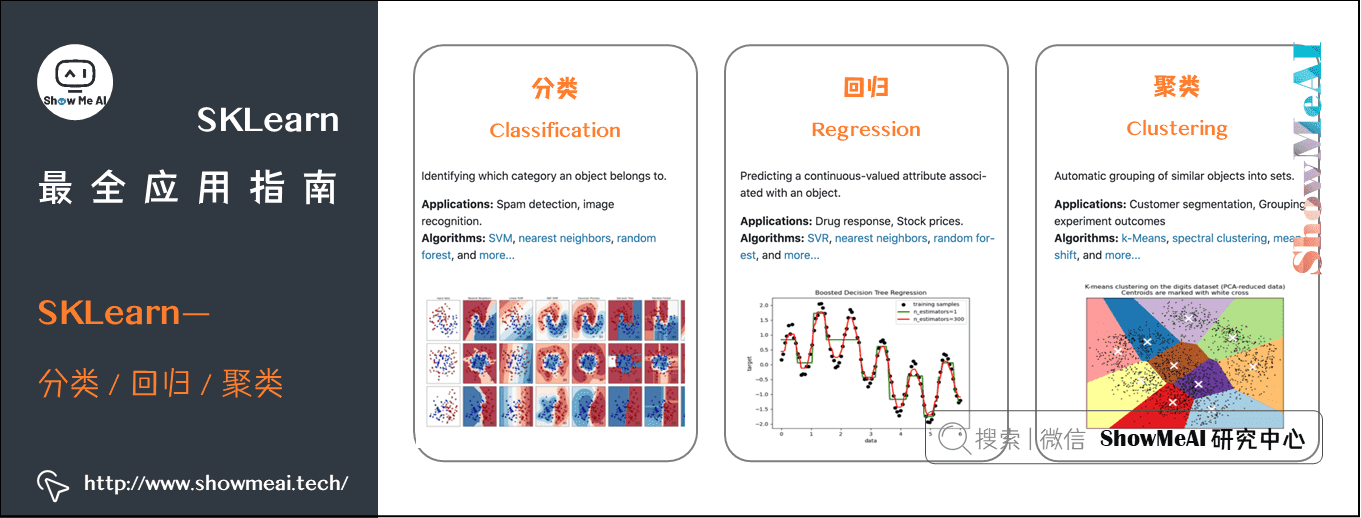
[**机器学习实战 | SKLearn最全应用指南**](https://www.cnblogs.com/showmeai/p/16037165.html)

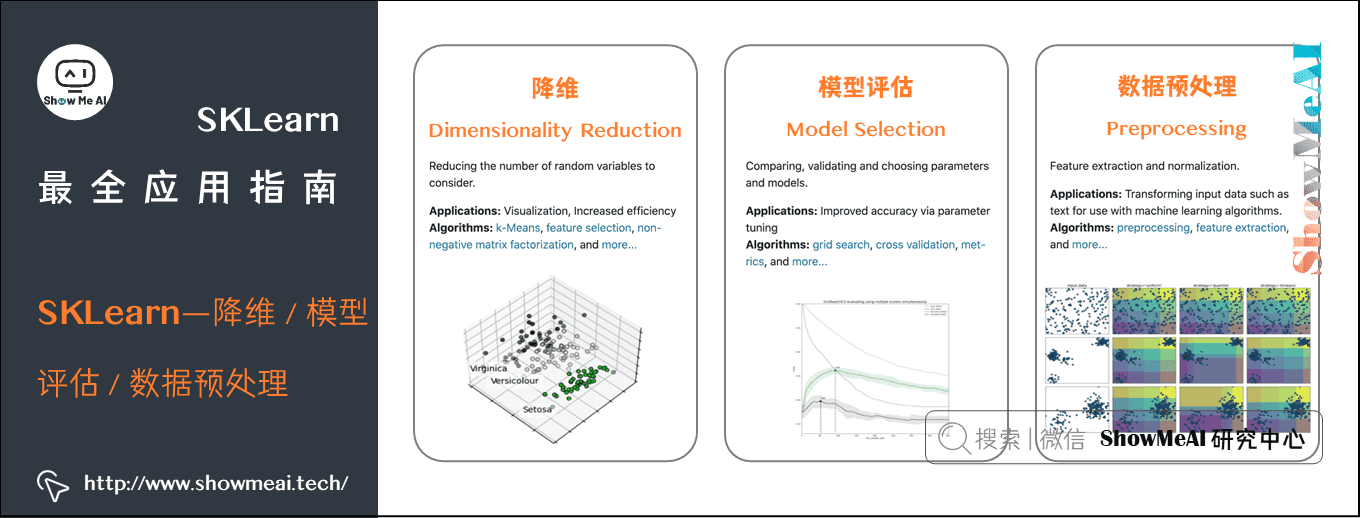
[教程地址](https://www.showmeai.tech/tutorials/41)：<https://www.showmeai.tech/tutorials/41>  
[本文地址](https://www.showmeai.tech/article-detail/203)：<https://www.showmeai.tech/article-detail/203>  
声明：版权所有，转载请联系平台与作者并注明出处  
收藏[ShowMeAI](https://www.showmeai.tech/)查看更多精彩内容

**引言**

我们在上一篇[**SKLearn入门与简单应用案例**](https://www.showmeai.tech/article-detail/202)里给大家讲到了SKLearn工具的基本板块与使用方法，在本篇内容中，我们展开讲解SKLearn的进阶与核心内容。SKLearn中有六大任务模块，如下图所示：分别是分类、回归、聚类、降维、模型选择和预处理。

* SKLearn官网：<https://scikit-learn.org/stable/>
* SKLearn的快速使用方法也推荐大家查看[ShowMeAI](https://www.showmeai.tech/)的文章和速查手册 [**AI建模工具速查|Scikit-learn使用指南**](https://www.showmeai.tech/article-detail/108)





在SKLearn中，因为做了上层的封装，分类模型、回归模型、聚类与降维模型、预处理器等等都叫做估计器(estimator)，就像在Python里「万物皆对象」，在SKLearn里「万物皆估计器」。

在本篇内容中，我们将给大家进一步深入讲解scikit-learn工具库的使用方法，力求完整覆盖sklearn工具库应用的方方面面。本文的内容板块包括：

* ① **机器学习基础知识**：机器学习定义与四要素：数据、任务、性能度量和模型。机器学习概念，以便和SKLearn对应匹配上。
* ② **SKLearn讲解**：API设计原理，sklearn几大特点：一致性、可检验、标准类、可组合和默认值，以及SKLearn自带数据以及储存格式。
* ③ **SKLearn三大核心API讲解**：包括估计器、预测器和转换器。这个板块很重要，大家实际应用时主要是借助于核心API落地。
* ④ **SKLearn高级API讲解**：包括简化代码量的流水线(Pipeline估计器)，集成模型(Ensemble估计器)、有多类别-多标签-多输出分类模型(Multiclass 和 Multioutput 估计器)和模型选择工具(Model Selection估计器)。

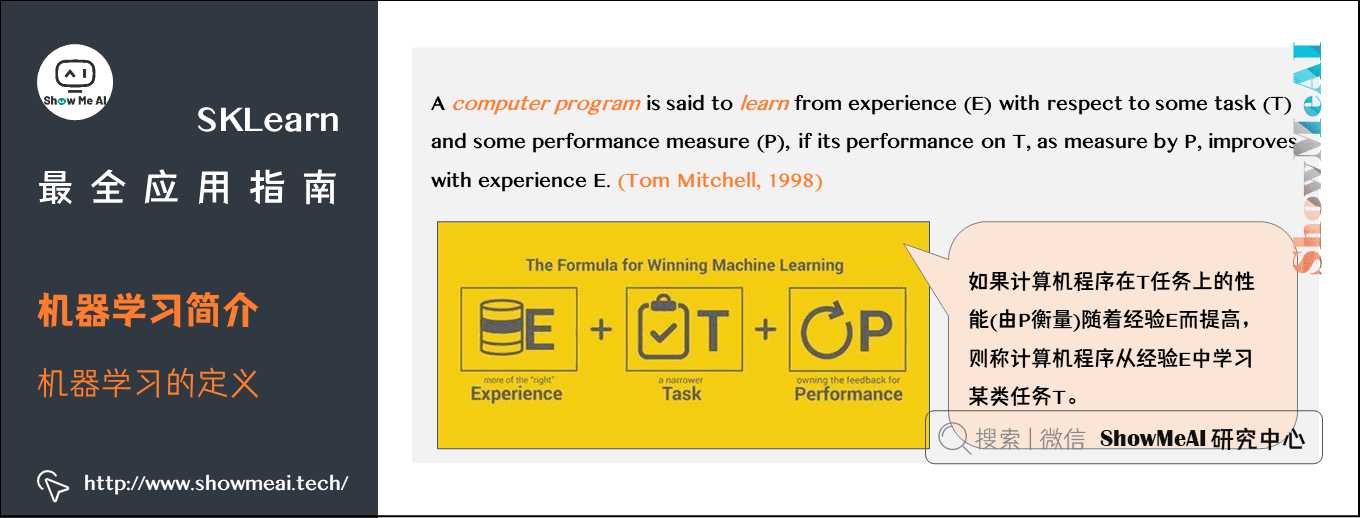
**1.机器学习简介**

关于本节内容，强烈推荐大家阅读[ShowMeAI](https://www.showmeai.tech/)文章 [**图解机器学习 | 机器学习基础知识**](https://www.showmeai.tech/article-detail/185) 和 [**图解机器学习 | 模型评估方法与准则**](https://www.showmeai.tech/article-detail/186) ，[ShowMeAI](https://www.showmeai.tech/)对相关知识内容展开做了详细讲解。

**1.1 定义和构成元素**

何为机器学习？大师汤姆米切尔（Tom Mitchell）对机器学习定义的原话是：

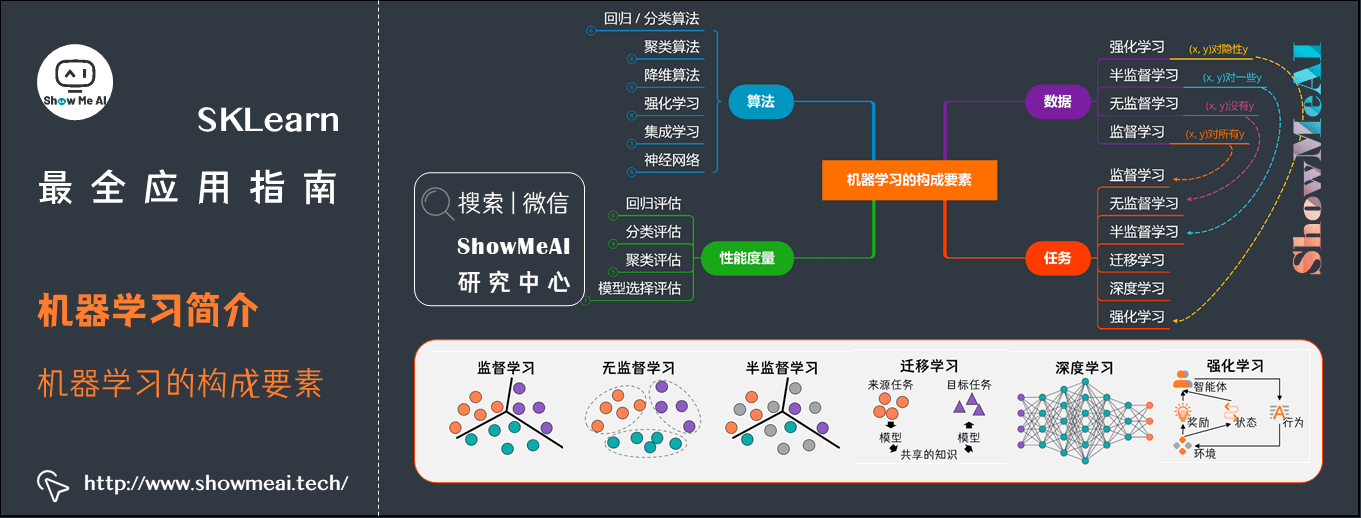
A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E.



