第5章

模型评估与优化

本章内容

- ◇ 算法链与管道
- ◇ 交叉验证
- ◇ 模型评价指标
- ◇ 处理类的不平衡问题
- ◇ 网格搜索优化模型

5.1 算法链与管道

大多数机器学习应用不仅需要应用单个算法，而且还需要将许多不同的处理步骤和机器学习模型链接在一起，这就是算法链。管道(Pipeline)指的是简化构建变换和模型链的过程。本节将介绍如何使用 Pipeline 类来简化构建变换和模型链的过程，以帮助快速构建模型。

微课视频

5.1.1 用管道方法简化工作流

管道可以理解为一个容器，然后把需要进行的操作都封装在这个管道里面进行操作，比如数据标准化、特征降维、主成分分析、模型预测等。为了便于理解，下面举一个实例来讲解。

1. 数据导入与预处理

本次导入威斯康星乳腺癌(Breast Cancer Wisconsin)二分类数据集，它包括 569 个样本。首列为主键 id，第 2 列为类别值(M＝恶性肿瘤，B＝良性肿瘤)，第 3～32 列为 30 个

根据细胞核的数字化图像计算出的实数特征值。

代码清单 5-1：算法链与管道

1）导入数据集

导入威斯康星乳腺癌数据集，输出前 5 条记录，如图 5.1 所示。

```
In[1]:
import pandas as pd
bcdf = pd.read_csv('wdbc.csv',header = None)
bcdf.head()
Out[1]:
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...

5 rows × 33 columns

图 5.1　数据集前 5 条记录

2）移除次要特征

首先移除 id 以及 Unnamed：32 这两个不需要的列，然后利用 map 映射函数将目标值(M,B)转为(1,0)。最后移除 diagnosis 列，移除之前将其保存到 y。移除次要特征之后的结果如图 5.2 所示。

```
In[2]:
bcdf.drop(['id','Unnamed: 32'], axis = 1, inplace = True)
bcdf['diagnosis'] = bcdf['diagnosis'].map({'M':1, 'B':0})
# 将目标转为 0-1 变量
X = bcdf
y = bcdf.diagnosis
bcdf.drop('diagnosis', axis = 1, inplace = True)
bcdf.head()
Out[2]:
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dim
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	

5 rows × 30 columns

图 5.2　移除次要特征后的前 5 条记录

3）划分训练及验证集

代码如下。

```
In[3]:
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                    test_size = 0.20, random_state = 1)
```

2. 使用管道创建工作流

很多机器学习算法要求特征取值范围要相同，因此需要对特征做标准化处理。此外，为了获取更好的分类效果，还应将原始的 30 维特征压缩至更少维度，这就需要用主成分分析(PCA)来完成，降维之后就可以利用各种模型进行回归预测了。

Pipeline 对象接收元组构成的列表作为输入，每个元组第一个值作为变量名，元组第二个元素是 sklearn 中的 transformer 或 estimator。管道中间每一步由 sklearn 中的 transformer 构成，最后一步是一个 estimator。

本次数据集中，管道包含两个中间步骤：StandardScaler 和 PCA，两者都属于 transformer，而 Logistic 回归分类器属于 estimator。

本案例中，当管道 Pipeline 执行 fit 方法时，实际上是分以下几步完成的。

(1) StandardScaler 执行 fit 和 transform 方法；

(2) 将转换后的数据输入 PCA；

(3) PCA 同样执行 fit 和 transform 方法；

(4) 最后将数据输入 LogisticRegression，训练一个 LR 模型。

对于管道来说，中间有多少个 transformer 都是可以的，这样就极大地简化了原本复杂的操作。管道的具体工作流程如图 5.3 所示。

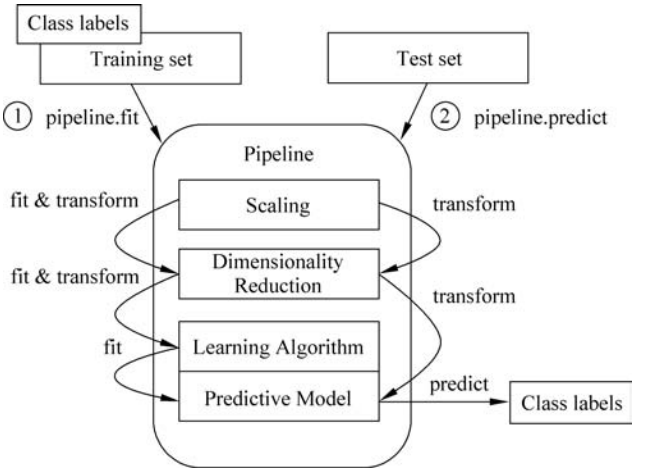

图 5.3　管道的工作流程

Pipeline 类的 Pipeline 函数可以把多个“处理数据的结点”按顺序打包在一起，数据将前一结点处理之后的结果，转到下一结点继续处理。当训练样本数据送进 Pipeline 进行处理时，它会逐个调用结点的 fit 和 transform 方法，最终用最后一个结点的 fit 方法来拟合数据。使用过程中要注意的是，管道执行 fit 方法，而 transformer 要执行 fit_transform。

本案例 Pipeline 的代码实现如下。

```
In[4]:
from sklearn.preprocessing import StandardScaler
# 用于数据标准化
from sklearn.decomposition import PCA        # 用于特征降维
from sklearn.linear_model import LogisticRegression
# 用于模型预测
from sklearn.pipeline import Pipeline
pipeline = Pipeline([('scl', StandardScaler()),
                     ('pca', PCA(n_components = 2)),
                     ('clf', LogisticRegression(random_state = 1))])
pipeline.fit(X_train, y_train)
print('Accuracy: %.3f' % pipeline.score(X_test, y_test))
y_pred = pipeline.predict(X_test)
Out[4]:
Accuracy: 0.959
```

5.1.2　通用的管道接口


微课视频

Pipeline 类不但可用于预处理和分类，实际上还可以将任意数量的估计器连接在一起。例如，可以构建一个包含特征提取、特征选择、缩放和分类的管道，总共 4 个步骤。当然，最后一步可以用回归或聚类代替分类。

对于管道中估计器的唯一要求就是，除了最后一步之外的所有步骤都需要具有 transform 方法，这样它们可以生成新的数据表示，以供下一个步骤使用。在调用 Pipeline.fit 的过程中，管道内部依次对每个步骤调用 fit 和 transform 方法，其输入是前一个步骤中 transform 方法的输出。对于管道中的最后一步，则仅调用 fit。

1. 用 make_pipeline 创建管道

利用传统语法创建管道较为麻烦，Pipeline 类提供了一个很方便的函数 make_pipeline，可以创建管道并根据每个步骤所属的类为其自动命名。如果多个步骤属于同一类，则会自动附加一个数字。

```
In[5]:
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
# 创建管道
pipe = make_pipeline(StandardScaler(), PCA(n_components = 2),
                     StandardScaler())
```

```
print(pipe.steps)
Out[5]:
[('standardscaler-1', StandardScaler(copy=True, with_mean=True, with_std=True)),
('pca', PCA(copy=True, iterated_power='auto', n_components=2, random_state=None,
  svd_solver='auto', tol=0.0, whiten=False)), ('standardscaler-2', StandardScaler
(copy=True, with_mean=True, with_std=True))]
```

可以清楚地看到,管道中的两个 StandardScaler 同属一个类,所以被自动命名为 standardscaler-1 和 standardscaler-2。

2. 访问步骤属性

Pipeline 类对象的步骤属性是由元组组成的步骤列表;如果想访问管道中某一步骤的属性,比如访问上述 PCA 提取的成分,最简单的方法是通过 named_steps 属性,它是一个字典,将步骤名称映射为估计器。

```
In[6]:
# 访问步骤属性
pipe.fit(bcdf)
components = pipe.named_steps["pca"].components_
#从 PCA 步骤中提取两个主成分
print(components.shape)
Out[6]:
(2, 30)
```

3. 访问网格搜索管道中的属性

使用管道主要是为了进行网格搜索。较为常见的任务是在网格搜索内访问管道的某些步骤。

下面举一个例子,假如要对威斯康星乳腺癌数据集上的 LogisticRegression 分类器进行网格搜索。

```
In[7]:
from sklearn.preprocessing import StandardScaler
# 进行数据标准化
from sklearn.model_selection import GridSearchCV
# 网格搜索
from sklearn.linear_model import LogisticRegression
# 逻辑回归模型预测
# 构建管道
pipe = make_pipeline(StandardScaler(), LogisticRegression())
# 对参数 C 在 0.01 至 100 之间进行搜索
param_grid = {"logisticregression__C":[0.01, 0.1, 1, 10, 100]}
# 在数据集上对网格搜索进行拟合
grid = GridSearchCV(pipe, param_grid, cv=5)
```

```
grid.fit(X_train, y_train)
# 输出最优估计器
print(grid.best_estimator_)
Out[7]:
Pipeline(memory = None,steps = [('standardscaler', StandardScaler(copy = True,
with_mean = True, with_std = True)), ('logisticregression',
LogisticRegression(C = 0.1, class_weight = None, dual = False,
fit_intercept = True,intercept_scaling = 1, max_iter = 100,
multi_class = 'ovr',n_jobs = 1,penalty = 'l2', random_state = None,
solver = 'liblinear',tol = 0.0001,verbose = 0, warm_start = False))])
```

从输出结果可以明显地看出，best_estimator_本质上是一个管道，它包含两个步骤：StandardScaler 和 LogisticRegression。

接下来可以使用管道的 named_steps 属性来访问 LogisticRegression 步骤。

```
In[8]:
# 访问网格搜索管道中的属性
print(grid.best_estimator_.named_steps["logisticregression"])
Out[8]:
LogisticRegression(C = 10)
#获取 LogisticRegression 实例后，就可以访问与每个输入特征相关的系数了
In[9]:
# 访问输入特征相关的系数
print(grid.best_estimator_.named_steps["logisticregression"].coef_)
Out[9]:
[[ 0.45085695    0.72799547    0.43505106  0.51008668    -0.06872988   -0.35212576
   0.90828396    0.5220248     0.09572507   -0.4305921    1.14523641    -0.32090688
   0.80349786    0.86030233    0.03973987   -0.56846732   0.20359127    0.0297218
    -0.44116833   -0.78750782  0.80001334   1.30721131    0.70507777    0.81499364
   1.01491881    0.22565912    1.15632453   0.71258993    1.05754829    0.06141073]]
```

5.2　交叉验证

交叉验证(Cross-Validation)主要用于防止模型过于复杂而引起的过拟合，是一种评价数据集泛化能力的统计方法。其基本思想是将原始数据划分为训练集(Train_Set)和测试集(Test_Set)。训练集用来对模型进行训练，测试集用来测试训练得到的模型，以此作为模型的评价指标。

交叉验证比单次划分训练集和测试集的方法更加稳定，在交叉验证中，数据被多次划分，并且需要训练多个模型。最常用的交叉验证是 K 折交叉验证(K-Fold Cross-Validation)以及分层 K 折交叉验证(Stratified K-Fold Cross Validation)。


微课视频

5.2.1 *K* 折交叉验证

1. *K* 折交叉验证的原理

***K* 折交叉验证**(*K*-Fold Cross-Validation)将数据集等比例划分成 *K* 份,每份叫作折(Fold)。以其中的一份作为测试集,其他的 *K*－1 份数据作为训练集,这样就完成了一次验证。因此,*K* 折交叉验证只有实验 *K* 次才算完整地完成,也就是说 *K* 折交叉验证实际是把验证重复做了 *K* 次,每次验证都是从 *K* 份选取一份不同的部分作为测试集(从而保证 *K* 份的数据都分别做过测试集),剩下的 *K*－1 份当作训练集,最后取 *K* 次准确率的平均值作为最终模型的评价指标。

K 折交叉验证可以有效避免过拟合和欠拟合状态的发生,*K* 值的选择可以根据实际情况调节。

图 5.4 展示了 *K*＝5 时 *K* 折交叉验证的完整过程。

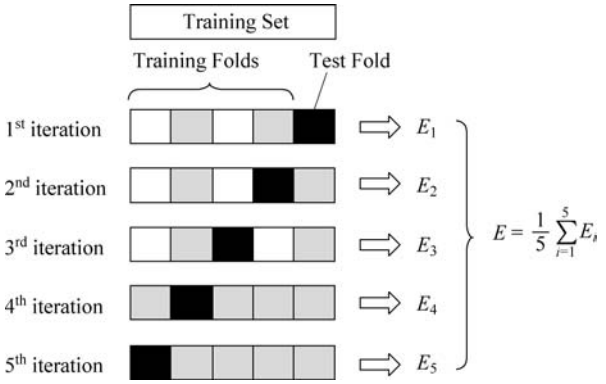

图 5.4　5 折交叉验证

下面举例来模拟一下图 5.4 的过程,首先导入所需模块,*X* 测试集中有 10 个数据,然后调用 *K* 折交叉验证函数,这里设置参数 n_splits 为 5,即进行 5 折交叉验证,最后用一个循环输出每一折的训练集和测试集的划分。

```
In[10]:
from sklearn.model_selection import KFold
X = ['a','b','c','d','e','f','g','h','i','j']
kf = KFold(n_splits = 5)
for train,test in kf.split(X):
    print("训练集 % s --- 测试集 % s" % (train,test))
Out[10]:
训练集[2 3 4 5 6 7 8 9] --- 测试集[0 1]
训练集[0 1 4 5 6 7 8 9] --- 测试集[2 3]
训练集[0 1 2 3 6 7 8 9] --- 测试集[4 5]
训练集[0 1 2 3 4 5 8 9] --- 测试集[6 7]
训练集[0 1 2 3 4 5 6 7] --- 测试集[8 9]
```

