标函数取得(或接近)其函数最小值时，我们便得到了目标函数中对应的未知参数。由此可知，欲通过梯度下降算法来最小化函数式(3-13)，则必须先计算并得到其关于各个参数的梯度，所以接下来就需要求解并得到目标函数关于各个参数的梯度。

目标函数 $J(\boldsymbol{W},b)$ 对 W_j 的梯度为

$$\begin{aligned}\frac{\partial J}{\partial W_j}&=-\frac{\partial}{\partial W_j}\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log h(\boldsymbol{x}^{(i)})+(1-y^{(i)})\log(1-h(\boldsymbol{x}^{(i)}))\right]\\&=-\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\frac{h'(\boldsymbol{x}^{(i)})}{h(\boldsymbol{x}^{(i)})}+(1-y^{(i)})\frac{-h'(\boldsymbol{x}^{(i)})}{1-h(\boldsymbol{x}^{(i)})}\right]\\&=-\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\frac{g(z^{(i)})(1-g(z^{(i)}))}{g(z^{(i)})}x_j^{(i)}-(1-y^{(i)})\frac{g(z^{(i)})(1-g(z^{(i)}))}{1-g(z^{(i)})}x_j^{(i)}\right]\\&=-\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}(1-g(z^{(i)}))-(1-y^{(i)})g(z^{(i)})\right]x_j^{(i)}\\&=-\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}-h(\boldsymbol{x}^{(i)})\right]x_j^{(i)}\end{aligned} \tag{3-14}$$

目标函数 $J(\boldsymbol{W},b)$ 对 b 的梯度为

$$\begin{aligned}\frac{\partial J}{\partial b}&=-\frac{\partial}{\partial b}\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log h(\boldsymbol{x}^{(i)})+(1-y^{(i)})\log(1-h(\boldsymbol{x}^{(i)}))\right]\\&=-\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\frac{h'(\boldsymbol{x}^{(i)})}{h(\boldsymbol{x}^{(i)})}+(1-y^{(i)})\frac{-h'(\boldsymbol{x}^{(i)})}{1-h(\boldsymbol{x}^{(i)})}\right]\\&=-\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\frac{g(z^{(i)})(1-g(z^{(i)}))}{g(z^{(i)})}-(1-y^{(i)})\frac{g(z^{(i)})(1-g(z^{(i)}))}{1-g(z^{(i)})}\right]\\&=-\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}(1-g(z^{(i)}))-(1-y^{(i)})g(z^{(i)})\right]\\&=-\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}-h(\boldsymbol{x}^{(i)})\right]\end{aligned} \tag{3-15}$$

进一步，对式(3-13)、式(3-14)、式(3-15)矢量化可得

```
J(W,b) = -1/m * np.sum(y * np.log(h_x) + (1 - y) * np.log(1 - h_x))
grad_w = (1 / m) * np.matmul(X.T, (h_x - y)) #[n,m] @ [m,1]
grad_b = (1 / m) * np.sum(h_x - y)
```

在求得各个参数的梯度计算公式后，便可以通过 Python 自己实现整个逻辑回归的建模与求解过程。

3.4.5 从零实现二分类逻辑回归

这里依旧以 breast_cancer 这个二分类数据集为例来建立逻辑回归模型，完整代码见 Book/Chapter03/04_implementation.py 文件。

1. 预测函数

在实现整个逻辑回归的建模与求解过程前，首先需要完成假设函数和预测函数的编码，代码如下：

```
def hypothesis(X, W, bias):
    z = np.matmul(X, W) + bias
    h_x = sigmoid(z)
    return h_x
```

如上代码便是假设函数的实现，其中第 2 行代码是样本和权重的线性组合，第 3 行代码通过映射函数将线性组合后的结果映射到[0,1]的概率值。接着便是根据一个阈值(这里默认设置为 0.5，也可以设置为其他值)将概率值转化为具体对应的类别，代码如下：

```
def prediction(X, W, bias, thre=0.5):
    h_x = hypothesis(X, W, bias)
    y_pre = (h_x>thre) * 1        #将大于阈值的 True 和 False 结果转换为 1 和 0 的标签结果
    return y_pre
```