这段英文中有两个词computer program和learn，翻译成中文就是机器(计算机程序)和学习，整体翻译下来就是说：如果计算机程序在T任务上的性能(由P衡量)随着经验E而提高，则称计算机程序从经验E中学习某类任务T。

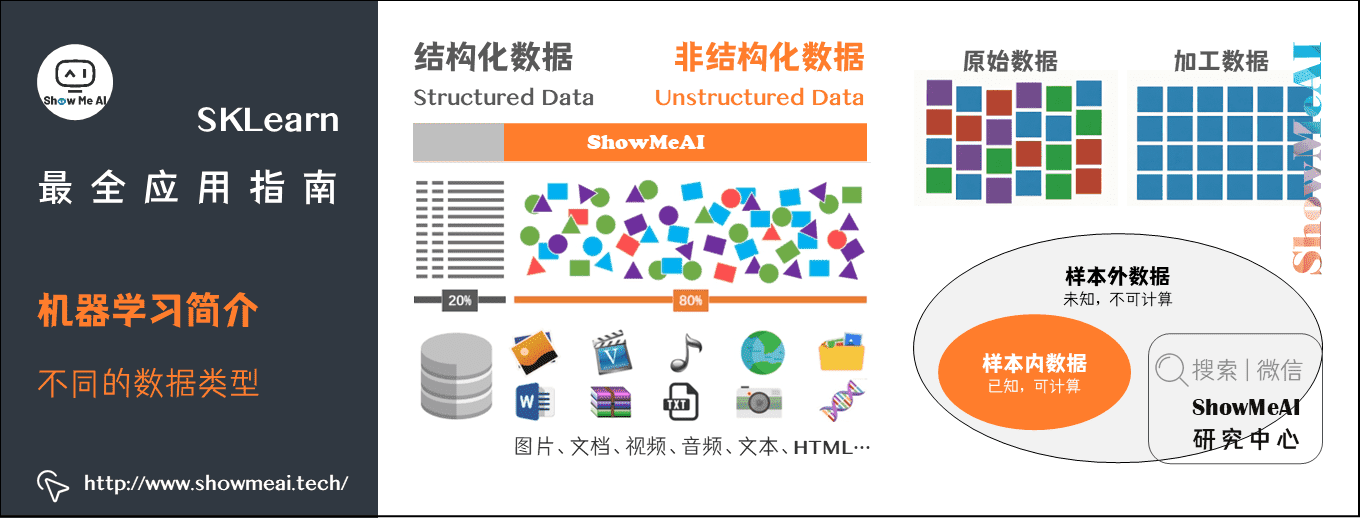
由上述机器学习的定义可知机器学习包含四个元素：

* 数据(Data)
* 任务(Task)
* 性能度量(Quality Metric)
* 算法(Algorithm)



**1.2 数据**

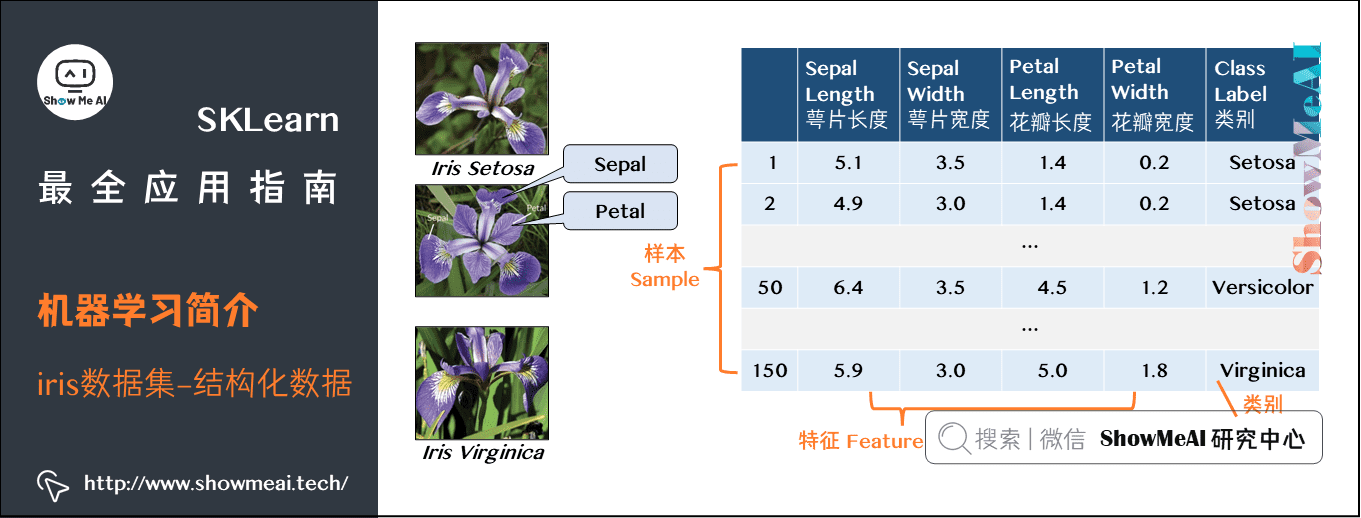
数据(data)是信息的载体。数据可以有以下划分方式：



* 从「**数据具体类型**」维度划分：结构化数据和非结构化数据。
  + 结构化数据(structured data)是由二维表结构来逻辑表达和实现的数据。
  + 非结构化数据是没有预定义的数据，不便用数据库二维表来表现的数据。非结构化数据包括图片，文字，语音和视频等。
* 从「**数据表达形式**」维度划分：原始数据和加工数据。
* 从「**数据统计性质**」维度划分：样本内数据和样本外数据。

对于非结构数据，通常神经网络有更好的效果，可以参考[ShowMeAI](https://www.showmeai.tech/)的文章[Python机器学习算法实践](https://www.cnblogs.com/showmeai/p/16037165.html)中的图像建模例子。

机器学习模型很多时候使用的是结构化数据，即二维的数据表。我们这里以iris花瓣数据集举例，如下图。



下面术语大家在深入了解机器学习前一定要弄清楚：

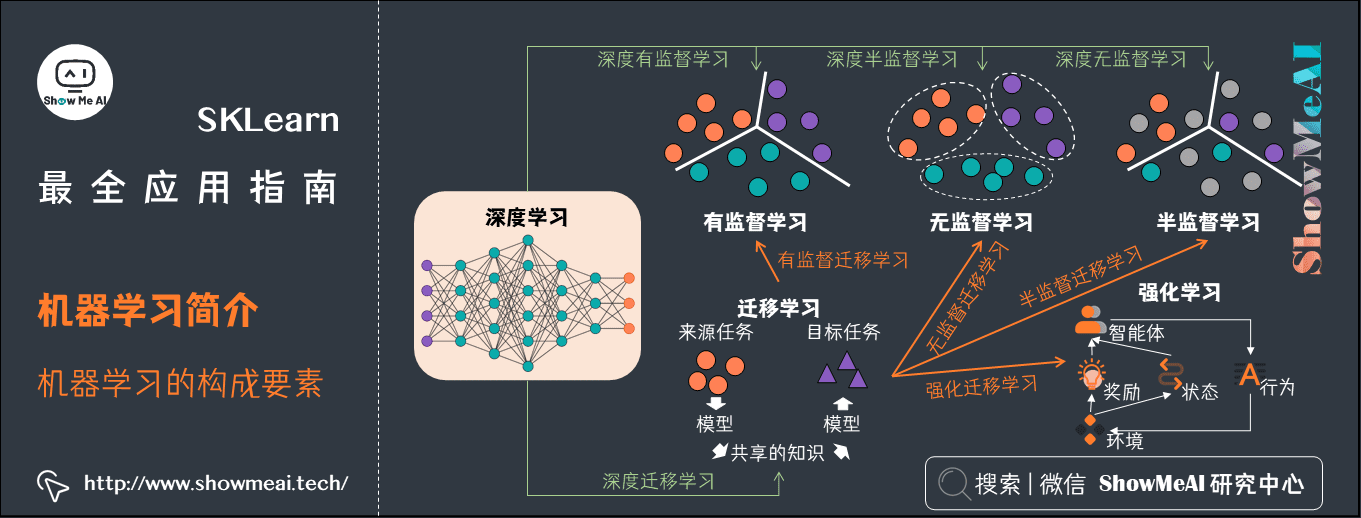
* 每行的记录(这是一朵鸢尾花的数据统计)，称为一个「**样本(sample)**」。
* 反映样本在某方面的性质，例如萼片长度(Sepal Length)、花瓣长度(Petal Length)，称为「**特征(feature)**」。
* 特征上的取值，例如「样本1」对应的5.1、3.5称为「**特征值(feature value)**」。
* 关于样本结果的信息，例如Setosa、Versicolor，称为「**类别标签(class label)**」。
* 包含标签信息的示例，则称为「**样例(instance)**」，即样例=(特征,标签)。
* 从数据中学得模型的过程称为「**学习(learning)**」或「**训练(training)**」。
* 在训练数据中，每个样例称为「**训练样例(training instance)**」，整个集合称为「**训练集(training set)**」。

**1.3 任务**

根据学习的任务模式(训练数据是否有标签)，机器学习可分为几大类：

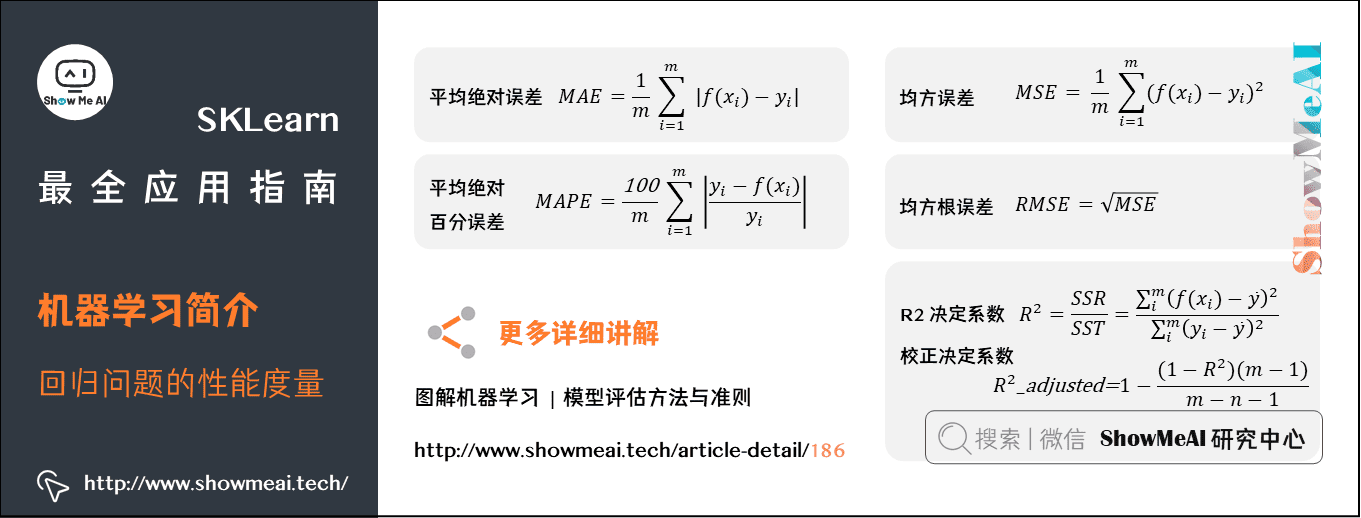
* 监督学习(有标签)
* 无监督学习(无标签)
* 半监督学习(有部分标签)
* 强化学习(有延迟的标签)

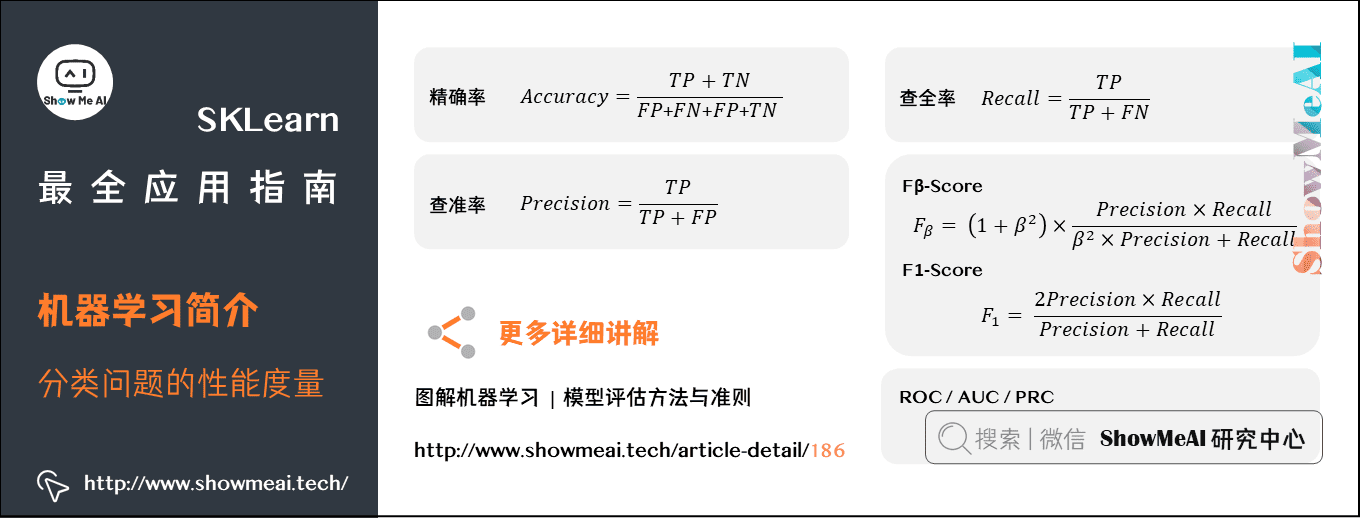
下图画出机器学习各类之间的关系。



**1.4 性能度量**

回归和分类任务中最常见的误差函数以及一些有用的性能度量如下，详细内容可以参考[ShowMeAI](https://www.showmeai.tech/)文章 [**机器学习评估与度量准则**](https://www.showmeai.tech/article-detail/186)。