2. *K* 折交叉验证的实现

实际使用的时候没必要像上面那样写，因为 sklearn 已经封装好了相关方法，可以直接去调用这些方法，根据 K 折交叉验证的原理，选取 $K=10$ 时，在威斯康星乳腺癌数据集上 K 折交叉验证的代码实现如下。

代码清单 5-2：交叉验证

```
In[11]:
# 使用 K-Fold 交叉验证来评估模型性能
import numpy as np
from sklearn.model_selection import cross_val_score
scores = cross_val_score(estimator=pipe,
                         X=X_train,
                         y=y_train,
                         cv=10,
                         n_jobs=1)
#输出 K 折交叉验证的得分
print('CV accuracy scores: %s' % scores)
#用 np.mean()来计算均值,用 np.std()来计算标准差
print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores), np.std(scores)))
Out[11]:
CV accuracy scores: [0.975    0.95    0.975    1.    1.    1.1.    1.
0.97435897 1.    ]
CV accuracy: 0.987 +/- 0.017
```

其中，pipe 是 5.1 节创建的管道，X_train，y_train 是 5.1 节划分好的训练集，参数 CV 表示折数即 K 值。

5.2.2　分层 *K* 折交叉验证


微课视频

如图 5.5 所示，K 折交叉验证每次划分时对数据进行均分，假如数据集有 3 类，抽取出来的也正好是按照类别划分的 3 类，也就是说第一折的标签全是 0 类，第二折全是 1 类，第三折全是 2 类。这样划分数据集导致的后果是训练模型时没有学习到测试集中数据的特点，从而导致模型得分很低。

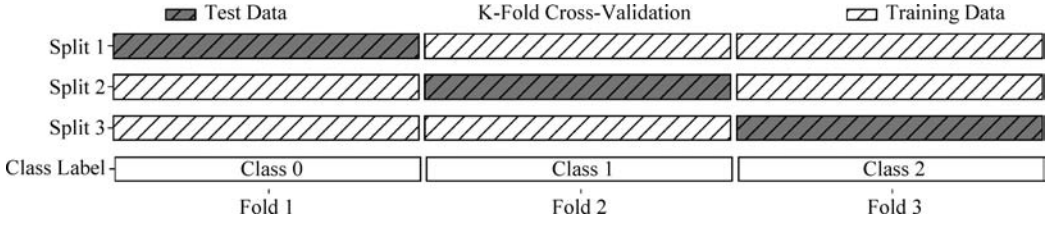

图 5.5　K 折交叉验证

下面来模拟图 5.5 的过程，首先导入所需模块，X 测试集中有 9 个数据，然后调用 K 折交叉验证函数，并设置参数 n_splits 为 3，即进行 3 折交叉验证，最后用一个循环输出每

一折的训练集和测试集的划分。

```
In[12]:
from sklearn.model_selection import KFold
X = ['a','b','c','d','e','f','g','h','i']
y = [1,1,1,2,2,2,3,3,3]
kf = KFold(n_splits = 3)
for train,test in kf.split(X,y):
    print("训练集 % s --- 测试集 % s" % (train,test))
Out[12]:
训练集[3 4 5 6 7 8] --- 测试集[0 1 2]
训练集[0 1 2 6 7 8] --- 测试集[3 4 5]
训练集[0 1 2 3 4 5] --- 测试集[6 7 8]
```

从第一次划分来看,并没有学习到测试集[0 1 2]中数据的特点,第二次划分也没有学习到测试集[3 4 5]中数据的特点,最后一次划分同样也没有学习到测试集[6 7 8]中数据的特点。

为了避免 K 折交叉验证出现的上述情况,又出现了以下几种交叉验证方式。

1. 分层 K 折交叉验证

分层 K 折交叉验证(Stratified K-Fold Cross Validation)同样属于交叉验证类型,分层的意思是在每一折中都保持着原始数据中各个类别的比例关系,比如:原始数据有 3 类,比例为 1∶2∶3,采用 3 折分层交叉验证,那么划分的 3 折中,每一折中的数据类别保持着 1∶2∶3 的比例,因为这样的验证结果更加可信,如图 5.6 所示。

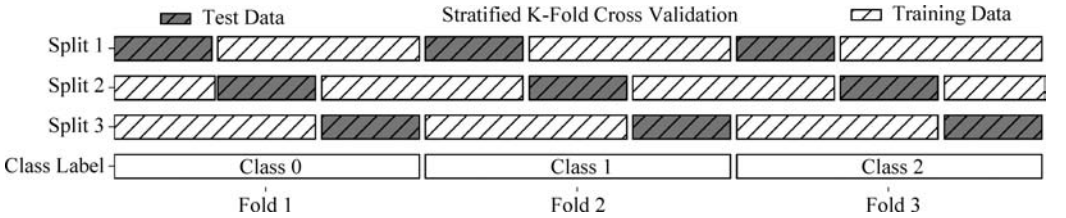

图 5.6 分层 K 折交叉验证

下面来模拟一下图 5.6 的过程,首先导入所需模块,X 测试集中有 9 个数据,然后调用 StratifiedKFold 即分层 K 折交叉验证函数,并设置参数 n_splits 为 3,即进行 3 折分层交叉验证,最后用一个循环输出每一折的训练集和测试集的划分。

```
In[13]:
from sklearn.model_selection import StratifiedKFold
X = ['a','b','c','d','e','f','g','h','i']
y = [1,1,1,2,2,2,3,3,3]
skf = StratifiedKFold(n_splits = 3)
for train,test in skf.split(X,y):
print("训练集 % s --- 测试集 % s" % (train,test))
```

```
Out[13]:
训练集[1 2 4 5 7 8] --- 测试集[0 3 6]
训练集[0 2 3 5 6 8] --- 测试集[1 4 7]
训练集[0 1 3 4 6 7] --- 测试集[2 5 8]
```

从结果可以清楚地看到，与 K 折交叉验证不同的是，分层 K 折交叉验证测试集每一次的划分都是从每个类别中各取一个数据，即每一折中的数据类别保持着 1∶1∶1 的比例，这样就可以充分学习到每一类中数据的特点，从而提高了模型的得分。

2. 留一法交叉验证

留一法交叉验证(Leave One Out Cross-Validation)是一种特殊的交叉验证方式。顾名思义，如果样本容量为 n，则 $K=n$，进行 n 折交叉验证，每次留下一个样本进行验证。由于每一折中几乎所有的样本皆用于训练模型，因此最接近原始样本的分布，这样评估所得的结果比较可靠。但其缺点也很明显，就是比较耗时，因此适合于数据集比较小的场合。

```
In[14]:
from sklearn.model_selection import LeaveOneOut
X = ['a','b','c','d','e','f','g','h','i','j']
y = [1,1,1,2,2,2,3,3,3,3]
loo = LeaveOneOut()
for train,test in loo.split(X,y):
    print("训练集 % s --- 测试集 % s" % (train,test))
Out[14]:
训练集[1 2 3 4 5 6 7 8 9] --- 测试集[0]
训练集[0 2 3 4 5 6 7 8 9] --- 测试集[1]
训练集[0 1 3 4 5 6 7 8 9] --- 测试集[2]
训练集[0 1 2 4 5 6 7 8 9] --- 测试集[3]
训练集[0 1 2 3 5 6 7 8 9] --- 测试集[4]
训练集[0 1 2 3 4 6 7 8 9] --- 测试集[5]
训练集[0 1 2 3 4 5 7 8 9] --- 测试集[6]
训练集[0 1 2 3 4 5 6 8 9] --- 测试集[7]
训练集[0 1 2 3 4 5 6 7 9] --- 测试集[8]
训练集[0 1 2 3 4 5 6 7 8] --- 测试集[9]
```

3. 打乱划分交叉验证

打乱划分交叉验证(Shuffle-Split Cross-Validation)是一种非常灵活的交叉验证。该方法控制更为灵活，可以控制每次划分时训练集和测试集的比例(通过 train_size 和 test_size 来控制)，以及划分迭代次数(通过 n_splits 来控制)。这种灵活的控制，甚至可以存在数据既不在训练集也不在测试集的情况。

```
In[15]:
from sklearn.model_selection import ShuffleSplit
X = ['a','b','c','d','e','f','g','h','i','j']
```

```
ssp = ShuffleSplit(test_size=.4, train_size=.4, n_splits=5)
for train,test in ssp.split(X):
print("训练集%s --- 测试集%s" %(train,test))
Out[15]:
训练集[8 0 4 6] --- 测试集[7 1 9 2]
训练集[6 0 4 5] --- 测试集[8 1 9 7]
训练集[0 2 4 6] --- 测试集[3 1 5 8]
训练集[9 3 1 7] --- 测试集[6 5 0 2]
训练集[5 0 8 3] --- 测试集[1 2 9 7]
```

从第一次划分可以看出,3 和 5 既不在训练集也不在测试集中。

4. 分组交叉验证

分组交叉验证(Group Cross-Validation)适用于数据中的分组高度相关的情况,即组内的各个变量之间不是独立的,而组间是独立的,也就是说测试集中的样本组别不能来自训练集中样本的组别。这种例子常见于医疗应用,可能拥有来自同一名病人的多个样本,但想要将其泛化到新的病人。

下面这个示例用到了一个由 groups 数组指定分组的模拟数据集。这个数据集包含了 6 个数据样本,groups 指定了该数据样本所属的分组,共分为 2 组,其中前 2 个样本为一组,后 4 个样本为一组,划分迭代次数为 2。

```
In[16]:
from sklearn.model_selection import GroupKFold
X=['a','b','c','d','e','f']
y=[0,0,1,1,1,1]
groups=[1,1,2,2,2,2]
gkf = GroupKFold(n_splits=2)
for train,test in gkf.split(X, y, groups=groups):
print("训练集%s --- 测试集%s" %(train,test))
Out[16]:
训练集[0 1] --- 测试集[2 3 4 5]
训练集[2 3 4 5] --- 测试集[0 1]
```

5.3 模型评价指标

对于一个模型来说,如何评价该模型的好坏,针对不同的问题需要不同的模型评价标准,也就是评估模型的泛化能力,这是机器学习中的一个关键性的问题。具体来讲,评价指标有两个作用:首先了解模型的泛化能力,可以通过同一个指标来对比不同模型,从而知道哪个模型相对较好,哪个模型相对较差;其次可以通过这个指标来逐步优化模型。因此,在选择模型与调参时,选择正确的指标是很重要的。本节将主要介绍分类模型的各种评价指标以及 ROC 和 AUC。

5.3.1　误分类的不同影响

误分类是指将被调查对象的特征错误地分到原本不属于它的类别中。例如将一个新型冠状病毒肺炎(Corona Virus Disease 2019,COVID-19)患者错误地诊断为健康人,这种误分类导致该患者不能得到及时的治疗,严重一点的话可能导致患者的死亡。把这种错误的阳性预测叫**假阳性**(False Positive),这种错误属于**第一类错误**(Type Ⅰ Error);相反,一个健康的人被错误的诊断为新型冠状病毒肺炎,不但会给患者造成不必要的物质上的损失,更重要的是会给患者带来精神上的极大痛苦。把这种错误的阴性预测叫**假阴性**(False Negative),这种错误属于**第二类错误**(Type Ⅱ Error)。

所有的分类器都存在偏好,因此都存在误分类的现象,但是可以通过调整分类器的阈值,比如高的召回率或者高的准确率。这样就可以通过设定分类器的阈值来避免不同类型的错误,这样模型才有实际的应用价值。

5.3.2　混淆矩阵

混淆矩阵(Confusion matrix)是表示精度评价的一种标准格式,用 n 行 n 列的矩阵来表示。混淆矩阵是总结分类模型预测结果的情形分析表,以矩阵形式将数据集中的记录按照真实的类别与分类模型预测的类别进行汇总。其中矩阵的行表示真实值,矩阵的列表示预测值。

在学习混淆矩阵之前,首先明确几个分类评估指标中相关符号的含义:

1. 真阳性(True Positive,TP):将正类预测为正类的次数,即真实值和预测值均为 1 的次数。

2. 假阳性(False Positive,FP):将负类预测为正类的次数,即真实值为 0,而预测值为 1 的次数。

3. 真阴性(True Negative,TN):将负类预测为负类的次数,即真实值和预测值均为 0 的次数。

4. 假阴性(False Negative,FN):将正类预测为负类的次数,即真实值为 1,而预测值为 0 的次数。

下面先以二分类为例,看下混淆矩阵的表现形式,如表 5.1 所示。

表 5.1　二分类的混淆矩阵

真　实　值	预　测　值	
	负类(N)	正类(P)
负类(N)	真阴性(TN)	假阳性(FP)
正类(P)	假阴性(FN)	真阳性(TP)

也可以画出二分类的混淆矩阵的解释图,如图 5.7 所示。

代码清单 5-3:模型评价指标

```
In[17]:
import mglearn
```

```
3   mglearn.plots.plot_binary_confusion_matrix()
4   Out[17]:
```

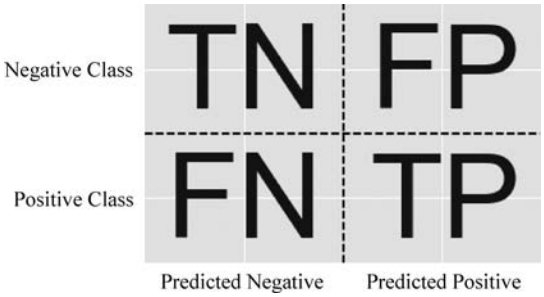

图 5.7　二分类混淆矩阵

在 Scikit-learn 中,提供了混淆矩阵函数 sklearn. metrics. confusion_matrix 的 API 接口,可以用于绘制混淆矩阵,如下所示。

```
1   sklearn.metrics.confusion_matrix(
2       y_true,                  # 样本真实的分类标签列表
3       y_pred,                  # 样本预测的分类结果列表
4       labels = None,           # 给出的类别,通过这个可对类别进行选择
5       sample_weight = None     # 样本权重
6   )
```

首先举个简单的例子。

```
1   In[18]:
2   from sklearn.metrics import confusion_matrix
3   y_true = [0, 1, 0, 1]
4   y_pred = [1, 1, 1, 0]
5   print(confusion_matrix(y_true, y_pred))
6   Out[18]:
7   [[0 2]
8   [1 1]]
```

输出了一个 2×2 的矩阵,该矩阵代表的含义如下。

第一行的 0,即真阴性(TN),表示将真实值 0 预测为 0 的次数,很显然,真实值里面有两个 0,但都预测错误(预测为 1)了,因此 TN 的值是 0。第一行的 2,即假阳性(FP),表示将真实值 0 错误预测为 1 的次数,很显然是 2 次,因此 FP 的值是 2。第二行前面的 1,即假阴性(FN),表示将 1 错误预测为 0 的次数,很明显是 1 次,因此 FN 的值是 1;第二行后面的 1,即真阳性(TP),表示将 1 预测正确的次数,很明显也是 1 次,所以 TP 的值是 1。

也可以通过 confusion_matrix 直接得到 TN、FP、FN、TP 这四个值,如下所示。

```
In[19]:
TN, FP, FN, TP = confusion_matrix([0, 1, 0, 1], [1, 1, 1, 0]).ravel()
print(TN, FP, FN, TP)
Out[19]:
0 2 1 1
```

理解了二分类的混淆矩阵，接下来来看多分类模型的混淆矩阵，以 Scikit-learn 官方所提供的例子为例。

```
In[20]:
from sklearn.metrics import multilabel_confusion_matrix # sklearn version >= 0.21
y_true = ["cat", "ant", "cat", "cat", "ant", "bird"]
y_pred = ["ant", "ant", "cat", "cat", "ant", "cat"]
mcm = multilabel_confusion_matrix(y_true, y_pred,labels = ["ant", "bird", "cat"])
mcm
Out[20]:
array([[[3, 1],
        [0, 2]],
       [[5, 0],
        [1, 0]],
       [[2, 1],
        [1, 2]]], dtype = int64)
```

本例显然是个三分类问题，mcm 可以看作返回了三个二分类混淆矩阵，在每一个二分类混淆矩阵中，如图 5.7 所示，TN 在[0，0]，FP 在[0，1]，FN 在[1，0]，TP 在[1，1]。

以第一个类别 ant 为例，正样本预测对的有 2 次，正样本预测错的是 0 次，它的负样本（bird、cat）预测成正样本的有 1 次，负样本预测对的有 3 次（其中 bird 预测成 cat，也算对，因为它们都是负样本）。

如果对官方的例子还不明白的话，接下来将上面的混淆矩阵可视化，最简单的方法是使用 seaborn 的热力图。热力图是机器学习数据可视化比较常用的显示方式，它通过颜色变化程度以直观反映出热点分布、区域聚集等相关数据信息。

代码如下，其输出如图 5.8 所示。

```
In[21]:
import pandas as pd
import seaborn as sns
from sklearn.metrics import confusion_matrix
y_true = ["cat", "ant", "cat", "cat", "ant", "bird"]
y_pred = ["ant", "ant", "cat", "cat", "ant", "cat"]
cm = confusion_matrix(y_true, y_pred, labels = ["ant", "bird", "cat"])
df = pd.DataFrame(cm,index = ["ant", "bird", "cat"],columns = ["ant", "bird", "cat"])
sns.heatmap(df,annot = True)
Out[21]:
```

正如前面所讲，从图 5.8 中可以明显地看出：混淆矩阵的每一行数据之和代表该类

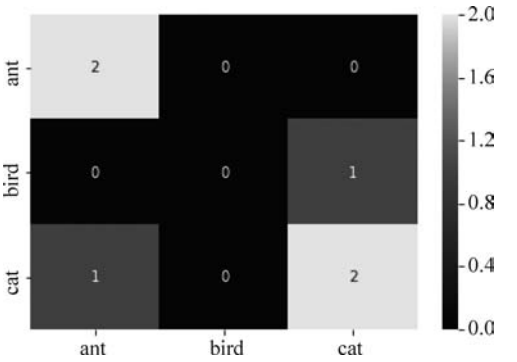

图 5.8 热力图绘制的混淆矩阵

别真实的数目,每一列之和代表该类别预测的数目,矩阵的对角线上的数值代表被正确预测的样本数目。

纵轴的标签表示真实属性,而横轴的标签表示分类的预测结果。以此热力图的第一行第一列这个数字 2 为例,它表示 ant 被成功分类成为 ant 的样本数目;同理,第三行第一列的数字 1 表示 cat 被分类成 ant 的样本数目,以此类推。

5.3.3 分类的不确定性

在开始深入探究如何使用不确定性来调试和解释模型之前,先来理解为什么不确定性如此重要。

1. 分类的不确定性

一个显著的例子就是高风险应用。假设你正在设计一个模型,用以辅助医生决定对患者的最佳治疗方案。在这种情况下,不仅要关注模型预测结果的准确性,还要关注模型对预测结果的确定性程度。如果结果的不确定性过高,那么医生应该慎重考虑。自动驾驶汽车是另外一个有趣的例子。当模型不确定道路上是否有行人时,可以使用此信息来减慢车速或者触发警报,便于驾驶员进行处理。

2. 不确定度指标

不确定度指标实际上反映了数据依赖于模型和参数的不确定程度。Scikit-learn 接口中有两个函数可以用于获取分类器的不确定度估计,即分类器预测某个测试点属于某个类别的置信程度。这两个函数分别是 decision_function 和 predict_proba。

下面对这两个函数进行简单的介绍:

1) decision_function

对于二分类的情况,decision_function(决策函数)的输出可以在任意范围取值,返回值的形状是 n_samples,注意这是一个一维数组,它为每个样本都返回一个浮点数,正值表示对正类(classes_属性的第二个元素)的置信程度,负值表示对负类(classes_属性的第一个元素)的置信程度,绝对值越大表示置信度越高。

对于多分类情况，decision_function 返回值的形状是(n_samples，n_classes)，每一列对应每个类别的“确定性分数”，分数较高的类别可能性更大。

2) predict_proba

predict_proba(预测概率)的输出是每个类别的概率，总是为 0～1，两个类别的元素(概率)之和始终为 1。不管是二分类还是多分类，predict_proba 函数返回值的形状均为 n_samples，n_classes。

下面以 SVM 二分类为例，来分析一下结果。

```
In[22]:
import numpy as np
from sklearn.svm import SVC

X = np.array([[1,2,3],
                    [1,3,4],
                    [2,1,2],
                    [4,5,6],
                    [3,5,3],
                    [1,7,2]])

y = np.array([3, 3, 3, 2, 2, 2])

clf = SVC(probability=True)
clf.fit(X, y)
print("decision_function: \n",clf.decision_function(x))
print("predict:",clf.predict(x))
print("predict_proba: \n",clf.predict_proba(x))
print("classes_: ",clf.classes_)
Out[22]:
decision_function:
 [ 1.00089036  0.64493601  0.97960658  -1.00023781 -0.9995244  -1.00023779]
predict: [3 3 3 2 2 2]
predict_proba:
[[0.09157972  0.90842028]
[0.20435202  0.79564798]
[0.09633253  0.90366747]
[0.94858803  0.05141197]
[0.94849393  0.05150607]
[0.94858803  0.05141197]]
classes_:[2 3]
```

如前面所讲，二分类问题的 decision_function 函数返回结果的形状与样本数量相同，且返回结果的数值表示模型预测样本属于正类的可信度。decision_function 函数返回的浮点数是可以带符号的，大于 0 表示正类的可信度大于负类，否则表示正类的可信度小于负类。

所以对于前 3 个样本，decision_function 都认为是正类的可信度高(大于 0)，后 3 个样本是负类的可信度高(小于 0)。

接下来看一下 predict 函数的结果，前 3 个预测为正类 3，后 3 个样本预测为负类 2。

再看一下 predict_proba 函数预测的样本所属的类别概率，可以看到前 3 个样本属于类别 3(正类)的概率更大，后 3 个样本属于类别 2(负类)的概率更大。两个类别的概率之和始终为 1，即每个列表中两个元素之和是 1。

最后，二分类情况下 classes_中的第一个标签“2”代表负类，第二个标签“3”代表正类。


微课视频

5.3.4 准确率-召回率曲线

在二分类混淆矩阵中，可以很容易地看出，主对角线上(TN 与 TP)是全部预测正确的，副对角线上(FN 与 FP)是全部预测错误的。因此当得到模型的混淆矩阵后，就需要去观察对角线上的数量是否大，副对角线上的数量是否小。但是，混淆矩阵里面统计的是个数，有时候面对大量的数据，仅仅算个数，很难衡量模型的优劣。因此混淆矩阵在基本的统计结果上又延伸了如下几个指标。

1. 精确率

精确率(Accuracy)即分类正确的样本数占总样本的比例。

$$\text{Accuracy}=\frac{\text{TP}+\text{TN}}{\text{TP}+\text{TN}+\text{FP}+\text{FN}} \tag{5-1}$$

以 5.3.2 节的例子为例，来计算一下它的 Accuracy。

```
In[23]:
from sklearn.metrics import confusion_matrix
y_test = [0, 1, 0, 1]
y_predict = [1, 1, 1, 0]
TN, FP, FN, TP = confusion_matrix(y_test, y_predict).ravel()
print(TN, FP, FN, TP)
Out[23]:
0 2 1 1
In[24]:
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_predict)
print("Accuracy = ",accuracy)
Out[24]:
Accuracy = 0.25
```

很明显 $\text{Accuracy}=\frac{1+0}{1+0+2+1}=0.25$。

一般来说系统的精确率越高，性能越好。但是，对于正负样本数量极不均衡的情况，只通过精确率往往难以反映出系统的真实性能。比如，对于一个地震预测系统，假设所有样本中，1000 天中有 1 天发生地震(0：不地震，1：地震)，分类器不假思索地将所有样本分类为 0，即可得到 99.99%的精确率，但当地震真正来临时，并不能成功预测，这种结果是我们不能接受的。

2. 准确率

准确率(Precision)又叫**查准率**,其含义为预测为真的样本中实际为真的样本的占比。

$$\text{Precision}=\frac{\text{TP}}{\text{TP}+\text{FP}} \tag{5-2}$$

```
In[25]:
from sklearn.metrics import precision_score
precision = precision_score(y_test, y_predict)
print("Precision = ",precision)
Out[25]:
Precision = 0.3333333333333333
```

很明显 $\text{Precision}=\frac{1}{1+2}\approx 0.3333333333333333$。

3. 召回率

召回率(Recall)也叫**敏感性**(Sensitivity),又叫**查全率**。这个指标表示正样本中预测对的占总正样本的比例。顾名思义,“查全”表明预测为真覆盖到了多少实际为真的样本,换句话说遗漏了多少。

$$\text{Recall}=\frac{\text{TP}}{\text{TP}+\text{FN}} \tag{5-3}$$

```
In[26]:
from sklearn.metrics import recall_score
recall = recall_score(y_test, y_predict)
print("Recall = ",recall)
Out[26]:
Recall = 0.5
```