2. 目标函数

为了更好地观察目标函数的收敛，同样需要计算每次参数更新后的损失值，具体实现代码如下：

```
def cost_function(X, y, W, bias):
    m, n = X.shape
    h_x = hypothesis(X, W, bias)
    cost = np.sum(y * np.log(h_x) + (1 - y) * np.log(1 - h_x))
    return -cost / m
```

3. 梯度下降

为了对整个模型进行求解，因此需要编程实现整个梯度下降的计算过程，代码如下：

```
def gradient_descent(X, y, W, bias, alpha):
    m, n = X.shape
    h_x = hypothesis(X, W, bias)
    grad_w = (1 / m) * np.matmul(X.T, (h_x - y)) #[n,m] @ [m,1]
    grad_b = (1 / m) * np.sum(h_x - y)
    W = W - alpha * grad_w                #梯度下降
```

```
    bias = bias - alpha * grad_b            #梯度下降
    return W, bias
```

在上述代码中，第 6～7 行用来执行梯度下降的过程。

4. 训练模型

在完成上述各部分函数的实现后便可以实现整个模型的训练过程，代码如下：

```
def train(X, y, ite = 200):
    m, n = X.shape #506,13
    W = np.random.uniform( - 0.1, 0.1, n).reshape(n, 1)
    b,alpha, costs = 0.1, 0.08, []
    for i in range(ite):
        costs.append(cost_function(X, y, W, b))
        W, b = gradient_descent(X, y, W, b, alpha)
    y_pre = prediction(X, W, b)
    return costs
```

在上述代码中，第 2 行用来随机初始化权重参数，然后通过梯度下降进行迭代更新，第 6 行用来保存参数在每一次迭代更新后计算出来的损失值，并进行了返回。最后，该模型经过 200 次迭代后，准确率能够达到 0.98 左右。

根据返回后的损失值，还能画出整个训练过程中模型的收敛情况，如图 3-10 所示。

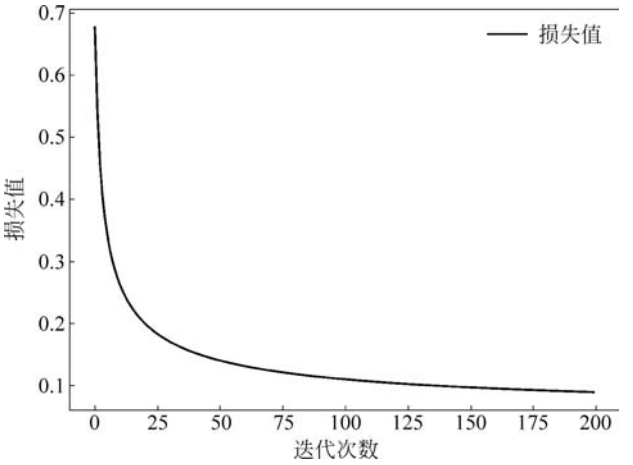

图 3-10　逻辑回归损失函数收敛图形

从图 3-10 可以看出，模型大概大约在第 100 次迭代后就慢慢进入了收敛阶段。

3.4.6　从零实现多分类逻辑回归

在 3.4.5 节内容中，笔者从零开始实现了整个二分类逻辑回归的建模与求解过程。在接下来的这一小节中，笔者将继续介绍如何从零开始实现一个多分类的逻辑回归模型。完

整代码见 Book/Chapter03/05_implementation_multi_class.py 文件。

1. 载入数据集

由于是多分类任务，所以这里使用的是 sklearn 中的一个 3 分类数据集 iris，具体信息在 3.2.6 节中已经介绍过，这里就不再赘述了，载入的代码如下：

```
from sklearn.datasets import load_iris
def load_data():
    data = load_iris()
    x, y = data.data, data.target
    return x, y
```

2. 定义二分类器

根据 3.2.2 节的内容可知，One-vs-all 的本质就是训练多个二分类器，然后用每个二分类器来对样本进行分类，因此这里需要定义一个函数来完成二分类器的训练，最后通过多次调用这个函数实现多个二分类模型的训练，代码如下：

```
def train_binary(X, y, iter=200):
    m, n = X.shape #506,13
    W = np.random.randn(n, 1)
    b,alpha,costs = 0.3,0.5,[]
    for i in range(iter):
        costs.append(cost_function(X, y, W, b))
        W, b = gradient_descent(X, y, W, b, alpha)
    return costs, W, b
```