**2. SKLearn数据**

SKLearn作为通用机器学习建模的工具包，包含六个任务模块和一个数据导入模块：

* [**监督学习：分类任务**](https://scikit-learn.org/stable/supervised_learning.html#supervised-learning)
* [**监督学习：回归任务**](https://scikit-learn.org/stable/supervised_learning.html#supervised-learning)
* [**无监督学习：聚类任务**](https://scikit-learn.org/stable/modules/clustering.html#clustering)
* [**无监督学习：降维任务**](https://scikit-learn.org/stable/modules/decomposition.html#decompositions)
* [**模型选择任务**](https://scikit-learn.org/stable/model_selection.html#model-selection)
* [**数据预处理任务**](https://scikit-learn.org/stable/modules/preprocessing.html#preprocessing)
* [**数据导入模块**](https://scikit-learn.org/stable/datasets.html)

首先看看SKLearn默认数据格式和自带数据集。

**2.1 SKLearn默认数据格式**

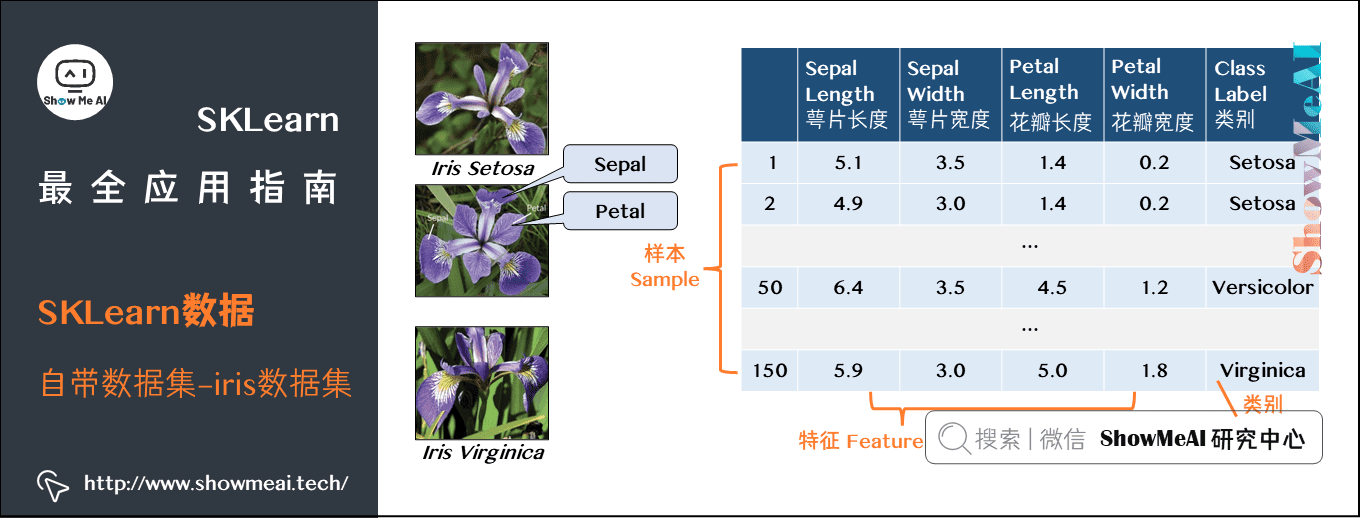
Sklean里模型能直接使用的数据有两种形式：

* Numpy二维数组(ndarray)的稠密数据(dense data)，通常都是这种格式。
* SciPy矩阵(scipy.sparse.matrix)的稀疏数据(sparse data)，比如文本分析每个单词(字典有100000个词)做独热编码得到矩阵有很多0，这时用ndarray就不合适了，太耗内存。

**2.2 自带数据集**

SKLearn里面有很多自带数据集供用户使用。

比如在之前文章[**Python机器学习算法实践**](https://www.showmeai.tech/article-detail/201)中用到的鸢尾花数据集，包含四个特征(萼片长/宽和花瓣长/宽)和三个类别。



我们可以直接从SKLearn里面的datasets模块中引入，代码如下(代码可以在 [线上Jupyter环境](https://jupyter.org/try) 中运行)：

|  |  |
| --- | --- |
|  | # 导入工具库 |
|  | from sklearn.datasets import load\_iris |
|  | iris = load\_iris() |
|  |  |
|  | #数据是以「字典」格式存储的，看看 iris 的键有哪些。 |
|  | iris.keys() |

输出如下：

|  |  |
| --- | --- |
|  | dict\_keys(['data', 'target', 'target\_names', 'DESCR', 'feature\_names', 'filename']) |

读取数据集的信息：

|  |  |
| --- | --- |
|  | #输出iris 数据中特征的大小、名称等信息和前五个样本。 |
|  | n\_samples, n\_features = iris.data.shape |
|  | print((n\_samples, n\_features)) |
|  | print(iris.feature\_names) |
|  | print(iris.target.shape) |
|  | print(iris.target\_names) |
|  | iris.data[0:5] |

输出如下：

|  |  |
| --- | --- |
|  | (150, 4) |
|  | ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)'] |
|  | (150,) |
|  | ['setosa' 'versicolor' 'virginica'] |
|  | array([[5.1, 3.5, 1.4, 0.2], |
|  | [4.9, 3. , 1.4, 0.2], |
|  | [4.7, 3.2, 1.3, 0.2], |
|  | [4.6, 3.1, 1.5, 0.2], |
|  | [5. , 3.6, 1.4, 0.2]]) |

构建Dataframe格式的数据集：

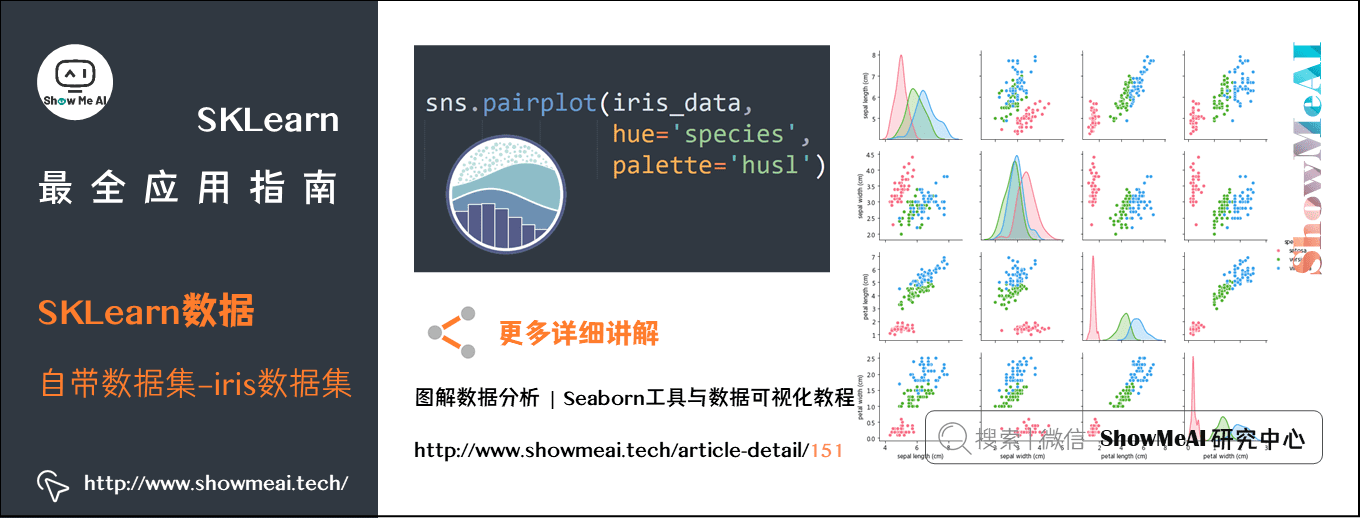
|  |  |
| --- | --- |
|  | # 将X和y合并为Dataframe格式数据 |
|  | import pandas as pd |
|  | import seaborn as sns |
|  | iris\_data = pd.DataFrame( iris.data, |
|  | columns=iris.feature\_names ) |
|  | iris\_data['species'] = iris.target\_names[iris.target] |
|  | iris\_data.head(3).append(iris\_data.tail(3)) |

输出如下：

|  | **sepal length (cm)** | **sepal width (cm)** | **petal length (cm)** | **petal width (cm)** | **species** |
| --- | --- | --- | --- | --- | --- |
| 0 | 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| 1 | 4.9 | 3.0 | 1.4 | 0.2 | setosa |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 | setosa |
| 147 | 6.5 | 3.0 | 5.2 | 2.0 | virginica |
| 148 | 6.2 | 3.4 | 5.4 | 2.3 | virginica |
| 149 | 5.9 | 3.0 | 5.1 | 1.8 | virginica |

我们使用seaborn来做一些数据分析，查看一下数据的分布特性。这里使用到的是成对维度的关联分析，关于seaborn的使用方法可以参阅[ShowMeAI](https://www.showmeai.tech/)的文章 [**seaborn工具与数据可视化教程**](https://www.showmeai.tech/article-detail/151)。

|  |  |
| --- | --- |
|  | # 使用Seaborn的pairplot查看两两特征之间的关系 |
|  | sns.pairplot( iris\_data, hue='species', palette='husl' ) |



**2.3 数据集引入方式**

前面提到的是鸢尾花iris数据集，我们通过load\_iris加载进来，实际上SKLearn有三种引入数据形式。

* 打包好的数据：对于小数据集，用sklearn.datasets.load\_\*
* 分流下载数据：对于大数据集，用sklearn.datasets.fetch\_\*
* 随机创建数据：为了快速展示，用sklearn.datasets.make\_\*

上面这个星号\*指代具体文件名，如果大家在Jupyter这种IDE环境中，可以通过tab制表符自动补全和选择。

* datasets.load\_
* datasets.fetch\_
* datasets.make\_

比如我们调用load\_iris

|  |  |
| --- | --- |
|  | from sklearn import datasets |
|  | datasets.load\_iris |

输出如下：

|  |  |
| --- | --- |
|  | <function sklearn.datasets.base.load\_iris(return\_X\_y=False)> |

我们调用load\_digits加载手写数字图像数据集

|  |  |
| --- | --- |
|  | digits = datasets.load\_digits() |
|  | digits.keys() |

输出：

|  |  |
| --- | --- |
|  | dict\_keys(['data', 'target', 'target\_names', 'images', 'DESCR']) |

我们再来看看通过fetch拉取数据的示例：

|  |  |
| --- | --- |
|  | #加州房屋数据集 |
|  | california\_housing = datasets.fetch\_california\_housing() |
|  | california\_housing.keys() |

输出：

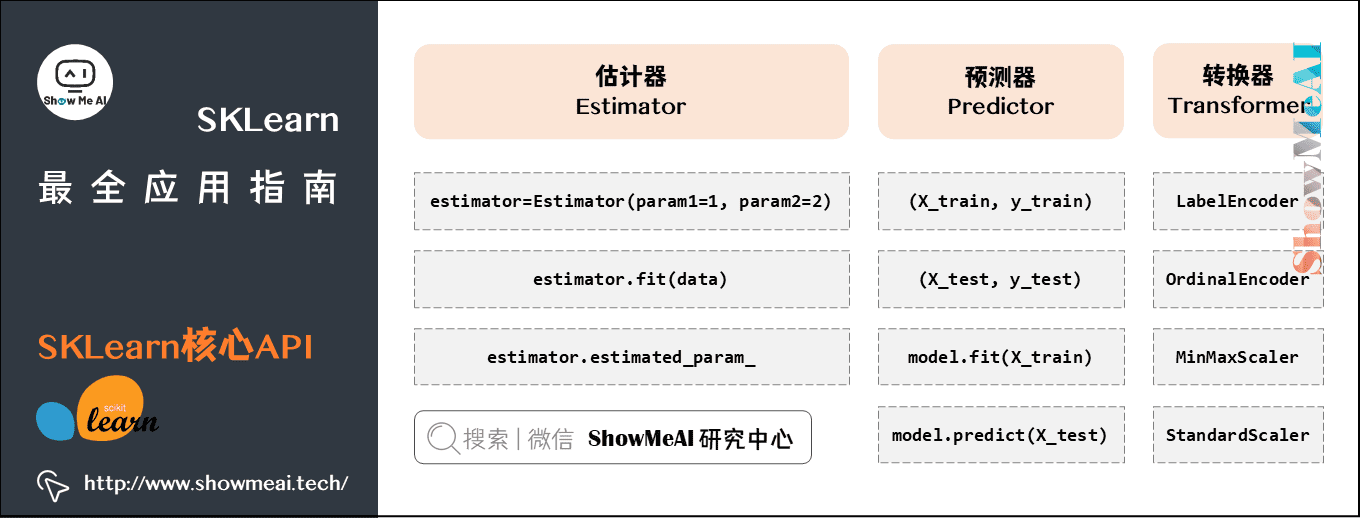
|  |  |
| --- | --- |
|  | dict\_keys(['data', 'target', 'feature\_names', 'DESCR']) |