很明显 $\text{Recall}=\frac{1}{1+1}=0.5$。

准确率和召回率是此消彼长的,即准确率提高了,召回率就会降低。因此,在不同的应用场景下,我们对准确率和召回率的关注点不同。例如,在假币预测、股票预测的时候,我们更关心的是准确率;而在地震预测、肿瘤预测的时候,我们更关注召回率,即地震或患病时预测错了的情况应该越少越好。

4. F1 值

F1 值(F1-Score)是用来衡量分类模型综合性能的一种指标。它同时兼顾了分类模型的准确率和召回率。F1 值可以看作模型准确率和召回率的调和平均数,它的最大值是 1,最小值是 0。

调和平均数的性质是,当准确率和召回率二者都非常高的时候,它们的调和平均才会高。如果其中之一很低,调和平均就会被拉得接近于很低的那个数。

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{5-4}$$

```
In[27]:
from sklearn.metrics import f1_score
f1_Score = f1_score(y_test, y_predict)
print("F1 - Score = ",f1_Score)
Out[27]:
F1 - Score =  0.4
```

很明显 $F1 = \frac{2 \times 0.3333333333333333 \times 0.5}{0.3333333333333333 + 0.5} \approx 0.4$。

5. 准确率-召回率曲线

如上所述,准确率-召回率(查准率-查全率)是一对相互矛盾的性能指标,而事实上我们期望的是既能保证查准率,又能提升查全率。

准确率-召回率曲线(Precision-Recall 曲线,P-R 曲线)也叫**查准率-查全率曲线**。对分类问题来讲,通过不断调整分类器的阈值,可以得到不同的 Precision-Recall 值,遍历所有可能的阈值,从而可以得到一条曲线(纵坐标为 Precision,横坐标为 Recall)。通常随着分类阈值从大到小变化,查准率减小,而查全率增加,如图 5.9 所示。

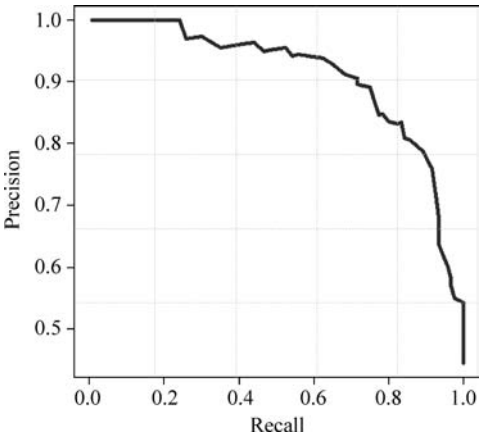

图 5.9 P-R 曲线

从图 5.9 中可以看出,查准率和查全率是一对相互矛盾的性能指标,比较两个分类器优劣时,显然是查得又准又全的比较好,也就是 P-R 曲线往坐标(1,1)的位置越靠近越好。

通常情况下,P-R 曲线在分类、检索等领域有着广泛的使用,用来表现分类、检索的性能。比如做病患检测、垃圾邮件过滤时,在保证查准率的前提下,提升查全率。

Scikit-learn 提供的接口 precision_recall_curve 函数是用来绘制 P-R 曲线,其函数原型如下。

```
sklearn.metrics.precision_recall_curve(
        y_true,                 # y_true 是在范围{0,1}或{-1,1}中真实的二进制标签
        probas_pred,            #array, shape = [n_samples] 估计概率或决策函数
        pos_label = None,       # 正类样本标签
        sample_weight = None    # 采样权重
        )
```

返回值为 Precision、Recall 和 thresholds(所选阈值)。

接下来,使用 precision_recall_curve 函数来画出 P-R 曲线,如图 5.10 所示。

```
In[28]:
import matplotlib
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_curve

#y_true 为样本实际的类别,y_scores 为样本为正例的概率
y_true = np.array([1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0])
y_scores = np.array([0.9, 0.75, 0.86, 0.47, 0.55, 0.56, 0.74, 0.62, 0.5, 0.86, 0.8,
0.47, 0.44, 0.67, 0.43, 0.4, 0.52, 0.4, 0.35, 0.1])

plt.figure("P-R Curve")
plt.title('Precision/Recall Curve')
plt.xlabel('Recall')
plt.ylabel('Precision')

precision, recall, thresholds = precision_recall_curve(y_true, y_scores)
plt.plot(recall,precision)
plt.show()
Out[28]:
```

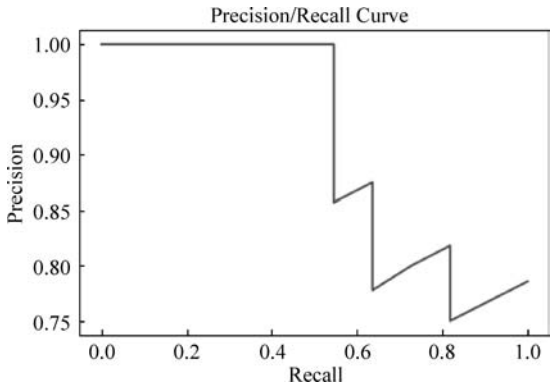

图 5.10　P-R 曲线


微课视频

5.3.5 受试者工作特征(ROC)与 AUC

1. 受试者工作特征

受试者工作特征曲线(Receiver Operator Characteristic Curve，ROC 曲线)通常被用来评价一个二值分类器的优劣。ROC 曲线的横坐标是**假阳性率**(False Positive Rate，FPR)，纵坐标是**真阳性率**(True Positive Rate，TPR)。

TPR 表示在所有实际为阳性的样本中，被正确地判断为阳性的比率，即 $TPR=\frac{TP}{TP+FN}$。而 **FPR** 表示在所有实际为阴性的样本中，被错误地判断为阳性之比率，即 $FPR=\frac{FP}{FP+TN}$。

TPR 越高，FPR 越低，则可以证明分类器分类效果越好。但是两者又是相互矛盾的，所以单凭 TPR 和 FPR 的两个值是没有办法比较两个分类器的好坏的，因此提出了 ROC 曲线。

对某个分类器而言，可以根据其在测试样本上的表现得到一个 TPR 和 FPR 点对。这样，此分类器就可以映射为 ROC 平面上的一个点。调整这个分类器在分类时使用的阈值，就可以得到一个经过(0，0)和(1，1)的曲线，该曲线就是此分类器的 ROC 曲线。如图 5.11 所示。

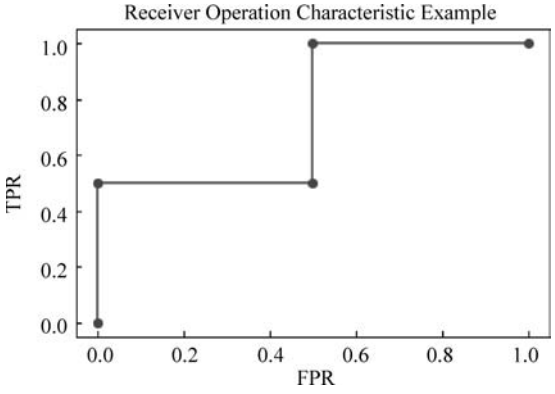

图 5.11 ROC 曲线

一般情况下，这个曲线都应该处于(0，0)和(1，1)连线的上方。因为(0，0)和(1，1)连线形成的 ROC 曲线实际上代表的是一个随机分类器。如果很不幸得到一个位于此直线下方的分类器的话，一个直观的补救办法就是把所有的预测结果反向处理。

ROC 曲线反映出 TPR 的增加以 FPR 的增加为代价，最靠近坐标左上方的点为 TPR 和 FPR 均较高的临界值，ROC 曲线下的面积是模型准确率的度量。

Sklearn 提供的接口 roc_curve 函数用来绘制 ROC 曲线。

```
1   fpr, tpr, thresholds = sklearn.metrics.roc_curve(
2         y_true,      # y_true 是在范围{0,1}或{-1,1}中真实的二进制标签.如果标签不是二进制的,则应该显式地给出 pos_label
```

```
    y_score,                    # 预测得分
    pos_label = None,           # 正类样本标签
    sample_weight = None,       # 即采样权重,可选取其中的一部分进行计算
    drop_intermediate = True    # 即可选择去掉一些对于 ROC 性能不利的阈值,使得得到的曲线有更好的表现性能
)
```

其中,返回值 thresholds 是所选择的不同的阈值,将预测结果 scores 按照降序排列。接下来以 Scikit-learn 官网给出的例子为例,代码如下。

```
In[29]:
import numpy as np
from sklearn import metrics
y = np.array([1, 1, 2, 2])
scores = np.array([0.1, 0.4, 0.35, 0.8])
fpr, tpr, thresholds = metrics.roc_curve(y, scores, pos_label = 2)
# 负类标签为 1,正类为 2
print("FPR:",fpr)
print("TPR:",tpr)
print("thresholds:",thresholds)
Out[29]:
FPR: [0. 0. 0.5 0.5 1. ]
TPR: [0. 0.5 0.5 1. 1. ]
thresholds: [1.8 0.8 0.4 0.35 0.1 ]
```

thresholds 为将预测结果 scores 从大到小排列的结果。

最后画出 ROC 曲线,如图 5.11 所示。

```
In[30]:
import matplotlib.pyplot as plt

plt.plot(fpr,tpr,marker = 'o')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('Receiver Operating Characteristic Example')
plt.show()
Out[30]:
```

2. AUC

AUC(Area Under ROC Curve)是一种用来度量分类模型好坏的一个标准,假设分类器的输出是样本属于正类的 score(置信度),则 AUC 的物理意义为,任取一对(正、负)样本,正样本的 score 大于负样本的 score 的概率。AUC 值为 ROC 曲线所覆盖的区域面积。显然,AUC 越大,分类器分类效果越好。

AUC=1 是完美分类器。采用这个预测模型时,不管设定什么阈值都能得出完美预

测。但绝大多数预测的场合,不存在完美分类器。

0.5<AUC<1,优于随机猜测,若妥善设定阈值,分类器将具有预测价值。

AUC=0.5,跟随机猜测一样,模型没有预测价值。

AUC<0.5,比随机猜测还差,但只要总是反预测而行,就优于随机猜测。

Scikit-learn 计算 ROC 曲线下面积 AUC 有两种方法。

1) AUC 函数

使用 sklearn.metrics.auc 函数,它是一个通用方法,根据梯形规则计算曲线下面积,其函数原型如下。

```
sklearn.metrics.auc(
    x,                  # x 坐标,即 FPR,它必须是单调递增或单调递减的
    y,                  # y 坐标,即 TPR
    reorder = False     #默认值
    )
```

下面用 AUC 函数来计算图 5.11 所示 ROC 曲线下的面积。

```
In[31]:
import numpy as np
from sklearn import metrics
y = np.array([1, 1, 2, 2])
pred = np.array([0.1, 0.4, 0.35, 0.8])
fpr, tpr, thresholds = metrics.roc_curve(y, pred, pos_label = 2)
metrics.auc(fpr, tpr)
Out[31]:
0.75
```

2) roc_auc_score 函数

使用 sklearn.metrics.roc_auc_score,根据预测得分计算接受者工作特征曲线(ROC 曲线)下的面积,其函数原型如下:

```
sklearn.metrics.roc_auc_score(
    y_true,             #真实的标签
    y_score,            #预测得分
    average = 'macro',  #{'micro', 'macro', 'samples', 'weighted'} or None, default = 'macro'
    sample_weight = None, #样本权重
    max_fpr = None,     #float,> 0 and <= 1, default = None
    multi_class = 'raise', #{'raise', 'ovr', 'ovo'}, default = 'raise'
    labels = None       # 数组的形状(n_classes,), default = None
    )
```

下面用 roc_auc_score 来计算图 5.11 所示 ROC 曲线下的面积。

```
In[32]:
import numpy as np
from sklearn.metrics import roc_auc_score
y_true = np.array([0, 0, 1, 1])
y_scores = np.array([0.1, 0.4, 0.35, 0.8])
roc_auc_score(y_true, y_scores)
Out[32]:
0.75
```