从上述代码可以看出，其整体上同 3.4.5 节中训练模型部分的代码一致，只是这里还额外地返回了训练好的参数。

3. 训练多分类模型

在完成上述两部分的代码后就可以进行整个多分类模型的训练了，代码如下：

```
def train(x, y, iter=1000):
    class_type = np.unique(y)
    costs, W, b = [], [], []
    for c in class_type:
        label = (y == c) * 1
        tmp = train_binary(x, label, iter=iter)
        costs.append(tmp[0])
        W.append(tmp[1])
        b.append(tmp[2])
    y_pre = prediction(x, W, b)
    print(classification_report(y, y_pre))
    return costs
```

在上述代码中，第 2 行用来判断数据集中一共有多少个类别，即需要训练多少个二分类器，第 4 行用来循环训练多个二分类器，第 7～9 行用来记录每个模型训练后的损失值和对应的权重参数。待整个模型训练完成后，便能够得到如图 3-11 所示的模型收敛图形。

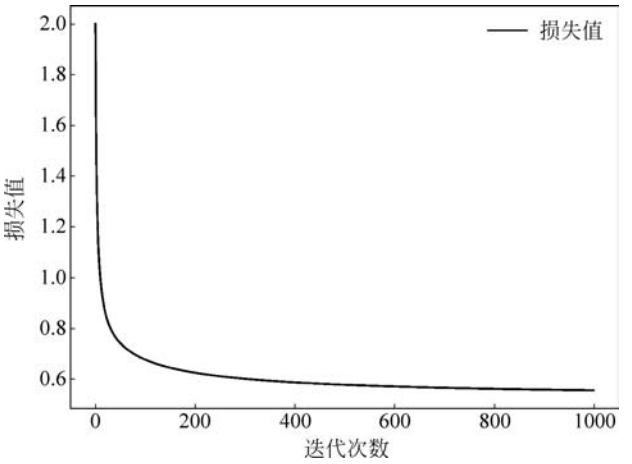

图 3-11　多分类模型收敛图形

从图 3-11 可以看出，整个模型大约在第 400 次迭代后就进入了收敛状态，最终也得到了大约 0.96 的准确率。

3.4.7　小结

在本节中，笔者首先介绍了逻辑回归中的映射函数（Sigmoid()函数）和样本分类时的概率表示，接着介绍了如何通过极大似然估计来推导并得到逻辑回归模型的目标函数，然后介绍了如何根据得到的目标函数来推导各个参数关于目标函数的梯度，最后，分别从零开始介绍了如何实现二分类模型和多分类逻辑回归模型。

总结一下，如图 3-12 所示，在本章中笔者首先通过一个示例引入了什么是分类模型，并通过在线性回归的基础上一步步地引出了什么是逻辑回归模型，然后笔者介绍了逻辑回归从建模到利用开源库进行求解的整个过程，接着介绍了如何通过逻辑回归来完成多分类任务及分类任务中常见的 4 种评价指标，并完成了阶段一的学习。最后，笔者通过本节的内容详细介绍了逻辑回归算法目标函数的推导及梯度的迭代公式等，还动手从 0 开始实现了逻辑回归的分类代码，进一步完成了后面两个阶段的学习。到此，对于逻辑回归的主要内容也就介绍完毕了。

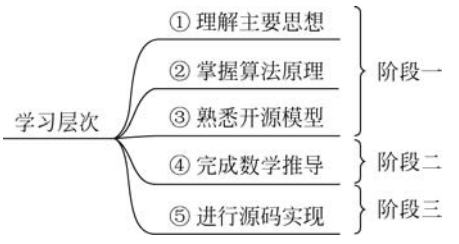

图 3-12　学习层次图

不过尽管如此，仍然还有一些提升模型性能的方法（例如数据集划分、正则化等）没有阐述，这些内容笔者将在第 4 章中进行详细介绍。