**3.SKLearn核心API**

我们前面提到SKLearn里万物皆估计器。估计器是个非常抽象的叫法，不严谨的一个理解，我们可以视其为一个模型(用来回归、分类、聚类、降维)，或一套流程(预处理、网格搜索交叉验证)。

本节三大API其实都是估计器：

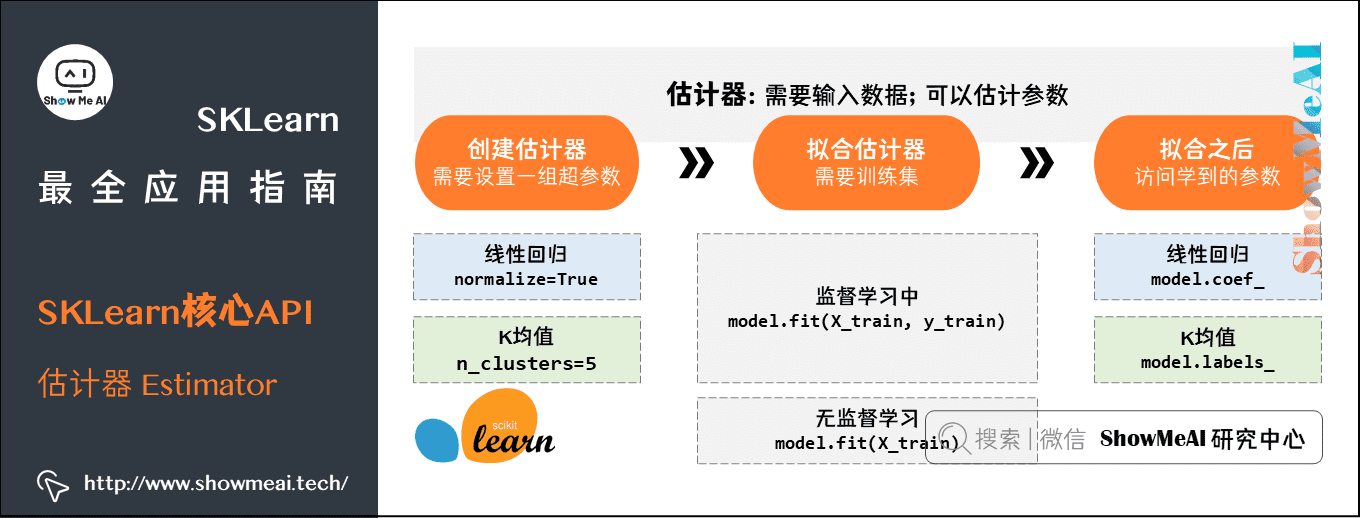
* **估计器**(**estimator**)通常是用于拟合功能的估计器。
* **预测器**(**predictor**)是具有预测功能的估计器。
* **转换器**(**transformer**)是具有转换功能的估计器。



**3.1 估计器**

任何可以基于数据集对一些参数进行估计的对象都被称为估计器，它有两个核心点：

* ① 需要输入数据。
* ② 可以估计参数。



估计器首先被创建，然后被拟合。

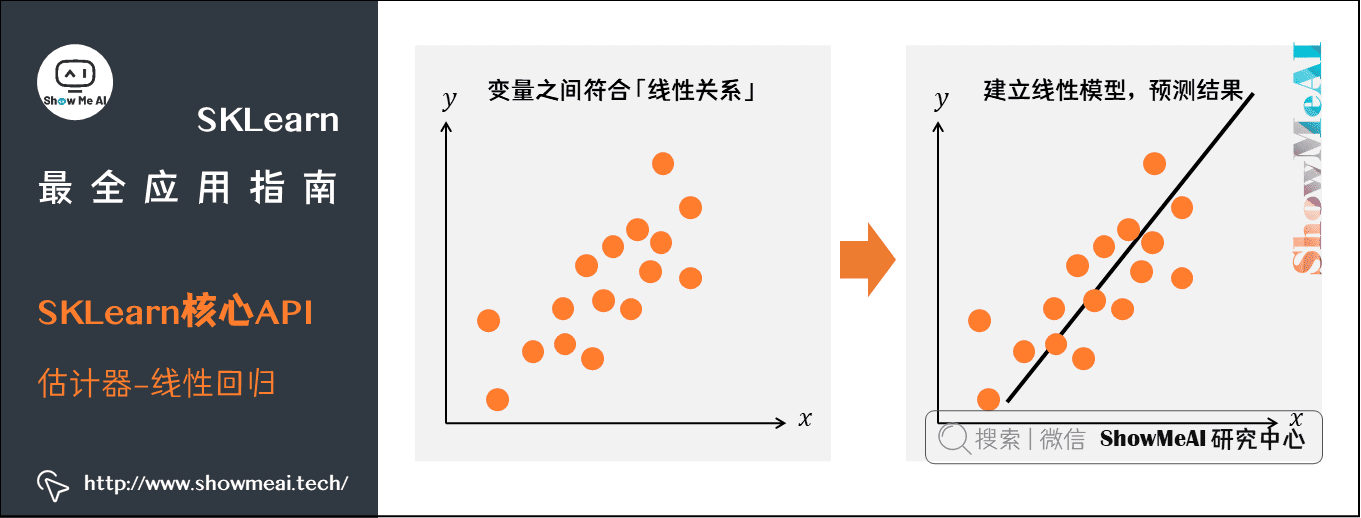
* **创建估计器：需要设置一组超参数，比如**
  + 线性回归里超参数normalize=True
  + K均值里超参数n\_clusters=5
* **拟合估计器：需要训练集**
  + 在监督学习中的代码范式为model.fit(X\_train, y\_train)
  + 在无监督学习中的代码范式为model.fit(X\_train)

拟合之后可以访问model里学到的参数，比如线性回归里的特征系数coef，或K均值里聚类标签labels，如下(具体的可以在SKLearn文档的每个模型页查到)。

* model.coef\_
* model.labels\_

下面看看监督学习的「**线性回归**」和无监督学习的「**K均值聚类**」的具体例子。

**(1) 线性回归**



首先从SKLearn工具库的linear\_model中引入LinearRegression；创建模型对象命名为model，设置超参数normalize为True（在每个特征值上做标准化，这样能保证拟合的稳定性，加速模型拟合速度）。

|  |  |
| --- | --- |
|  | from sklearn.linear\_model import LinearRegression |
|  | model = LinearRegression(normalize=True) |
|  | model |

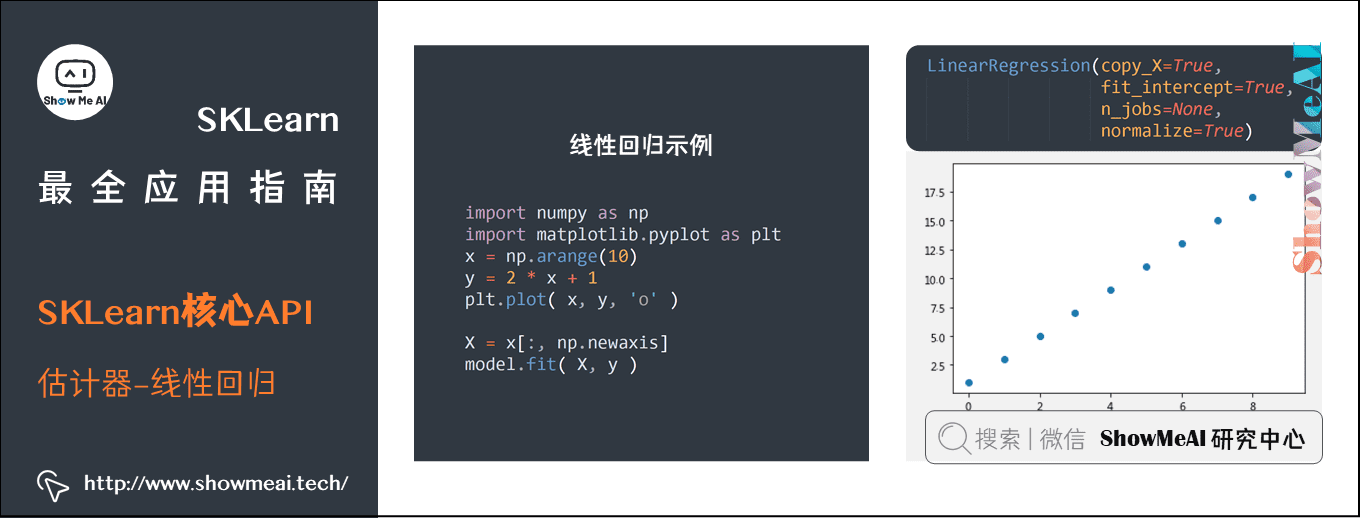
输出：

|  |  |
| --- | --- |
|  | LinearRegression(copy\_X=True, fit\_intercept=True, n\_jobs=None, normalize=True) |

创建完后的估计器会显示所有的超参数（比如刚才设置的normalize=True），未设置的超参数都使用默认值。

自己创建一个简单数据集（一条直线上的数据点），简单讲解一下估计器里面的特征。

|  |  |
| --- | --- |
|  | import numpy as np |
|  | import matplotlib.pyplot as plt |
|  | x = np.arange(10) |
|  | y = 2 \* x + 1 |
|  | plt.plot( x, y, 'o' ) |



在我们生成的数据里，X是一维，我们做一点小小的调整，用np.newaxis加一个维度，把[1,2,3]转成[[1],[2],[3]]，这样的数据形态可以符合sklearn的要求。接着把X和y送入fit()函数来拟合线性模型的参数。

|  |  |
| --- | --- |
|  | X = x[:, np.newaxis] |
|  | model.fit( X, y ) |

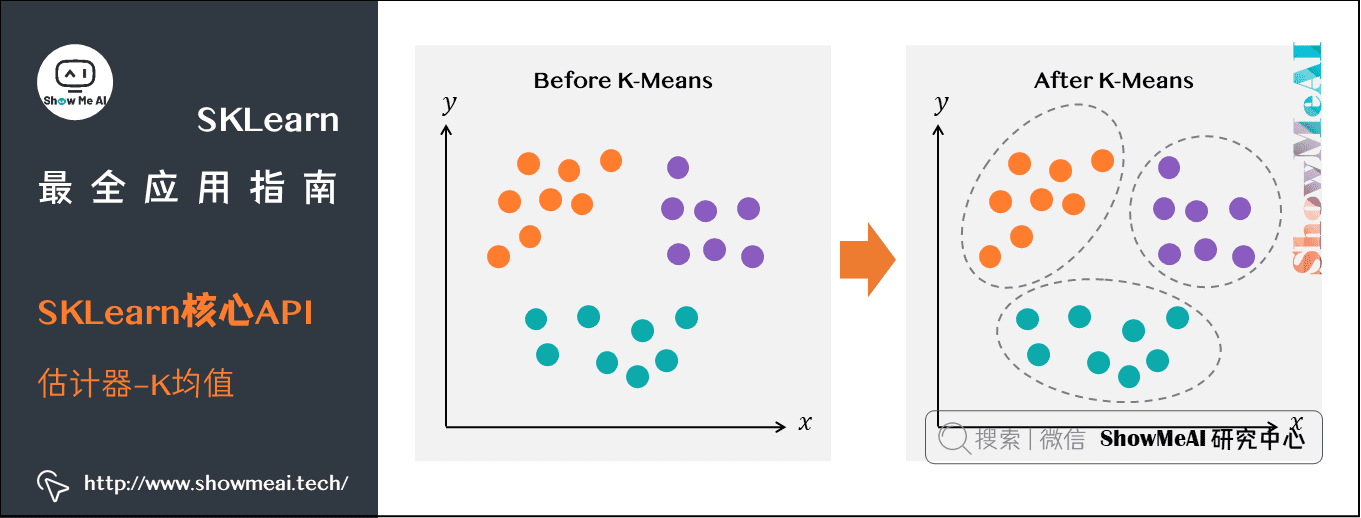
输出为：

|  |  |
| --- | --- |
|  | LinearRegression(copy\_X=True, fit\_intercept=True, n\_jobs=None, normalize=True) |