最后，以威斯康星乳腺癌二分类数据集为例，结合 5.2 节的交叉验证以及 ROC 与 AUC，画出 ROC 曲线并求出面积 AUC，完整代码如下，输出结果如图 5.12 所示。

```
In[33]:
    #导入所需要的模块
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import StratifiedKFold

#导入威斯康星乳腺癌数据集并进行预处理
bcdf = pd.read_csv('wdbc.csv')
bcdf.drop(['id','Unnamed: 32'], axis = 1, inplace = True)
bcdf['diagnosis'] = bcdf['diagnosis'].map({'M':1, 'B':0})
X = bcdf
y = bcdf.diagnosis
bcdf.drop('diagnosis', axis = 1, inplace = True)
n_samples, n_features = X.shape

# 增加噪声特征
random_state = np.random.RandomState(0)
X = np.c_[X, random_state.randn(n_samples, 200 * n_features)]

cv = StratifiedKFold(n_splits = 5)      #利用分层 K 折交叉验证，将数据划分为 5 份
# SVC 模型
classifier = svm.SVC(kernel = 'linear', probability = True,random_state = random_state)
# 画平均 ROC 曲线的两个参数
mean_tpr = 0.0                          # 用来记录画平均 ROC 曲线的信息
mean_fpr = np.linspace(0, 1, 100)
cnt = 0
for i, (train, test) in enumerate(cv.split(X,y)):   # 利用模型划分数据集和目标变量
    cnt += 1
    probas_ = classifier.fit(X[train], y[train]).predict_proba(X[test])   # 训练模
型并预测概率
    fpr, tpr, thresholds = roc_curve(y[test], probas_[:, 1])   # 调用 roc_curve 函数
    mean_tpr += np.interp(mean_fpr, fpr, tpr)   # 累计每次循环的总值后以求平均值
    mean_tpr[0] = 0.0                           # 坐标以 0 为起点
```

```
roc_auc = auc(fpr, tpr)                    # 求 AUC 面积
    # 画出当前分割数据的 ROC 曲线
    plt.plot(fpr, tpr, lw = 1, label = 'ROC fold {0:.2f} (area = {1:.2f})'.format(i, roc_auc))
plt.plot([0, 1], [0, 1], '--', color = (0.6, 0.6, 0.6), label = 'Luck')
# 画对角线

mean_tpr /= cnt                            # 求平均值
mean_tpr[-1] = 1.0                         # 坐标以 1 为终点
mean_auc = auc(mean_fpr, mean_tpr)         # 求平均 AUC 面积

plt.plot(mean_fpr, mean_tpr, 'k--',label = 'Mean ROC (area = {0:.2f})'.format(mean_auc), lw = 2)
# 设置 x、y 轴的上下限,设置宽一点,以免和边缘重合,以便更好地观察图像的整体
plt.xlim([-0.05, 1.05])
plt.ylim([-0.05, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Breast Cancer Wisconsin ROC')
plt.legend(loc = "lower right")
plt.show
Out[33]:
```

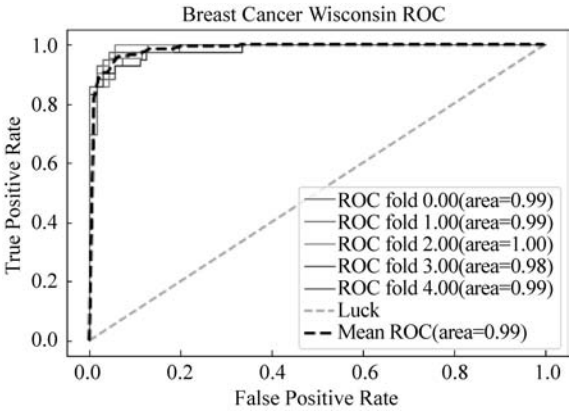

图 5.12 威斯康星乳腺癌 ROC 曲线


微课视频

5.3.6 多分类指标

前面介绍了二分类的评价指标,多分类问题的所有评价指标基本上都来自二分类指标,但是要对所有类别进行平均。具体来讲,评价多分类问题时,通常把多分类问题分解为 n 个二分类问题,每次以其中一个类为正类,其余类统一为负类,并计算各种二分类指标,最后再对所有类别进行平均进而得到多分类评价指标。

多类分类评估有三种办法,分别对应 sklearn. metrics 中参数 average 值为 micro、macro 和 weighted 的情况,三种方法所求的值一般也不同。

1. 多分类的 Accuracy、Precision、Recall 和 F1-score 指标

（1）macro：分别计算第 i 类的 Precision、Recall 和 F1-score（把第 i 类当作正类，其余所有类统一为负类），然后进行平均。

（2）micro：计算所有类别中 FP、FN 和 TP 的总数，然后利用这些数来分别计算 Precision、Recall 和 F1-score，这些值都等于 Accuracy 的值。

（3）weighted：是为了解决 macro 中没有考虑样本不均衡的情况，在计算 Precision 与 Recall 时，各个类别的 Precision 与 Recall 要乘以该类在总样本中的占比来求和。

接下来以三分类为例，分别来计算参数 average 值为 micro、macro 和 weighted 时的 Precision、Recall 和 F1-score。

首先生成一组数据，数据分为－1、0、1 三类，真实数据 y_true 中，一共有 30 个－1、240 个 0 和 30 个 1。生成数据并计算混淆矩阵，代码如下。

```
In[34]:
import numpy as np
from sklearn.metrics import confusion_matrix
y_true = np.array([-1]*30 + [0]*240 + [1]*30)
y_pred = np.array([-1]*10 + [0]*10 + [1]*10 +
                  [-1]*40 + [0]*160 + [1]*40 +
                  [-1]*5 + [0]*5 + [1]*20)
confusion_matrix(y_true, y_pred)
Out[34]:
array([[ 10, 10, 10],
       [ 40, 160, 40],
       [ 5, 5, 20]], dtype=int64)
```

计算参数 average 值为 macro 的情况。

```
In[35]:
from sklearn.metrics import precision_score,recall_score,f1_score
precision=precision_score(y_true, y_pred, average="macro")
recall=recall_score(y_true, y_pred, average="macro")
f1=f1_score(y_true, y_pred, average="macro")
print("precision:",precision)
print("recall:",recall)
print("f1:",f1)
Out[35]:
precision: 0.46060606060606063
recall: 0.5555555555555555
f1: 0.4687928183321521
```

计算参数 average 值为 micro 的情况。

```
In[36]:
precision = precision_score(y_true, y_pred, average = "micro")
recall = recall_score(y_true, y_pred, average = "micro")
f1 = f1_score(y_true, y_pred, average = "micro")
print("precision:",precision)
print("recall:",recall)
print("f1:",f1)
Out[36]:
precision: 0.6333333333333333
recall: 0.6333333333333333
f1: 0.6333333333333333
```

由于 micro 算法把所有的类放在一起算，比如 Precision，就是把所有类的 TP 加起来，再除以所有类的 TP 和 FP 相加之和。因此 micro 方法下的 Precision、Recall 和 F1-score 的值都等于 Accuracy。

计算参数 average 值为 weighted 的情况。

```
In[37]:
precision = precision_score(y_true, y_pred, average = "weighted")
recall = recall_score(y_true, y_pred, average = "weighted")
f1 = f1_score(y_true, y_pred, average = "weighted")
print("precision:",precision)
print("recall:",recall)
print("f1:",f1)
Out[37]:
precision: 0.7781818181818182
recall: 0.6333333333333333
f1: 0.6803968816442238
```

2. 多分类的 ROC 曲线及 AUC

计算多分类的 ROC 曲线以及 AUC 值时，若使用 roc_auc_score 函数，则参数 average 要设置为 average="micro"或"macro"；若使用 AUC 函数，则采用 fpr["micro"]、tpr["micro"]或者 fpr["macro"]、tpr["macro"]。

以鸢尾花数据集为例，代码如下，输出的 ROC 曲线如图 5.13 所示。

```
In[38]:
import numpy as np
import matplotlib.pyplot as plt
from itertools import cycle
from sklearn import svm, datasets
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier
```

```
from scipy import interp

# 加载数据
iris = datasets.load_iris()
x = iris.data
y = iris.target
# 将标签二值化,即变成[1 0 0] [0 0 1] [0 1 0]
y = label_binarize(y, classes = [0, 1, 2])

# 设置种类
n_classes = y.shape[1]

# 训练模型并预测
random_state = np.random.RandomState(0)
n_samples, n_features = x.shape

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = .5,random_state = 0)

classifier = OneVsRestClassifier(svm.SVC(kernel = 'linear', probability = True,random_
state = random_state))
y_score = classifier.fit(x_train, y_train).decision_function(x_test)

# 计算每一类的 ROC
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):          # 遍历三个类别
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Compute micro-average ROC curve and ROC area(micro 方法)
fpr["micro"], tpr["micro"], _ = roc_curve(y_test.ravel(), y_score.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# Compute macro-average ROC curve and ROC area(macro 方法)
# First aggregate all false positive rates
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
# Then interpolate all ROC curves at this points
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += interp(all_fpr, fpr[i], tpr[i])

# Finally average it and compute AUC
mean_tpr /= n_classes
fpr["macro"] = all_fpr
tpr["macro"] = mean_tpr
roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

```

```
# Plot all ROC curves
lw = 2
plt.figure()
plt.plot(fpr["micro"], tpr["micro"],
         label = 'micro - average ROC curve (area = {0:0.2f})'
               ''.format(roc_auc["micro"]),
         color = 'deeppink', linestyle = ':', linewidth = 4)

plt.plot(fpr["macro"], tpr["macro"],
         label = 'macro - average ROC curve (area = {0:0.2f})'
               ''.format(roc_auc["macro"]),
         color = 'navy', linestyle = ':', linewidth = 4)

colors = cycle(['aqua', 'darkorange', 'cornflowerblue'])
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color = color, lw = lw,
             label = 'ROC curve of class {0} (area = {1:0.2f})'
             ''.format(i, roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k -- ', lw = lw)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic of multi - class')
plt.legend(loc = "lower right")
plt.show()
Out[38]:
```

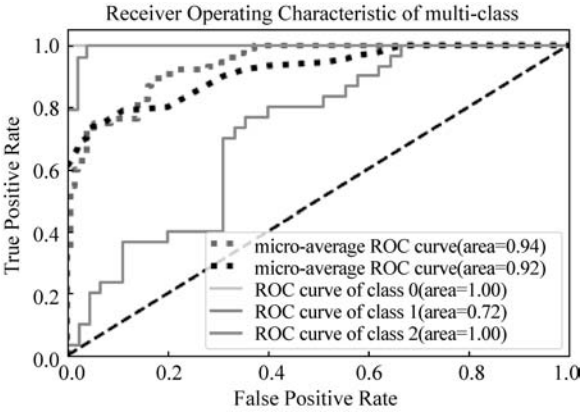

图 5.13　鸢尾花 ROC 曲线

3. classification_report

sklearn 中的 classification_report 函数用于显示主要分类指标的文本报告。在报告

中显示每个类的 Precision、Recall 和 F1-score 以及 Accuracy(Micro 平均)、Macro 平均、Weighted 平均等信息,是评价模型便捷且全面的工具。

```
sklearn.metrics.classification_report(
    y_true,              #一维数组,或标签指示器数组/稀疏矩阵,目标值
    y_pred,              #一维数组,或标签指示器数组/稀疏矩阵,分类器返回的估计值
    labels = None,       #array,shape = [n_labels],报表中包含的标签索引的可选列表
    target_names = None, #字符串列表,与标签匹配的可选显示名称(相同顺序)
    sample_weight = None, #类似于 shape = [n_samples]的数组,可选项,样本权重
    digits = 2,          #输出浮点值的位数,如果 output_dict = True,此参数不起作用
    output_dict = False  #为 True 则评估结果以字典形式返回
)
```

```
In[39]:
from sklearn.metrics import classification_report
y_true = [0, 1, 2, 2, 2]
y_pred = [0, 0, 2, 2, 1]
target_names = ['class A', 'class B', 'class C']
print(classification_report(y_true, y_pred, target_names = target_names))
Out[39]:
                precision    recall  f1 - score    support

      class A       0.50      1.00      0.67         1
      class B       0.00      0.00      0.00         1
      class C       1.00      0.67      0.80         3

     accuracy                           0.60         5
    macro avg       0.50      0.56      0.49         5
weighted avg       0.70      0.60      0.61         5
```

其中,support(支持度)指原始的真实数据中属于该类的数目,即每个标签出现的次数。

5.3.7　回归指标


微课视频

回归模型是机器学习中很重要的一类模型,不同于常见的分类模型,分类问题的评价指标是准确率,回归算法的评价指标主要是 MSE、RMSE、MAE、MedAE、R2、EVS。

1. 均方误差

均方误差(Mean Squared Error,MSE),是反映估计量与被估计量之间差异程度的一种度量,其值越小说明拟合效果越好,所以常被用作线性回归的损失函数。

MSE 的计算公式如式(5-5)所示。

$$\mathrm{MSE}=\frac{1}{n}\sum_{i=1}^{n}(y_i-\hat{y}_i)^2 \tag{5-5}$$

sklearn 使用 mean_squared_error 函数计算均方误差。

```
In[40]:
from sklearn.metrics import mean_squared_error
y_true = [3, -0.5, 2, 7]
y_pred = [2.5, 0.0, 2, 8]
mean_squared_error(y_true, y_pred)
Out[40]:
0.375
```

2. 平均绝对误差

平均绝对误差(Mean Absolute Error,MAE),预测目标值和实际目标值之间误差的绝对值的平均数,可以更好地反映预测值误差的实际情况,其值越小越好。

MAE 的计算公式如式(5-6)所示。

$$\mathrm{MAE}=\frac{1}{n}\sum_{i=1}^{n}|y_i-\hat{y}_i| \tag{5-6}$$

sklearn 使用 mean_absolute_error 函数计算平均绝对误差。

```
In[41]:
from sklearn.metrics import mean_absolute_error
y_true = [3, -0.5, 2, 7]
y_pred = [2.5, 0.0, 2, 8]
mean_absolute_error(y_true, y_pred)
Out[41]:
0.5
```

3. 中位绝对误差

中位绝对误差(Median Absolute Error,MedAE)通过取目标值和预测值之间的所有绝对差值的中值来计算损失,其值越小越好。

MedAE 的计算公式如式(5-7)所示。

$$\mathrm{MedAE}(y,\hat{y})=\mathrm{median}(|y_1-\hat{y}_1|,\cdots,|y_n-\hat{y}_n|) \tag{5-7}$$

sklearn 使用 median_absolute_error 函数计算中位绝对误差。

```
In[42]:
from sklearn.metrics import median_absolute_error
y_true = [3, -0.5, 2, 7]
y_pred = [2.5, 0.0, 2, 8]
median_absolute_error(y_true, y_pred)
Out[42]:
0.5
```

4. R2 决定系数

R2 决定系数(R-Squared)表示回归方程在多大程度上解释了因变量的变化,或者说

方程对观测值的拟合程度如何。R2 决定系数的最优值为 1(完全拟合)；为 0 时，说明模型和样本基本没有关系；也可为负，为负时说明模型非常差。

R-Squared 计算公式为：

$$R^2(y,\hat{y})=1-\frac{\sum_{i=1}^{n}(y_i-\hat{y}_i)^2}{\sum_{i=1}^{n}(y_i-\bar{y})^2} \tag{5-8}$$

sklearn 使用 r2_score 函数计算 R2 决定系数。

```
In[43]:
from sklearn.metrics import r2_score
y_true = [3, -0.5, 2, 7]
y_pred = [2.5, 0.0, 2, 8]
r2_score(y_true, y_pred)
Out[43]:
0.9486081370449679
```

5. 可释方差得分

可释方差得分(Explained Variance Score，EVS)，也叫解释方差得分。表征模型中残差的方差在整个数据集所占的比重的变量，可释方差值最好的分数是 1.0，分数越低效果越差。

计算公式为：

$$\text{explained\_variance}(y,\hat{y})=1-\frac{\text{var}\{y-\hat{y}\}}{\text{var}\{y\}} \tag{5-9}$$

sklearn 使用 explained_variance_score 函数计算可释方差得分。

```
In[44]:
from sklearn.metrics import explained_variance_score
y_true = [3, -0.5, 2, 7]
y_pred = [2.5, 0.0, 2, 8]
explained_variance_score(y_true, y_pred)
Out[44]:
0.9571734475374732
```

5.3.8 在模型选择中使用评估指标


微课视频

前面详细讨论了若干种评估方法，本节以手写数字数据集(digits)为例，并根据真实情况和具体模型来应用前面所学的分类指标及回归指标。

1. 分类指标的应用

首先导入 digits 数据集。

```
In[45]:
import pandas as pd
```

```
from sklearn.datasets import load_digits
digits = load_digits()
```

查看模块的属性列表。

```
In[46]:
dir(digits)
Out[46]:
['DESCR', 'data', 'feature_names', 'frame', 'images', 'target', 'target_names']
```

样本集中第一个手写数字“0”的图像特征。

```
In[47]:
print(digits.images[0])
Out[47]:
[[ 0.  0.  5.  13.  9.  1.  0.  0.]
 [ 0.  0.  13.  15.  10.  15.  5.  0.]
 [ 0.  3.  15.  2.  0.  11.  8.  0.]
 [ 0.  4.  12.  0.  0.  8.  8.  0.]
 [ 0.  5.  8.  0.  0.  9.  8.  0.]
 [ 0.  4.  11.  0.  1.  12.  7.  0.]
 [ 0.  2.  14.  5.  10.  12.  0.  0.]
 [ 0.  0.  6.  13.  10.  0.  0.  0.]]
```

样本集中第一个手写数字“0”的可视化，如图 5.14 所示。

```
In[48]:
import matplotlib.pyplot as plt
plt.imshow(digits.images[0],cmap = 'binary')
plt.show()
Out[48]:
```

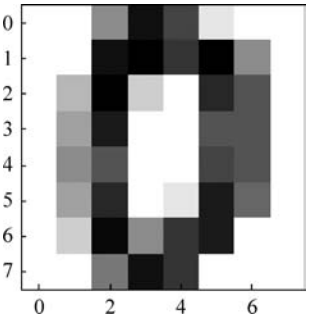

图 5.14　手写“0”的可视化

划分数据集，其中 20%用于测试，80%用于训练。

```
In[49]:
from sklearn.model_selection import train_test_split
```

```
X_train,X_test,y_train,y_test = train_test_split(x,y,test_size = 0.2,random_state = 0)
print("x.shape:",x.shape,"y.shape:",y.shape)
print("X_train:", X_train.shape)
print("y_train:", y_train.shape)
print("X_test:", X_test.shape)
print("y_test:", y_test.shape)
Out[49]:
x.shape: (1797, 64) y.shape: (1797,)
X_train: (1437, 64)
y_train: (1437,)
X_test: (360, 64)
y_test: (360,)
```

训练模型并调优。

```
In[50]:
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
param_grid = { 'C': [0.001, 0.01, 1, 10], 'gamma': [0.001, 0.01, 1, 10], 'kernel':
['linear', 'rbf']}
grid = GridSearchCV(SVC(), param_grid = param_grid)
grid.fit(X_train, y_train)
print("Best parameters:", grid.best_params_)
print("Best score: {:.3f}".format(grid.best_score_))
print("Test set accuracy: {:.3f}".format(grid.score(X_test, y_test)))
print("Best estimator: {}".format(grid.best_estimator_))
Out[50]:
Best parameters: {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}
Best score: 0.991
Test set accuracy: 0.992
Best estimator: SVC(C = 1, gamma = 0.001)
```

调用 classification_report 输出各评估指标的值。

```
In[51]:
from sklearn.metrics import classification_report
predicted = grid.best_estimator_.predict(X_test)
print(classification_report(y_test, predicted))
Out[51]:
                precision    recall  f1 - score  support

           0       1.00      1.00      1.00        27
           1       0.97      1.00      0.99        35
           2       1.00      1.00      1.00        36
           3       1.00      1.00      1.00        29
           4       1.00      1.00      1.00        30
           5       0.97      0.97      0.97        40
           6       1.00      1.00      1.00        44
```

```
15              7       1.00      1.00      1.00        39
16              8       1.00      0.97      0.99        39
17              9       0.98      0.98      0.98        41
18
19       accuracy                           0.99       360
20      macro avg       0.99      0.99      0.99       360
21   weighted avg       0.99      0.99      0.99       360
```

最后可视化混淆矩阵,如图 5.15 所示。

```
In[52]:
import seaborn as sn
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,predicted)
sn.heatmap(cm, annot = True)
Out[52]:
```

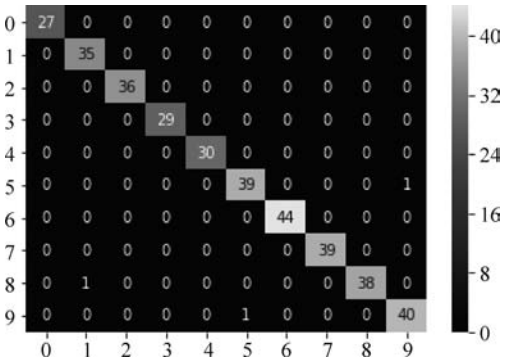

图 5.15 热力图绘制的混淆矩阵(见彩插)

从上面的热力图可以看出,模型仅有一次将数字 1 识别为 8,一次将数字 5 识别为 9,一次将数字 9 识别为 5。除此之外,模型全部识别正确。

2. 回归指标的应用

具体代码如下。

```
In[53]:
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import r2_score
from sklearn.metrics import explained_variance_score
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import median_absolute_error
```

```
# 导入 digits 数据集
digits = load_digits()
n_samples = len(digits.images)
x = digits.images.reshape((n_samples, -1))
y = digits.target
# 划分数据集,其中 20 % 用于测试,80 % 用于训练
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.2,random_state = 0)
# 训练数据
lr = LogisticRegression()
lr.fit(x_train,y_train)
y_pre_lr = lr.predict(x_test)
lr_R2 = r2_score(y_test,y_pre_lr)
lr_EVS = explained_variance_score(y_test, y_pre_lr)
lr_MAE = mean_absolute_error(y_test, y_pre_lr)
lr_medAE = median_absolute_error(y_test, y_pre_lr)
lr_MSE = mean_squared_error(y_test, y_pre_lr)
# 输出评估指标
print("R2 决定系数:",lr_R2)
print("可释方差:",lr_EVS)
print("平均绝对误差:",lr_MAE)
print("中位绝对误差:",lr_medAE)
print("均方误差:",lr_MSE)
Out[53]:
R2 决定系数: 0.8996259447195352
可释方差: 0.8997035535251644
平均绝对误差: 0.1527777777777778
中位绝对误差: 0.0
均方误差: 0.8083333333333333
```

最后可视化混淆矩阵,如图 5.16 所示。

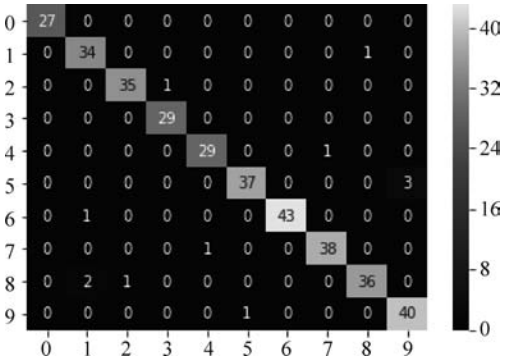

图 5.16　热力图绘制的混淆矩阵(见彩插)

```
In[54]:
import seaborn as sn
from sklearn.metrics import confusion_matrix
```

```
4   cm = confusion_matrix(y_test,y_pre_lr)
5   sn.heatmap(cm, annot = True)
6   Out[54]:
```

从图 5.16 中可以看出,由于未对模型进行调优,因此识别效果明显不如图 5.15。

微课视频

5.4 处理类的不平衡问题

在机器学习的实践中,通常会遇到实际数据中正负样本比例不平衡的情况,也叫数据倾斜。对于数据倾斜的情况,如果选取的算法不合适,或者评价指标不合适,那么对于实际应用线上时效果往往不尽人意,所以如何解决数据不平衡问题是实际生产中非常常见且重要的问题。

5.4.1 类别不平衡问题

类别不平衡(Class-Imbalance)是指分类任务中不同类别的训练样例数目差别很大的情况。在现实的分类学习任务中,经常会遇到类别不平衡的现象,比如在二分类问题中,通常假设正负类别相对均衡,然而实际应用中类别不平衡的现象是非常常见的,比如疾病检测、产品抽检、邮件过滤、信用卡欺诈等。

如果不同类别的训练样例数目稍有差别,通常影响不大,但若差别很大,则会对学习过程造成困扰。如图 5.17 所示,产品抽检数据集中有 998 个反例,但是正例只有 2 个,那么学习方法只需要返回一个永远将新样本预测为反例的学习器,就能达到 99.8%的精度;然而这样的学习器没有任何实际价值,因为它不能预测出任何正例,因此有必要了解类别不平衡问题处理的基本方法。

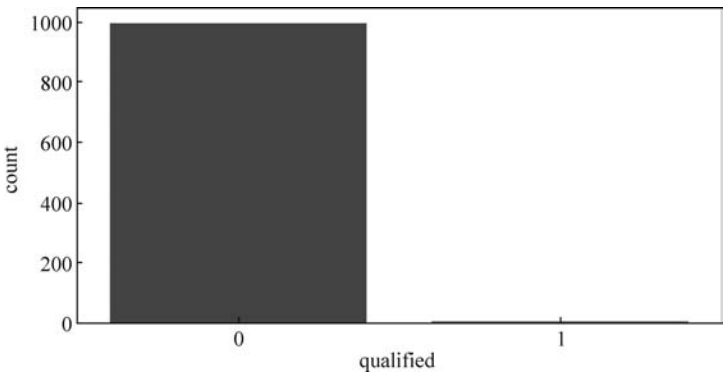

图 5.17 抽检数据集正反例对比图

如何解决机器学习中类别不平衡问题呢?严格地讲,任何数据集都存在数据不平衡的现象,这往往是由问题本身决定的,处理时只需要关注那些分布差别比较悬殊的;另外,虽然很多数据集都包含多个类别,但这里着重考虑二分类,因为在解决了二分类中的数据不平衡问题后,推而广之,就能得到多分类情况下的解决方案。