拟合完后的估计器和创建完似乎没有差别，但我们已经可以用model.param\_访问到拟合完数据的参数了，如下代码。

|  |  |
| --- | --- |
|  | print( model.coef\_ ) |
|  | print( model.intercept\_ ) |
|  | # 输出结果 |
|  | # [2.] |
|  | # 0.9999999999999982 |

**(2) K均值**



我们来看看[**聚类**](https://www.showmeai.tech/article-detail/197)的例子，先从SKLearn的cluster中导入[**KMeans**](https://www.showmeai.tech/article-detail/197)，初始化模型对象命名为model，设置超参数n\_cluster为3(为了展示方便而我们知道用的iris数据集有3类，实际上可以设置不同数量的n\_cluster)。

虽然iris数据里包含标签y，但在无监督的聚类中我们不会使用到这个信息。

|  |  |
| --- | --- |
|  | from sklearn.cluster import KMeans |
|  | model = KMeans( n\_clusters=3 ) |
|  | model |

输出为：

|  |  |
| --- | --- |
|  | KMeans(algorithm='auto', copy\_x=True, init='k-means++', max\_iter=300, |
|  | n\_clusters=3, n\_init=10, n\_jobs=None, precompute\_distances='auto', |
|  | random\_state=None, tol=0.0001, verbose=0) |

iris数据集包含四维特征(萼片长、萼片宽、花瓣长、花瓣宽)，在下面的例子中我们希望可视化，这里我们简单选取两个特征(萼片长、萼片宽)来做聚类并且可视化结果。

注意下面代码X = iris.data[:,0:2]其实就是提取特征维度。

|  |  |
| --- | --- |
|  | from sklearn.datasets import load\_iris |
|  | iris = load\_iris() |
|  | X = iris.data[:,0:2] |
|  | model.fit(X) |

输出为：

|  |  |
| --- | --- |
|  | KMeans(algorithm='auto', copy\_x=True, init='k-means++', max\_iter=300, |
|  | n\_clusters=3, n\_init=10, n\_jobs=None, precompute\_distances='auto', |
|  | random\_state=None, tol=0.0001, verbose=0) |

拟合完后的估计器和创建完似乎没有差别，但我们已经可以用model.param\_访问到拟合完数据的参数了，如下代码。

|  |  |
| --- | --- |
|  | print( model.cluster\_centers\_, '\n') |
|  | print( model.labels\_, '\n' ) |
|  | print( model.inertia\_, '\n') |
|  | print(iris.target) |
|  | [[5.77358491 2.69245283] |
|  | [6.81276596 3.07446809] |
|  | [5.006 3.428 ]] |
|  |  |
|  | [2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 |
|  | 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 0 1 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 |
|  | 1 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 0 1 1 1 1 |
|  | 1 1 0 0 1 1 1 1 0 1 0 1 0 1 1 0 0 1 1 1 1 1 0 0 1 1 1 0 1 1 1 0 1 1 1 0 1 |
|  | 1 0] |
|  |  |
|  | 37.05070212765958 |
|  |  |
|  | [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |
|  | 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 |
|  | 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 |
|  | 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 |
|  | 2 2] |

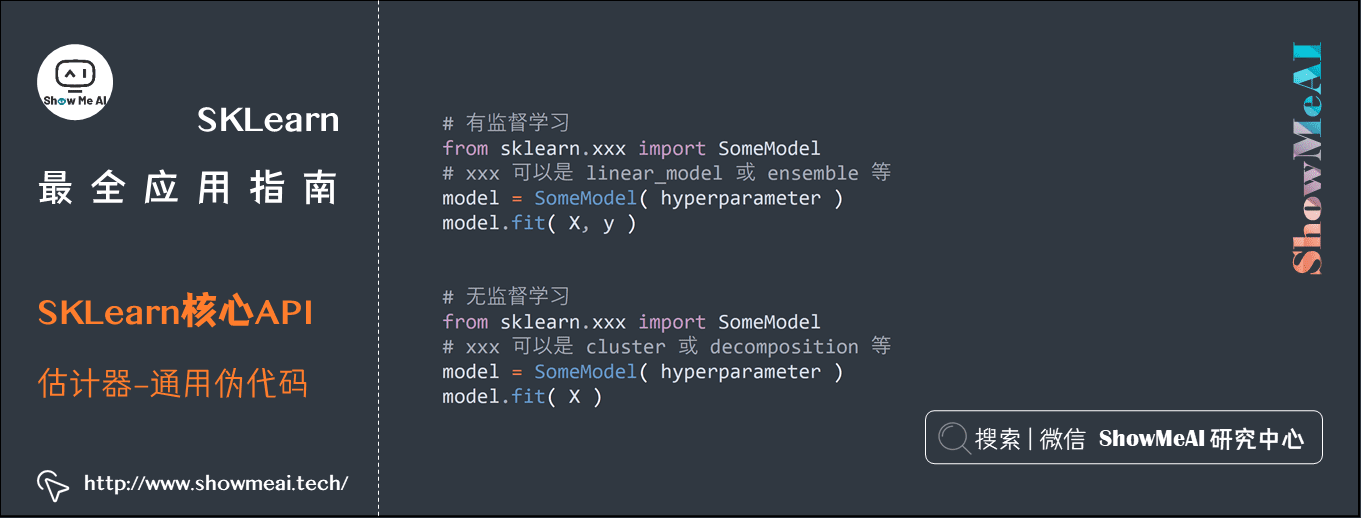
这里解释一下KMeans模型这几个参数：

* model.clustercenters：簇中心。三个簇意味着有三个坐标。
* model.labels\_：聚类后的标签。
* model.inertia\_：所有点到对应的簇中心的距离平方和(越小越好)

**小结**

虽然上面以有监督学习的Linear Regression和无监督学习的KMeans举例，但实际上你可以将它们替换成其他别的模型，比如监督学习的Logistic Regression和无监督学习的DBSCAN。它们都是「估计器」，因此都有fit()方法。

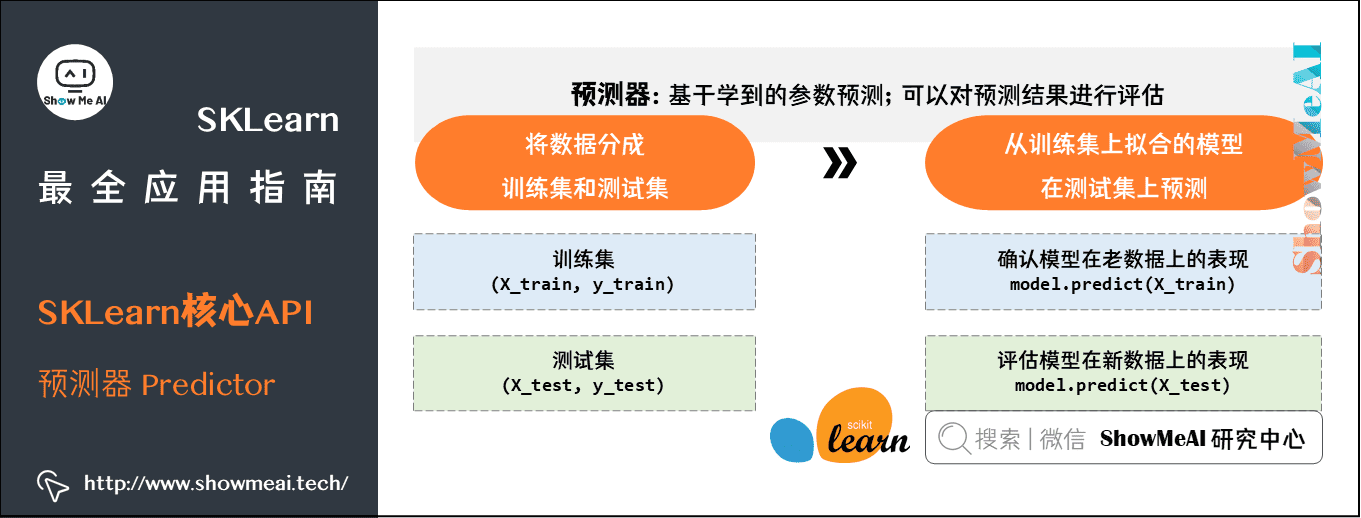
使用它们的通用伪代码如下：



|  |  |
| --- | --- |
|  | # 有监督学习 |
|  | from sklearn.xxx import SomeModel |
|  | # xxx 可以是 linear\_model 或 ensemble 等 |
|  | model = SomeModel( hyperparameter ) |
|  | model.fit( X, y ) |
|  |  |
|  | # 无监督学习 |
|  | from sklearn.xxx import SomeModel |
|  | # xxx 可以是 cluster 或 decomposition 等 |
|  | model = SomeModel( hyperparameter ) |
|  | model.fit( X ) |

**3.2 预测器**

预测器是估计器做的一个延展，具备对数据进行预测的功能。



预测器最常见的是predict()函数：

* model.predict(X\_test)：评估模型在新数据上的表现。
* model.predict(X\_train)：确认模型在老数据上的表现。

为了进行新数据评估，我们先将数据分成80:20的训练集(X\_train, y\_train)和测试集(X\_test, y\_test)，再用从训练集上拟合fit()的模型在测试集上预测predict()。

|  |  |
| --- | --- |
|  | from sklearn.datasets import load\_iris |
|  | iris = load\_iris() |
|  | from sklearn.model\_selection import train\_test\_split |
|  | X\_train, X\_test, y\_train, y\_test = train\_test\_split( iris['data'], |
|  | iris['target'], |
|  | test\_size=0.2 ) |
|  | print( 'The size of X\_train is ', X\_train.shape ) |
|  | print( 'The size of y\_train is ', y\_train.shape ) |
|  | print( 'The size of X\_test is ', X\_test.shape ) |
|  | print( 'The size of y\_test is ', y\_test.shape ) |
|  | The size of X\_train is (120, 4) |
|  | The size of y\_train is (120,) |
|  | The size of X\_test is (30, 4) |
|  | The size of y\_test is (30,) |

**predict & predict\_proba**

对于分类问题，我们不仅想知道预测的类别是什么，有时我们还希望获取预测概率等信息。前者用 predict()，后者用predict\_proba()。

|  |  |
| --- | --- |
|  | y\_pred = model.predict( X\_test ) |
|  | p\_pred = model.predict\_proba( X\_test ) |
|  | print( y\_test, '\n' ) |
|  | print( y\_pred, '\n' ) |
|  | print( p\_pred ) |

**score & decision\_function**

预测器里还有额外的两个函数可以使用。在分类问题中：

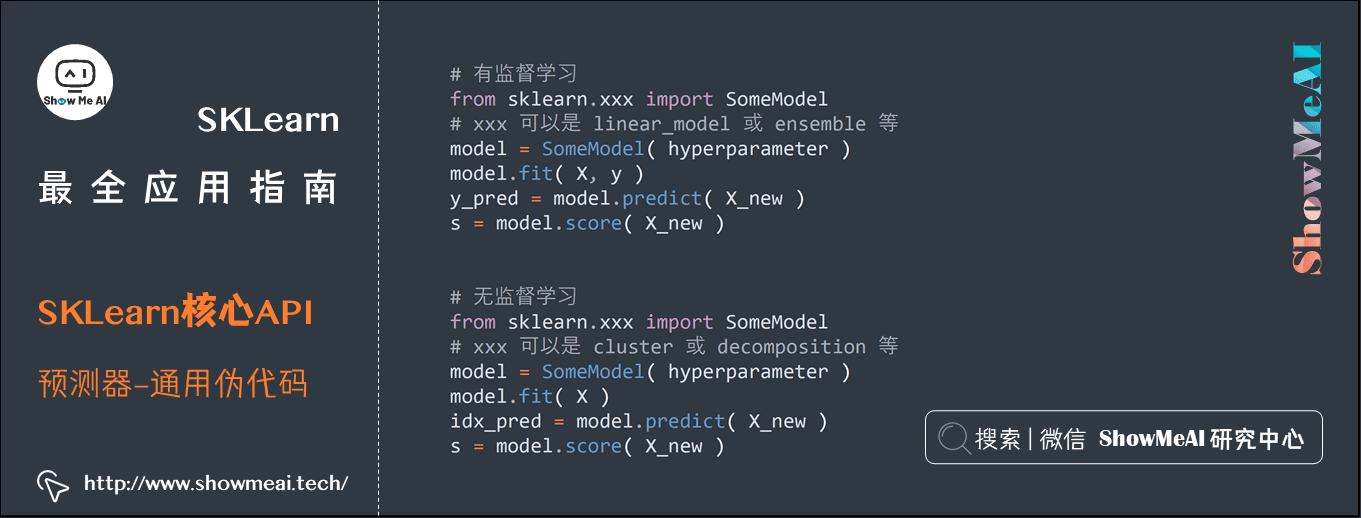
* score()返回的是分类准确率。
* decision\_function()返回的是每个样例在每个类下的分数值。

|  |  |
| --- | --- |
|  | print( model.score( X\_test, y\_test ) ) |
|  | print( np.sum(y\_pred==y\_test)/len(y\_test) ) |
|  | decision\_score = model.decision\_function( X\_test ) |
|  | print( decision\_score ) |