5.4.2　解决类别不平衡问题

1. 采样法

采样法是通过对训练集进行预处理，使其从不平衡转变为较平衡的数据集的方法。该方法是较为常用的方法，并且通常情况下比较有效。采样法分为欠采样和过采样两种。

1）**欠采样**

欠采样（Undersampling）通过减少分类中多数类的样本数量，使得正例、反例数目平衡。最直接的方法是随机地去掉一些多数类样本，减小多数类的规模，然后再进行学习。该方法的缺点是会丢失多数类样本中的一些重要信息。

2）**过采样**

过采样（Oversampling）通过增加分类中少数类样本的数量，使得正例、反例数目平衡。最直接的方法是简单复制少数类样本，形成多条记录，然后再进行学习。该方法的缺点是如果样本特征少，可能导致过拟合。

2. 惩罚权重

该方法在算法实现过程中，对于分类中不同样本数量的类别分别赋予不同的权重，即小样本数量类别赋予较高权重，而大样本数量类别赋予较低权重，然后进行计算和建模。使用这种方法时需要对样本作额外处理，只需要在算法模型的参数中进行相应设置即可。很多模型和算法中都有基于类别参数的调整设置，以 Scikit-learn 中的 SVM 为例，通过将 class_weight 参数设置为 balanced 即可。这样 SVM 会将权重设置为与不同类别样本数量成反比的权重来做自动均衡处理，因此该方法是更加简单且高效的方法。

代码清单 5-4：处理类的不平衡问题

对 Scikit-learn 官网的例子稍加改动为例，具体代码如下，输出如图 5.18 所示。

```
In[55]:
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn.datasets import make_blobs

# we create two clusters of random points
n_samples_1 = 1000
n_samples_2 = 100
centers = [[0.0, 0.0], [2.0, 2.0]]
clusters_std = [1.5, 0.5]
X, y = make_blobs(n_samples=[n_samples_1, n_samples_2],
                  centers=centers,
                  cluster_std=clusters_std,
                  random_state=0, shuffle=False)

# fit the model and get the separating hyperplane
```

```
clf = svm.SVC(kernel = 'linear', C = 1.0)
clf.fit(X, y)

# fit the model and get the separating hyperplane using weighted classes
wclf = svm.SVC(kernel = 'linear', class_weight = 'balanced')   # 官网中 class_weight = {1: 10}
wclf.fit(X, y)

# plot the samples
plt.scatter(X[:, 0], X[:, 1], c = y, cmap = plt.cm.Paired, edgecolors = 'k')

# plot the decision functions for both classifiers
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()

# create grid to evaluate model
xx = np.linspace(xlim[0], xlim[1], 30)
yy = np.linspace(ylim[0], ylim[1], 30)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T

# get the separating hyperplane
Z = clf.decision_function(xy).reshape(XX.shape)

# plot decision boundary and margins
a = ax.contour(XX, YY, Z, colors = 'k', levels = [0], alpha = 0.5, linestyles = ['-'])

# get the separating hyperplane for weighted classes
Z = wclf.decision_function(xy).reshape(XX.shape)

# plot decision boundary and margins for weighted classes
b = ax.contour(XX, YY, Z, colors = 'r', levels = [0], alpha = 0.5, linestyles = ['-'])
plt.legend([a.collections[0], b.collections[0]], ["non weighted", "weighted"], loc =
"upper right")
plt.show()
Out[55]:
```

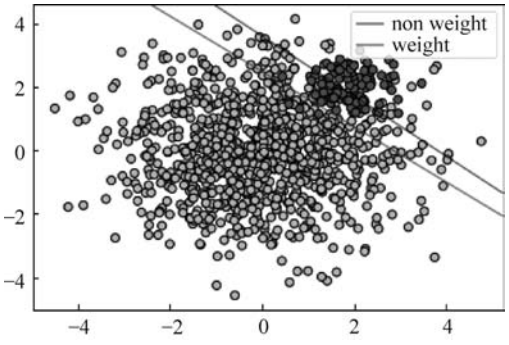

图 5.18　用 SVC 最优分离超平面(见彩插)

首先用平面 SVC 找到分离平面(non weighted,用蓝色线表示),然后通过将 class_weight 参数设置为 balanced,自动校正不平衡类别并绘制分离超平面(weighted,用红色线表示)。

5.5　网格搜索优化模型

机器学习应用中,有两种类型的参数:一种是从训练集中学习到的参数,例如逻辑回归的权重;另一种是为了使学习算法达到最优而可调节的参数,即需要预先优化设置而非通过训练得到的参数,例如逻辑回归中的正则化参数或决策树中的深度参数、人工神经网络模型中隐藏层层数和每层的结点个数、正则项中常数大小等,这种可调节的参数称为超参数(Hyperparameters)。超参数选择不恰当的话,就会出现欠拟合或者过拟合的问题。

在选择超参数的时候,有两种途径:一种是凭经验微调;另一种就是选择不同大小的参数,代入模型中,挑选表现最好的参数。第一种微调的方法是手工调制超参数,直到找到一个好的超参数组合,但是这么做的话非常耗时,也可能没有时间探索多种组合。所以最常用的方法就是利用网格搜索,通过调参来评估一个模型的泛化能力。

5.5.1　简单网格搜索选择超参数


微课视频

网格搜索是模型超参数(即需要预先优化设置而非通过训练得到的参数)的优化技术,常用于优化三个或者更少数量的超参数,本质是穷举法。对于每个超参数,使用者选择一个较小的有限集去探索。然后,由这些超参数的笛卡儿乘积得到若干组超参数。网格搜索使用每组超参数训练模型,挑选验证集误差最小的超参数作为最优的超参数。

网格搜索采用的是穷举法的思路,其计算复杂度将随需要优化的超参数规模指数增长。因此该方法只适用于规模很小的超参数优化问题。当超参数规模较大时,随机搜索将会是更高效的超参数优化方法。

现在,要用网格搜索这个更加强大的超参数优化工具来找到超参数值的最优组合,从而进一步改善模型的性能。以鸢尾花数据集为例,通过调节 SVM 分类器的 C 和 gamma 参数,在 2 个参数上使用 for 循环,对每种参数组合分别训练并评估一个分类器,实现简单的网格搜索。

代码清单 5-5:网格搜索优化模型

```
In[56]:
from sklearn.datasets import load_iris
iris_dataset = load_iris()
In[57]:
# 用简单网格搜索选择超参数
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
X_train, X_test, y_train, y_test = train_test_split( iris.data, iris.target, random_
state = 0)
```

```
print("training set: {} test set: {}".format( X_train.shape[0], X_test.shape[0]))
best_score = 0                                    # 存储当前最好分数
for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:      # gamma 参数
    for C in [0.001, 0.01, 0.1, 1, 10, 100]:      # C 参数
        # 对 gamma 和 C 参数的每种组合都训练一个 SVC
        svm = SVC(gamma = gamma, C = C)
        svm.fit(X_train, y_train)
        # 在测试集上对 SVC 进行评估
        score = svm.score(X_test, y_test)
        # 保存更高的分数和对应的参数
        if score > best_score:
            best_score = score
            best_parameters = {'C': C, 'gamma': gamma}
print("Best score: {:.2f}".format(best_score))
print("Best parameters: {}".format(best_parameters))
Out[57]:
Size of training set: 112 size of test set: 38
Best score: 0.97
Best parameters: {'C': 100, 'gamma': 0.001}
```

得分为97%,看起来还不错。但值得注意的是,将原始数据集划分成训练集和测试集以后,其中测试集除了用作调整参数,也用来测量模型的好坏。这样做导致最终的评分结果比实际效果好,因为测试集在调参过程中被送到模型里,而我们的目的是将训练模型应用到未曾见过的新数据上。

微课视频

5.5.2 验证集用于选择超参数

已经知道,用测试集估计学习器的泛化误差,其重点在于测试样本不能以任何形式参与到模型的选择之中,包括超参数的设定,否则将导致过拟合。基于这个原因,测试集中的样本不能用于验证集。因此,只能从训练数据中构建验证集,对训练集再进行一次划分,分为训练集和验证集。这样划分的结果就是:原始数据划分为3份,分别为训练集(Training Set)、验证集(Validation Set)和测试集(Testing Set)。其中训练集用于学习参数(即训练模型);验证集用于估计训练中或训练后的泛化误差,更新超参数(即挑选超参数);而测试集用来衡量模型表现的好坏(即评价模型的泛化能力)。

这样一来就将原始数据集划分为如图5.19所示的训练集、验证集及测试集3个数据集。

```
In[58]:
import mglearn
mglearn.plots.plot_threefold_split()
Out[58]:
```

具体流程如图5.20所示。

将数据集划分为训练集、验证集和测试集后,利用验证集选定最佳参数,用找到的最

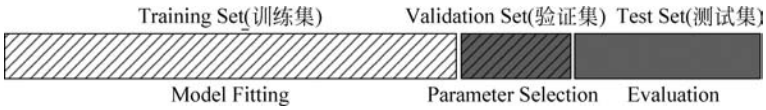

图 5.19　训练集、验证集和测试集

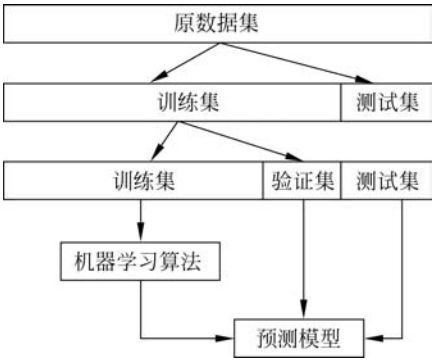

图 5.20　验证集用于选择超参数的流程

优超参数重新构建模型，并同时在训练集和验证集上进行训练，以便使用尽可能多的数据来构建模型，从而获得较好的评估结果。

具体的代码实现如下。

```
In[59]:
from sklearn.svm import SVC
# 将数据划分为训练集与测试集
X_trainval, X_test, y_trainval, y_test = train_test_split(iris.data, iris.target,
random_state = 0)
# 将训练集划分为训练集与验证集
X_train, X_valid, y_train, y_valid = train_test_split(X_trainval, y_trainval, random_
state = 1)
print("training set: {} validation set: {} test set:"
" {}\n".format(X_train.shape[0], X_valid.shape[0], X_test.shape[0]))

best_score = 0

for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:          # gamma 参数
    for C in [0.001, 0.01, 0.1, 1, 10, 100]:          # C 参数
        # 对 gamma 和 C 参数的每种组合都训练一个 SVC
        svm = SVC(gamma = gamma, C = C)
        svm.fit(X_train, y_train)
        # 在验证集上评估 SVC
        score = svm.score(X_valid, y_valid)
        # 保存更高的分数和对应的参数
        if score > best_score:
          best_score = score
          best_parameters = {'C': C, 'gamma': gamma}
```

```
    # 在训练+验证集上重新构建一个模型,并在测试集上进行评估
    svm = SVC(**best_parameters)
    svm.fit(X_trainval, y_trainval)
test_score = svm.score(X_test, y_test)

print("Best score on validation set: {:.2f}".format(best_score))
print("Best parameters: ", best_parameters)
print("Test set score with best parameters: {:.2f}".format(test_score))
Out[59]:
training set: 84 validation set: 28 test set: 38

Best score on validation set: 0.96
Best parameters: {'C': 10, 'gamma': 0.001}
Test set score with best parameters: 0.92
```

从输出结果可以看到,验证集上的最高分数是96%,比之前的97%低了1%,主要因为这次使用了更少的数据(一部分被划分为了验证集)来训练模型。测试集上的分数为92%,这个分数实际反映了模型的泛化能力,也就是说模型仅对92%的新数据进行了正确的分类,而不是之前认为的97%。

微课视频

5.5.3 带交叉验证的网格搜索

尽管将数据集划分为训练集、验证集和测试集的方法相对有用,可行性较高。但是该方法对数据的划分比较敏感,也就是说其最终的表现好坏与初始数据的划分结果有很大的关系,且有时候泛化性能较低,为了得到更好的泛化性能的更好估计,可以通过交叉验证来评估每种组合的性能并以此来降低偶然性。

```
In[60]:
from sklearn.model_selection import cross_val_score

best_score = 0
for gamma in [0.001,0.01,1,10,100]:
    for c in [0.001,0.01,1,10,100]:
        # 对于每种参数可能的组合,进行一次训练
        svm = SVC(gamma=gamma,C=c)
        # 5折交叉验证
        scores = cross_val_score(svm,X_trainval,y_trainval,cv=5)
        score = scores.mean()
        # 找到表现最好的参数
        if score>best_score:
            best_score = score
            best_parameters = {'gamma':gamma,"C":c}

# 使用最佳参数,构建新的模型
svm = SVC(**best_parameters)