**小结**

估计器都有fit()方法，预测器都有predict()和score()方法，言外之意不是每个预测器都有predict\_proba()和decision\_function()方法，这个在用的时候查查官方文档就清楚了(比如RandomForestClassifier就没有decision\_function()方法)。

使用它们的通用伪代码如下：

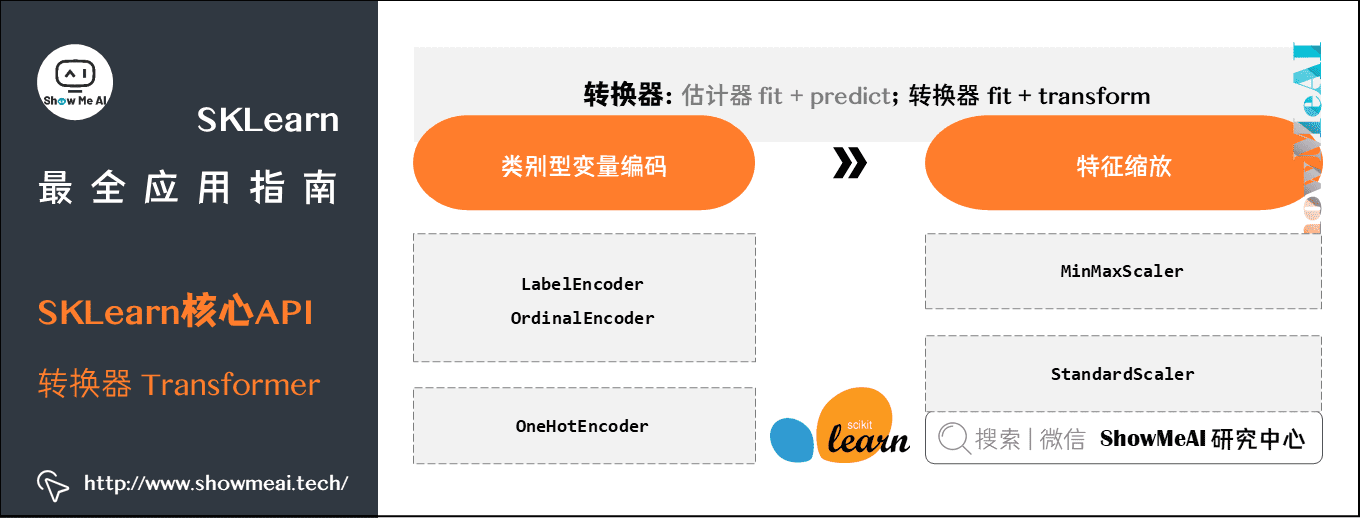


|  |  |
| --- | --- |
|  | # 有监督学习 |
|  | from sklearn.xxx import SomeModel |
|  | # xxx 可以是 linear\_model 或 ensemble 等 |
|  | model = SomeModel( hyperparameter ) |
|  | model.fit( X, y ) |
|  | y\_pred = model.predict( X\_new ) |
|  | s = model.score( X\_new ) |
|  |  |
|  | # 无监督学习 |
|  | from sklearn.xxx import SomeModel |
|  | # xxx 可以是 cluster 或 decomposition 等 |
|  | model = SomeModel( hyperparameter ) |
|  | model.fit( X ) |
|  | idx\_pred = model.predict( X\_new ) |
|  | s = model.score( X\_new ) |

**3.3 转换器**

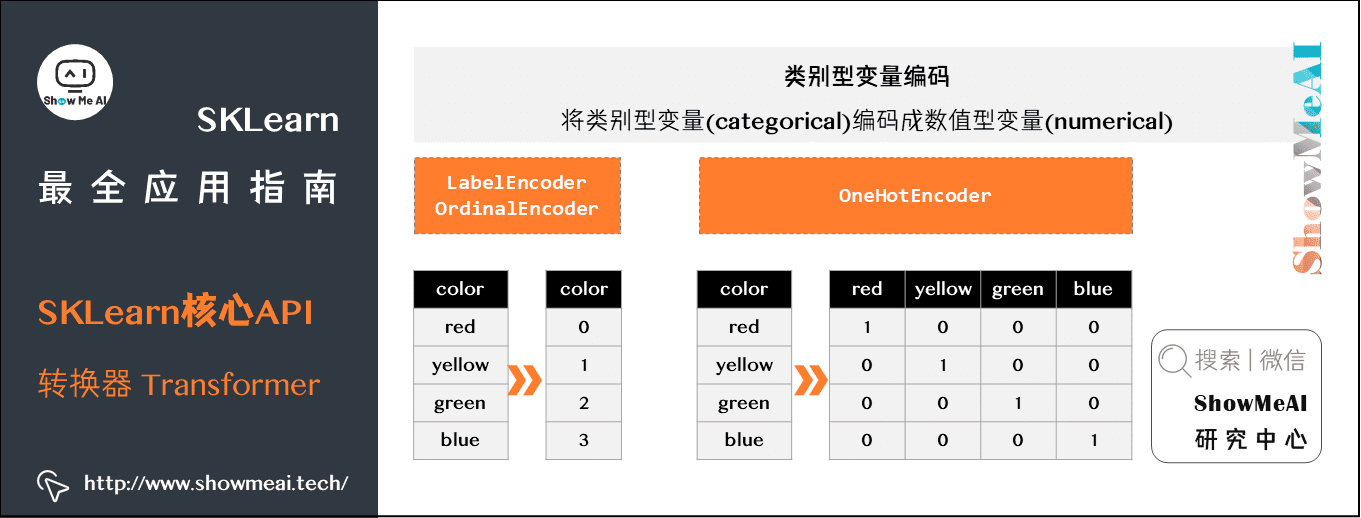
转换器是一种估计器，也有拟合功能，对比预测器做完拟合来预测，转换器做完拟合来转换。核心点如下：

* 估计器里fit + predict
* 转换器里fit + transform



本节介绍两大类转换器：

* 将类别型变量(categorical)编码成数值型变量(numerical)
* 规范化(normalize)或标准化(standardize)数值型变量



**(1) 类别型变量编码**

**① LabelEncoder&OrdinalEncoder**

LabelEncoder和OrdinalEncoder都可以将字符转成数字，但是：

* LabelEncoder的输入是一维，比如1d ndarray
* OrdinalEncoder的输入是二维，比如 DataFrame

|  |  |
| --- | --- |
|  | # 首先给出要编码的列表 enc 和要解码的列表 dec。 |
|  | enc = ['red','blue','yellow','red'] |
|  | dec = ['blue','blue','red'] |
|  |  |
|  | # 从sklearn下的preprocessing中引入LabelEncoder，再创建转换器起名LE，不需要设置任何超参数。 |
|  | from sklearn.preprocessing import LabelEncoder |
|  | LE = LabelEncoder() |
|  | print(LE.fit(enc)) |
|  | print( LE.classes\_ ) |
|  | print( LE.transform(dec) ) |
|  | LabelEncoder() |
|  | ['blue' 'yellow' 'red'] |
|  | [0 1 2] |

除了LabelEncoder，OrdinalEncoder也可以完成编码。如下代码所示：

|  |  |
| --- | --- |
|  | from sklearn.preprocessing import OrdinalEncoder |
|  | OE = OrdinalEncoder() |
|  | enc\_DF = pd.DataFrame(enc) |
|  | dec\_DF = pd.DataFrame(dec) |
|  | print( OE.fit(enc\_DF) ) |
|  | print( OE.categories\_ ) |
|  | print( OE.transform(dec\_DF) ) |
|  | OrdinalEncoder(categories='auto', dtype=<class 'numpy.float64'>) |
|  | [array(['blue', 'yellow', 'red'], dtype=object)] |
|  | [[0.] |
|  | [1.] |
|  | [2.]] |

上面这种编码的问题是，在编码过后会带来不同类别的大小关系，比如这里3种颜色其实本质上是平等的，没有大小关系。

我们的另外一种类别型数据编码方式，独热向量编码(one-hot encoding)可以解决这个问题，大家继续往下看。

**② OneHotEncoder**

独热向量编码其实就是把一个整数用向量的形式表现。上图右侧就是对颜色做独热向量编码。转换器OneHotEncoder可以接受两种类型的输入：

* ① 用LabelEncoder编码好的一维数组
* ② DataFrame

**一、用LabelEncoder编码好的一维数组(元素为整数)，重塑(用reshape(-1,1))成二维数组作为OneHotEncoder输入**。

|  |  |
| --- | --- |
|  | from sklearn.preprocessing import OneHotEncoder |
|  | OHE = OneHotEncoder() |
|  | num = LE.fit\_transform( enc ) |
|  | print( num ) |
|  | OHE\_y = OHE.fit\_transform( num.reshape(-1,1) ) |
|  | OHE\_y |
|  | [2 0 1 2] |

输出为：

|  |  |
| --- | --- |
|  | <4x3 sparse matrix of type '<class 'numpy.float64'>' |
|  | with 4 stored elements in Compressed Sparse Row format> |

上面结果解释如下：

* 第3行打印出编码结果[2 0 1 2]。
* 第5行将其转成独热形式，输出是一个「稀疏矩阵」形式，因为实操中通常类别很多，因此就一步到位用稀疏矩阵来节省内存。

想看该矩阵里具体内容，用toarray()函数。

|  |  |
| --- | --- |
|  | OHE\_y.toarray() |

输出为：

|  |  |
| --- | --- |
|  | array([[0., 0., 1.], |
|  | [1., 0., 0.], |
|  | [0., 1., 0.], |
|  | [0., 0., 1.]]) |

**二、用DataFrame作为OneHotEncoder输入**。

|  |  |
| --- | --- |
|  | OHE = OneHotEncoder() |
|  | OHE.fit\_transform( enc\_DF ).toarray() |

输出为：

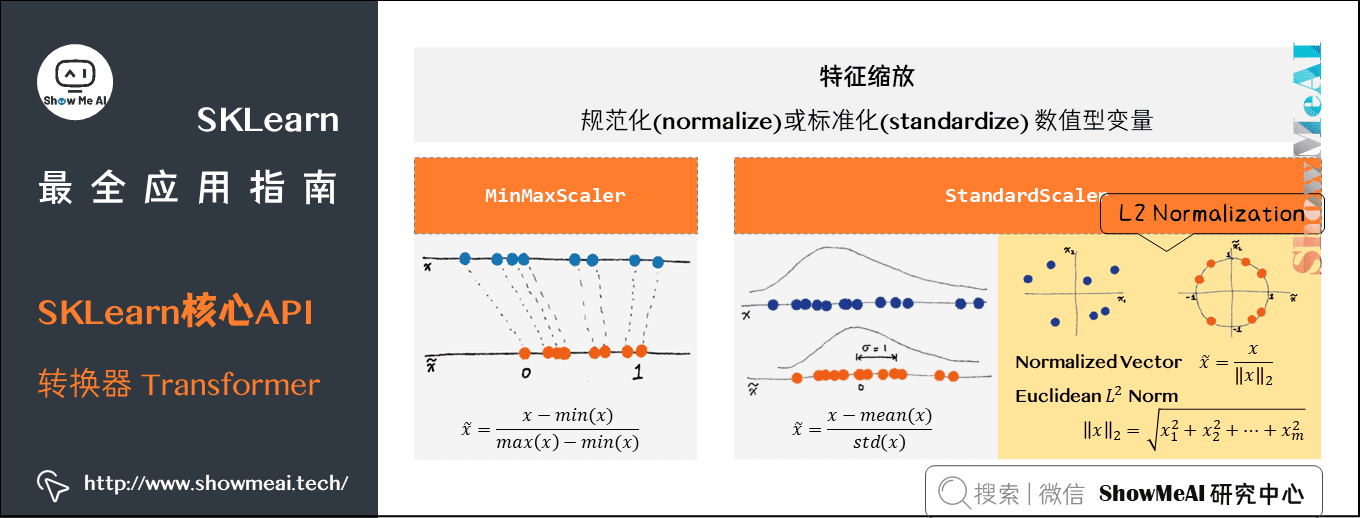
|  |  |
| --- | --- |
|  | array([[0., 0., 1.], |
|  | [1., 0., 0.], |
|  | [0., 1., 0.], |
|  | [0., 0., 1.]]) |