```

```
# 使用训练集和验证集进行训练
svm.fit(X_trainval,y_trainval)

# 模型评估
test_score = svm.score(X_test,y_test)

print('Best score on validation set :{:.2f}'.format(best_score))
print('Best parameters:{}'.format(best_parameters))
print('Best score on test set:{:.2f}'.format(test_score))
Out[60]:
Best score on validation set :0.97
Best parameters:{'gamma': 0.01, 'C': 100}
Best score on test set:0.97
```

从运行结果可以看出，验证集上的最高分数是97%，比5.5.2节提高了1%；测试集上的分数为97%，比5.5.2节提高了5%，说明交叉验证的使用进一步提高了模型的泛化能力。

在实际应用中，交叉验证经常与网格搜索进行结合，即**带交叉验证的网格搜索**(Grid Search with Cross Validation)，并以此作为参数评价的一种常用方法。为此Scikit-learn提供了GridSearchCV类，它以估计器(Estimator)的形式实现了这种方法。

1. GridSearchCV简介

GridSearchCV存在的意义就是自动调参，只要把参数输进去，它就能给出最优化结果和参数。GridSearchCV其实可以拆分为GridSearch和CV两部分，即网格搜索和交叉验证。网格搜索，搜索的是参数，即在指定的参数范围内，按步长依次调整参数，利用调整的参数训练学习器，从所有的参数中找到在验证集上精度最高的参数，这其实是一个训练和比较的过程。交叉验证的目的是提高模型的泛化能力，得到可靠稳定的模型。

GridSearchCV可以保证在指定的参数范围内找到精度最高的参数，但是这也是网格搜索的缺陷所在：它要求遍历所有可能参数的组合，在面对大数据集和多参数的情况下，非常耗时。

2. GridSearchCV类构造方法参数说明

GridSearchCV类的构造方法的语法格式如下。

```
__init__(self, estimator, param_grid, scoring = None, fit_params = None,n_jobs = 1, iid =
True, refit = True, cv = None, verbose = 0, pre_dispatch = '2 * n_jobs', error_score = 'raise',
return_train_score = 'warn')
```

GridSearchCV类构造方法各参数如下。

1) estimator

选择使用的分类器，并且传入除需要确定最佳的参数之外的其他参数。每一个分类器都需要一个scoring参数或者score方法，举例如下。

```
estimator = RandomForestClassifier(min_sample_split = 100,min_samples_leaf = 20,max_
depth = 8,max_features = 'sqrt', random_state = 10)
```

2) param_grid

需要最优化的参数的取值,值为字典或者列表,举例如下。

```
param_grid = {'kernel': ['linear', 'rbf'], 'gamma': [0.001,0.01,1,10,100], 'C': [0.001,
0.01,1,10,100]}
```

3) scoring

模型评价标准,默认为 None,这时需要使用 score 函数;根据所选模型不同,评价准则不同,字符串(函数名)或可调用对象需要其函数签名,形如 scorer(estimator,X,y);如果是 None,则使用 estimator 的误差估计函数。

4) fit_params

该参数通常取 None。

5) n_jobs

n_jobs:并行数。n_jobs 为-1 表示跟 CPU 核数一致,默认值为 1。

6) iid

iid:默认为 True,为 True 时,默认为各个样本 fold 概率分布一致,误差估计为所有样本之和,而非各个 fold 的平均。

7) refit

默认为 True,程序将会以交叉验证训练集得到的最佳参数,重新对所有可能的训练集与开发集进行,作为最终用于性能评估的最佳模型参数。即在搜索参数结束后,用最佳参数结果再次 fit 一遍全部数据集。

8) cv

交叉验证参数,默认为 None,使用 3 折交叉验证。指定 fold 数量,默认为 3,也可以是 yield 训练/测试数据的生成器。

9) verbose

verbose:日志冗长度。为 0 表示不输出训练过程,为 1 表示偶尔输出,大于 1 表示对每个子模型都输出。

10) pre_dispatch

指定总的并行任务数,当 n_jobs 大于 1 时,数据将在每个运行点进行复制,这可能导致 OOM,而设置 pre_dispatch 参数,则可以预先划分总的任务数量,使数据最多被复制 pre_dispatch 次。

11) error_score

该参数通常取 raise。

12) return_train_score

该参数通常取 warn。如果取 False,cv_results_属性将不包括训练分数。

3. GridSearchCV 对象属性说明

1) cv_results

具有键作为列标题和值作为列的字典，可以导入 DataFrame 中，params 键用于存储所有参数候选项的参数设置列表。

2) best_estimator

通过搜索选择的估计器，即在左侧数据上给出最高分数(或指定的最小损失)的估计器。如果 refit = False，则该属性不可用。

3) best_score

best_estimator 的交叉验证平均分数。

4) best_params_

在保存数据上给出最佳结果的参数设置。

5) best_index_

对应于最佳候选参数设置的索引。

6) scorer_

选出最佳参数的估计器所使用的评分器。

7) n_splits

交叉验证时折叠或迭代的数量。

8) refit_time

在整个数据集的基础上选出最佳模型所耗费的时间。

4. GridSearchCV 对象的方法

1) decision_function(X)

使用找到的参数最好的分类器调用 decision_function，仅当 refit 参数为 True 且估计器有 decision_function 方法时可用。

2) fit(X, y=None, groups=None, ** fit_params)

遍历所有参数组合，对模型进行训练。

3) get_params(deep=True)

获取估计器的参数。

4) predict(self, X)

用找到的最佳参数预测模型结果，仅当 refit 参数为 True 且估计器含有 predict 方法时才可用。

5) predict_log_proba(X)

调用最佳模型的 predict_log_proba 方法，仅当 refit 参数为 True 且估计器有 predict_log_proba 方法时才可用。

6) predict_proba(X)

调用最佳模型的 predict_proba 方法，仅当 refit 参数为 True 且估计器有 predict_proba 方法时才可用。

7) score(X, y=None)

如果预估器已经选出最优的分类器,则返回给定数据集的得分。

8) set_params(** params)

设置模型的参数。

9) transform(X)

调用最优分类器对 X 进行转换,仅当 refit 参数为 True 且估计器有 transform 方法时才可用。

要使用 GridSearchCV 类,首先需要用一个字典指定要搜索的参数名称,字典的值是想要尝试的参数设置。如果 C 和 gamma 想要尝试的取值为 0.001,0.01,0.1,1,10 和 100, kernel 想要尝试 linear 和 rbf,可以将其转化为下面的字典。

下面来调节 SVM 分类器的 C、kernel、gamma 参数,完整的代码如下。

```
In[61]:
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(
    iris.data, iris.target, random_state=0)

param_grid = { 'C': [0.001, 0.01, 1, 10, 100], 'gamma': [0.001, 0.01, 1, 10, 100], '
kernel': ['linear', 'rbf']}
grid_search_svc = GridSearchCV(estimator=SVC(),
param_grid=param_grid, scoring='accuracy', cv=10, n_jobs=-1)
grid_search_svc = grid_search_svc.fit(X_train, y_train)

print('Best score on validation set :{:.2f}'.format(grid_search_svc.best_score_))
print('Best parameters:{}'.format(grid_search_svc.best_params_))
print('Best score on test set:{:.2f}'.format(grid_search_svc.score(X_test, y_test)))
print("Best estimator:\n{}".format(grid_search_svc.best_estimator_))
Out[61]:
Best score on validation set :0.98
Best parameters:{'C': 1, 'gamma': 0.001, 'kernel': 'linear'}
Best score on test set:0.97
Best estimator:
SVC(C=1, gamma=0.001, kernel='linear')
```

上例中划分数据集、运行网格搜索并评估最终参数的完整过程如图 5.21 所示。

```
In[62]:
mglearn.plots.plot_grid_search_overview()
Out[62]:
```

网格搜索的结果可以在 cv_results_属性中找到(sklearn 2.0 版本以下用 grid_scores_属

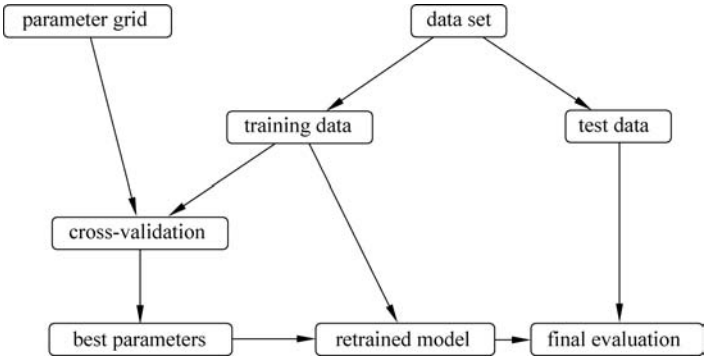

图 5.21　划分数据集、运行网格搜索并评估最终参数的完整过程

性查看),它是一个字典,保存了搜索的所有内容,代表搜索的整个过程,其输出如表 5.2 所示。

```
In[63]:
import pandas as pd
results = pd.DataFrame(grid_search_svc.cv_results_)
display(results)
Out[63]:
```

表 5.2　cv_results_属性的值

id	mean_fit_time	…	params	split0_test_score	…	split4_test_score	mean_test_score	…
0	0.0018	…	{'C': 0.001, 'gamma': 0.001, 'kernel': 'linear'}	0.347826	…	0.409091	0.366403	…
1	0.002	…	{'C': 0.001, 'gamma': 0.001, 'kernel': 'rbf'}	0.347826	…	0.409091	0.366403	…
2	0.001	…	{'C': 0.001, 'gamma': 0.01, 'kernel': 'linear'}	0.347826	…	0.409091	0.366403	…
…	…	…	…	…	…	…	…	…
22	**0**	…	{**'C': 1, 'gamma': 0.01, 'kernel': 'linear'**}	**1**	…	**0.954545**	**0.973123**	…
…	…	…	…	…	…	…	…	…
47	0.0014	…	{'C': 100, 'gamma': 10, 'kernel': 'rbf'}	0.869565	…	0.954545	0.911067	…
48	0.0008	…	{'C': 100, 'gamma': 100, 'kernel': 'linear'}	0.956522	…	0.954545	0.955336	…
49	0.0016	…	{'C': 100, 'gamma': 100, 'kernel': 'rbf'}	0.521739	…	0.681818	0.581423	…

从输出结果可以看出,要想使用 5 折交叉验证对 C、gamma 以及 kernel 特定取值的

SVM 的精度进行评估，总共需要训练 5×5×2 = 50 轮。每一轮尝试其中一种 C、gamma 以及 kernel 的组合，并且每一轮中需要交叉验证 5 次。最终从中找出在验证集上平均得分最高的参数组合，即第 23 轮中(用灰色表示)验证集上平均最高得分 0.973123，参数组合为{'C': 1, 'gamma': 0.01, 'kernel': 'linear'}。

5.6 本章小结

本章首先讨论了算法链与管道。管道可以理解为一个容器，然后把需要进行的操作封装在其中进行操作，比如数据标准化、特征降维、主成分分析、模型预测等。Pipeline 类不但可用于预处理和分类，还可以将任意数量的估计器连接，极大地方便了使用。

接着讨论了交叉验证、模型评价指标以及处理类的不平衡问题。交叉验证主要用于防止模型过于复杂而引起的过拟合问题。模型评价指标是评估模型的泛化能力，这是机器学习中的一个关键性的问题。评价指标的作用是了解模型的泛化能力，通过这些指标来逐步优化模型。在实际应用中，分类问题很少会遇到平衡的类别，因此需要了解这些分类不平衡的后果，并选择相应的评估指标。

最后讨论了在机器学习中如何通过网格搜索对模型调优。调优的过程就是寻找超参数的过程，如果超参数选择不恰当，模型就会出现欠拟合或者过拟合的问题。尽管将数据集划分为训练集、验证集和测试集的方法相对有用、可行性较高，但是该方法对数据的划分比较敏感，也就是说其最终的表现好坏与初始数据的划分结果有很大的关系，且有时候泛化性能较低。为了更好地提高模型的泛化能力，最好选择带交叉验证的网格搜索来评估每种组合的性能并以此来降低偶然性。

习题

1. 什么是算法链和管道？它们有什么作用？

2. 为什么要进行交叉验证？

3. K 折交叉验证、分层 K 折交叉验证、留一法交叉验证、打乱划分交叉验证以及分组交叉验证之间有什么区别？

4. 为什么要对模型进行评价？

5. 解释一下什么是混淆矩阵。

6. 举例说明分类问题的评价指标都有哪些。

7. 举例说明回归问题的评价指标都有哪些。

8. 用 GridSearchCV 对 5.3.8 节 In[53]的逻辑回归模型进行调优，并输出相关评价指标及混淆矩阵。

9. 将 5.5.2 节 In[59]中 random_state 设置为 117，观察输出情况，并分析造成这种结果的原因是什么。