**(2) 特征缩放**

数据要做的最重要的转换之一是特征缩放(feature scaling)。类似逻辑回归，神经网络这种计算型模型，对于不同特征的幅度大小差异是敏感的。

具体来说，对于某个特征，我们有两种变换方法：

* **标准化**(**standardizati**on)：每个维度的特征减去该特征均值，除以该维度的标准差。
* **规范化**(**normalization**)：每个维度的特征减去该特征最小值，除以该特征的最大值与最小值之差。



**① MinMaxScaler**

如上图所示，MinMaxScaler会根据特征的最大最小取值，对数据进行幅度缩放。

|  |  |
| --- | --- |
|  | from sklearn.preprocessing import MinMaxScaler |
|  | X = np.array( [0, 0.5, 1, 1.5, 2, 100] ) |
|  | X\_scale = MinMaxScaler().fit\_transform( X.reshape(-1,1) ) |
|  | X\_scale |

输出为：

|  |  |
| --- | --- |
|  | array([[0. ], |
|  | [0.005], |
|  | [0.01 ], |
|  | [0.015], |
|  | [0.02 ], |
|  | [1. ]]) |

**② StandardScaler**

StandardScaler做的事情是调整数据分布，尽量接近正态分布。

|  |  |
| --- | --- |
|  | from sklearn.preprocessing import StandardScaler |
|  | X\_scale = StandardScaler().fit\_transform( X.reshape(-1,1) ) |
|  | X\_scale |

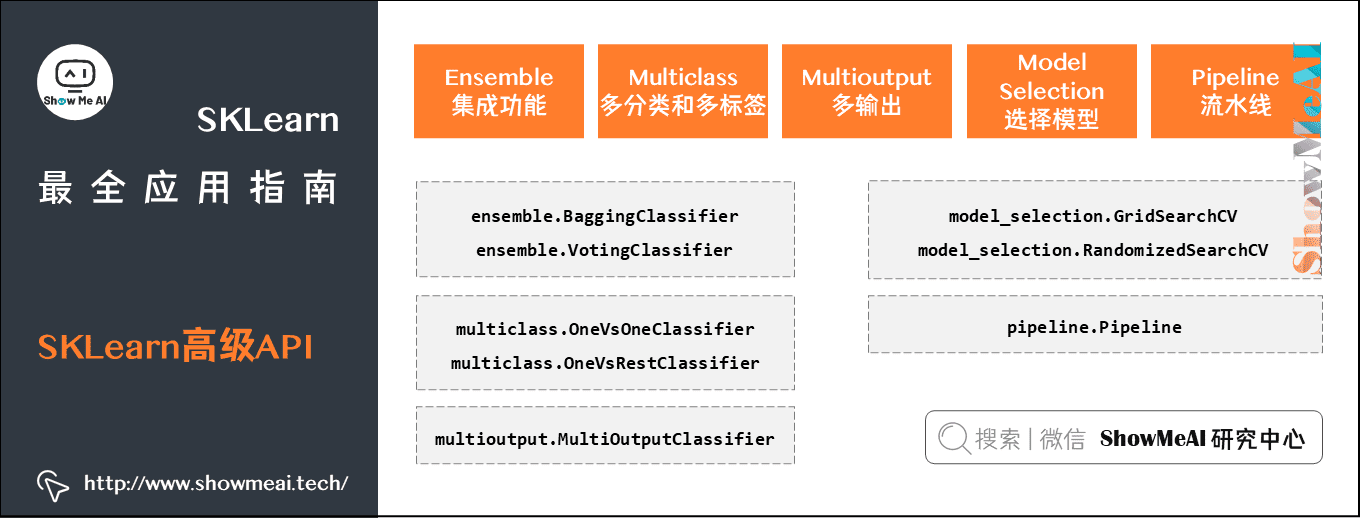
输出为：

|  |  |
| --- | --- |
|  | array([[-0.47424487], |
|  | [-0.46069502], |
|  | [-0.44714517], |
|  | [-0.43359531], |
|  | [-0.42004546], |
|  | [ 2.23572584]]) |

注意：fit()函数只能作用在训练集上，如果希望对测试集变换，只要用训练集上fit好的转换器去transform即可。不能在测试集上fit再transform，否则训练集和测试集的变换规则不一致，模型学习到的信息就无效了。

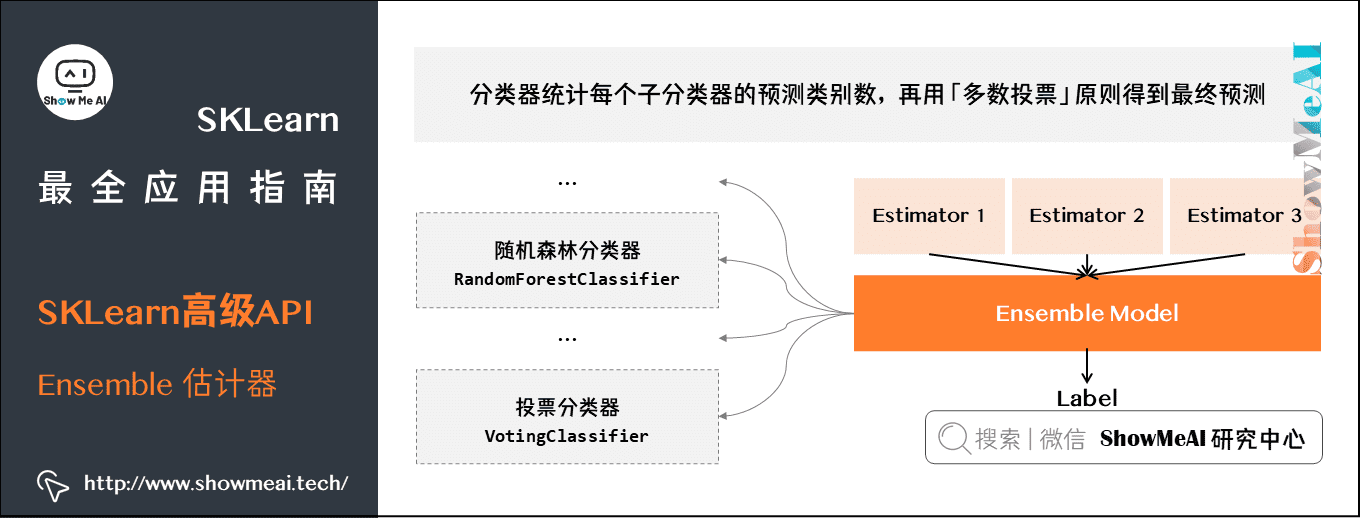
**4.高级API**

我们在这节中给大家介绍SKLearn的「**高级API**」，即五大元估计器（集成功能的Ensemble，多分类和多标签的Multiclass，多输出的Multioutput，选择模型的Model Selection，流水线的Pipeline）。



* ensemble.BaggingClassifier
* ensemble.VotingClassifier
* multiclass.OneVsOneClassifier
* multiclass.OneVsRestClassifier
* multioutput.MultiOutputClassifier
* model\_selection.GridSearchCV
* model\_selection.RandomizedSearchCV
* pipeline.Pipeline

**4.1 Ensemble 估计器**



如上图：分类器统计每个子分类器的预测类别数，再用「多数投票」原则得到最终预测。回归器计算每个子回归器的预测平均值。

最常用的Ensemble估计器排列如下：

* AdaBoostClassifier：逐步提升分类器
* AdaBoostRegressor：逐步提升回归器
* BaggingClassifier：Bagging分类器
* BaggingRegressor：Bagging回归器
* GradientBoostingClassifier：梯度提升分类器
* GradientBoostingRegressor：梯度提升回归器
* RandomForestClassifier：随机森林分类器
* RandomForestRegressor：随机森林回归器
* VotingClassifier：投票分类器
* VotingRegressor：投票回归器

我们用鸢尾花数据iris，拿以下estimator来举例：

* 含同质估计器RandomForestClassifier
* 含异质估计器VotingClassifier

首先将数据分成80:20的训练集和测试集，并引入metrics来计算各种性能指标。

|  |  |
| --- | --- |
|  | from sklearn.datasets import load\_iris |
|  | iris = load\_iris() |
|  | from sklearn.model\_selection import train\_test\_split |
|  | from sklearn import metrics |
|  | X\_train, X\_test, y\_train, y\_test = train\_test\_split(iris['data'], iris['target'], test\_size=0.2) |

**(1) RandomForestClassifier**



[**随机森林**](https://www.showmeai.tech/article-detail/191)RandomForestClassifier通过控制n\_estimators超参数来决定基估计器的个数，在这里是4棵决策树(森林由树组成)；此外每棵树的最大树深为5(max\_depth=5)。

|  |  |
| --- | --- |
|  | from sklearn.ensemble import RandomForestClassifier |
|  | RF = RandomForestClassifier( n\_estimators=4, max\_depth=5 ) |
|  | RF.fit( X\_train, y\_train ) |

输出为：

|  |  |
| --- | --- |
|  | RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='gini', |
|  | max\_depth=5, max\_features='auto', max\_leaf\_nodes=None, |
|  | min\_impurity\_decrease=0.0, min\_impurity\_split=None, |
|  | min\_samples\_leaf=1, min\_samples\_split=2, |
|  | min\_weight\_fraction\_leaf=0.0, n\_estimators=4, |
|  | n\_jobs=None, oob\_score=False, random\_state=None, |
|  | verbose=0, warm\_start=False) |

元估计器和预估器一样也有fit()。下面看看随机森林里包含的估计器个数和其本身。

|  |  |
| --- | --- |
|  | print( RF.n\_estimators ) |
|  | RF.estimators\_ |

输出为：

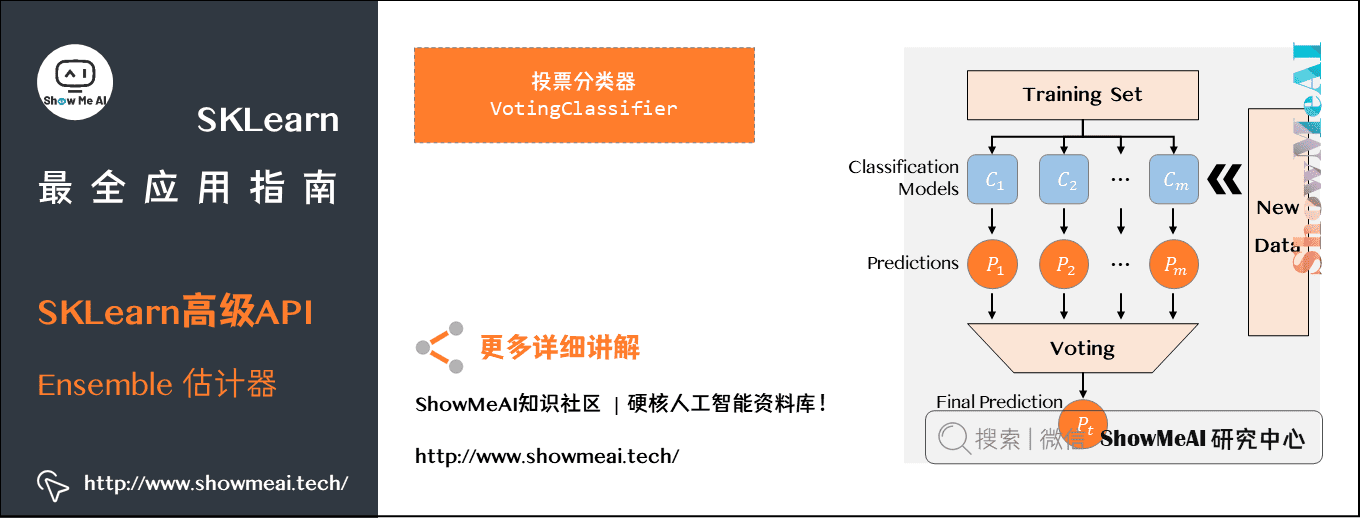
|  |  |
| --- | --- |
|  | 4 |
|  |  |
|  | [DecisionTreeClassifier(class\_weight=None, criterion='gini', max\_depth=5, |
|  | max\_features='auto', max\_leaf\_nodes=None, |
|  | min\_impurity\_decrease=0.0, min\_impurity\_split=None, |
|  | min\_samples\_leaf=1, min\_samples\_split=2, |
|  | min\_weight\_fraction\_leaf=0.0, presort=False, |
|  | random\_state=705712365, splitter='best'), |
|  | DecisionTreeClassifier(class\_weight=None, criterion='gini', max\_depth=5, |
|  | max\_features='auto', max\_leaf\_nodes=None, |
|  | min\_impurity\_decrease=0.0, min\_impurity\_split=None, |
|  | min\_samples\_leaf=1, min\_samples\_split=2, |
|  | min\_weight\_fraction\_leaf=0.0, presort=False, |
|  | random\_state=1026568399, splitter='best'), |
|  | DecisionTreeClassifier(class\_weight=None, criterion='gini', max\_depth=5, |
|  | max\_features='auto', max\_leaf\_nodes=None, |
|  | min\_impurity\_decrease=0.0, min\_impurity\_split=None, |
|  | min\_samples\_leaf=1, min\_samples\_split=2, |
|  | min\_weight\_fraction\_leaf=0.0, presort=False, |
|  | random\_state=1987322366, splitter='best'), |
|  | DecisionTreeClassifier(class\_weight=None, criterion='gini', max\_depth=5, |
|  | max\_features='auto', max\_leaf\_nodes=None, |
|  | min\_impurity\_decrease=0.0, min\_impurity\_split=None, |
|  | min\_samples\_leaf=1, min\_samples\_split=2, |
|  | min\_weight\_fraction\_leaf=0.0, presort=False, |
|  | random\_state=1210538094, splitter='best')] |

拟合RF完再做预测，用metrics里面的accuracy\_score来计算准确率。训练准确率98.33%，测试准确率100%。

|  |  |
| --- | --- |
|  | print ( "RF - Accuracy (Train): %.4g" % |
|  | metrics.accuracy\_score(y\_train, RF.predict(X\_train)) ) |
|  | print ( "RF - Accuracy (Test): %.4g" % |
|  | metrics.accuracy\_score(y\_test, RF.predict(X\_test)) ) |

|  |  |
| --- | --- |
|  | RF - Accuracy (Train): 1 |
|  | RF - Accuracy (Test): 0.9667 |

**(2) VotingClassifier**



和[**随机森林**](https://www.showmeai.tech/article-detail/191)由同质分类器「决策树」不同，投票分类器由若干个异质分类器组成。下面我们用VotingClassifier建立个含有逻辑回归(Logistic regression)、随机森林(RandomForest)和高斯朴素贝叶斯(GNB)三个分类器的集成模型。

RandomForestClassifier的基分类器只能是决策树，因此只用通过控制n\_estimators超参数来决定树的个数，而VotingClassifier的基分类器要输入每个异质分类器。

|  |  |
| --- | --- |
|  | from sklearn.linear\_model import LogisticRegression |
|  | from sklearn.naive\_bayes import GaussianNB |
|  | from sklearn.ensemble import RandomForestClassifier |
|  | from sklearn.ensemble import VotingClassifier |
|  |  |
|  | LR = LogisticRegression( solver='lbfgs', multi\_class='multinomial' ) |
|  | RF = RandomForestClassifier( n\_estimators=5 ) |
|  | GNB = GaussianNB() |
|  |  |
|  | Ensemble = VotingClassifier( estimators=[('lr', LR), (‘rf', RF), ('gnb', GNB)], voting='hard' ) |
|  |  |
|  | Ensemble. fit( X\_train, y\_train ) |
|  |  |

结果如下：

|  |  |
| --- | --- |
|  | VotingClassifier(estimators=[('lr', LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True,intercept\_scaling=1, max\_iter=100, multi\_class='multinomial',n\_jobs=None, penalty='12', random\_state=None, solver='lbfgs',tol=0.0001, verbose=6, warm\_start=False)), ('rf', ...e, verbose=0,warm\_start=False)), ('gnb', GaussianNB(priors=None, var\_smoothing=1e-09))],flatten\_transform=None, n\_jobs=None, voting='hard', weights=None) |

看看Ensemble集成模型里包含的估计器个数和其本身。

|  |  |
| --- | --- |
|  | print( len(Ensemble.estimators\_) ) |
|  | Ensemble.estimators\_ |

结果如下：

|  |  |
| --- | --- |
|  | 3 |
|  |  |
|  | [LogisticRegression(C=1.0, class\_weight-None, dual-False, fit\_intercept=True,intercept\_scaling=1, max\_iter=100, multi\_class='multinomial',n\_jobs-None, penalty="12", random\_state-None, solver='1bfgs',t01=0.0001, verbose=0, warm\_start=False), |
|  |  |
|  | RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='gini',max\_depth=None, max\_features='auto', max\_leaf\_nodes=None,min\_impurity\_decrease-0.0, min\_impurity\_splitmin\_samples\_leaf=1, min\_samples\_split=2,min\_weight\_fraction\_leaf=0.0, n\_estimator:oob\_score=False, random\_state-None, verbose= |
|  | warm\_start=False), |
|  |  |
|  | GaussianNB(priors-None, var\_smoothing=1e-9)] |

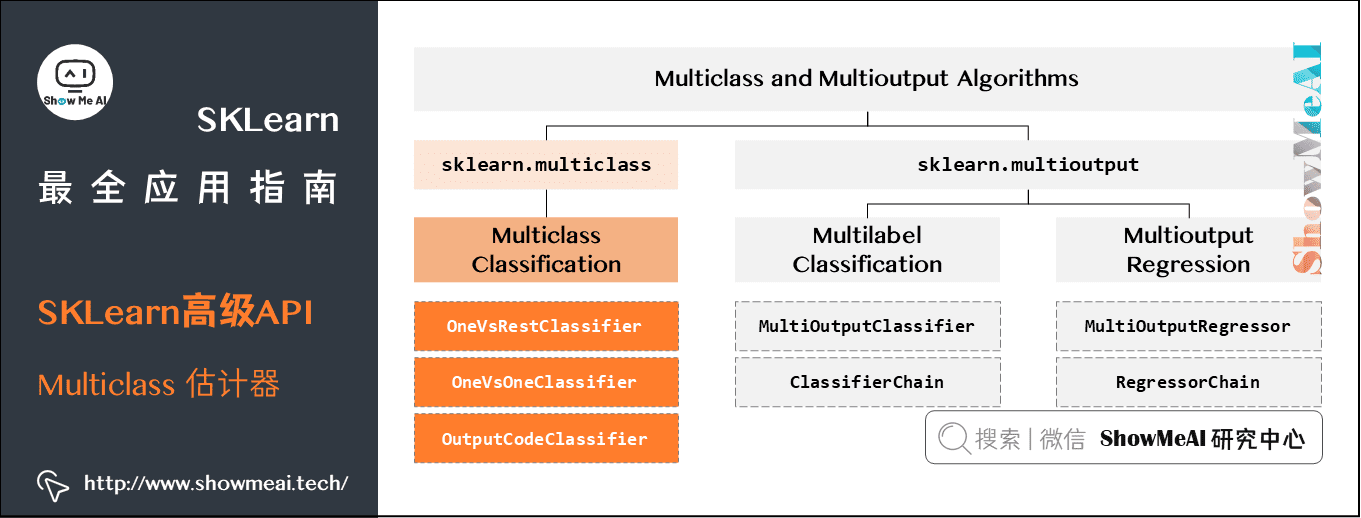
对比元估计器和它三个组成元素的表现，下过表现如下：

|  |  |
| --- | --- |
|  | # 拟合 |
|  | LR.fit( X\_train, y\_train ) |
|  | RF.fit( X\_train, y\_train ) |
|  | GNB.fit( X\_train, y\_train ) |

|  |  |
| --- | --- |
|  | # 评估效果 |
|  | print ( "LR - Accuracy (Train): %.4g" % metrics.accuracy\_score(y\_train, LR.predict(X\_train)) ) |
|  | print ( "RF - Accuracy (Train): %.4g" % metrics.accuracy\_score(y\_train, RF.predict(X\_train)) ) |
|  | print ( "GNB - Accuracy (Train): %.4g" % metrics.accuracy\_score(y\_train, GNB.predict(X\_train)) ) |
|  | print ( "Ensemble - Accuracy (Train): %.4g" % metrics.accuracy\_score(y\_train, Ensemble.predict(X\_train)) ) |
|  | print ( "LR - Accuracy (Test): %.4g" % metrics.accuracy\_score(y\_test, LR.predict(X\_test)) ) |
|  |  |
|  | print ( "RF - Accuracy (Test): %.4g" % metrics.accuracy\_score(y\_test, RF.predict(x\_test)) ) |
|  | print ( "GNB - Accuracy (Test): %.4g" % metrics.accuracy\_score(y\_test, RF.predict(X\_test)) ) |
|  | print ( "Ensemble - Accuracy (Test): %.4g" % metrics.accuracy\_score(y test, Ensemble.predict(X\_test)) ) |

|  |  |
| --- | --- |
|  | # 运行结果 |
|  | LR - Accuracy (Train): 0.975 |
|  | RF - Accuracy (Train): 0.9833 |
|  | GNB - Accuracy (Train): 0.95 |
|  | Ensemble - Accuracy (Train): 0.9833 |
|  | LR - Accuracy (Test): 1 |
|  | RF - Accuracy (Test): 1 |
|  | GNB - Accuracy (Test): 1 |
|  | Ensemble - Accuracy (Test): 1 |

**4.2 Multiclass 估计器**



sklearn.multiclass可以处理多类别(multi-class) 的多标签(multi-label) 的分类问题。下面我们会使用数字数据集digits作为示例数据来讲解。我们先将数据分成 80:20 的训练集和测试集。

|  |  |
| --- | --- |
|  | # 导入数据 |
|  | from sklearn.datasets import load\_digits |
|  | digits = load\_digits() |
|  | digits.keys() |

输出如下：

|  |  |
| --- | --- |
|  | # 输出结果 |
|  | dict\_keys(['data', 'target', 'target\_names','images', 'DESCR']) |

下面我们切分数据集：

|  |  |
| --- | --- |
|  | # 数据集切分 |
|  | X\_train, X\_test, y\_train, y\_test = train\_test\_split( digits['data'], digits['target'], test\_size=0.2 ) |
|  |  |
|  | print( 'The size of X\_train is ', X\_train.shape ) |
|  | print( 'The size of y\_train is ', y\_train.shape ) |
|  | print( 'The size of X\_test is ', X\_test.shape ) |
|  | print( 'The size of y\_test is ', y\_test.shape ) |

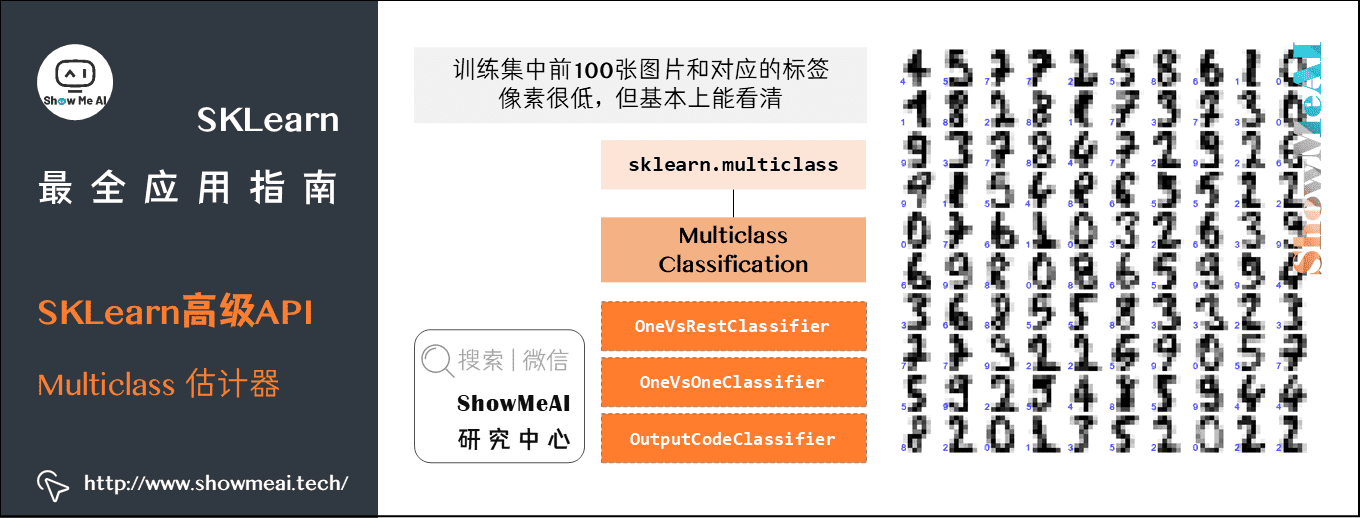
输出如下

|  |  |
| --- | --- |
|  | The size of X\_train is (1437, 64) |
|  | The size of y\_train is (1437,) |
|  | The size of X\_test is (360, 64) |
|  | The size of y\_test is (360,) |

训练集和测试集分别有1437和360张图像。每张照片是包含8×8的像素，我们用flatten操作把2维的8×8展平为1维的64。

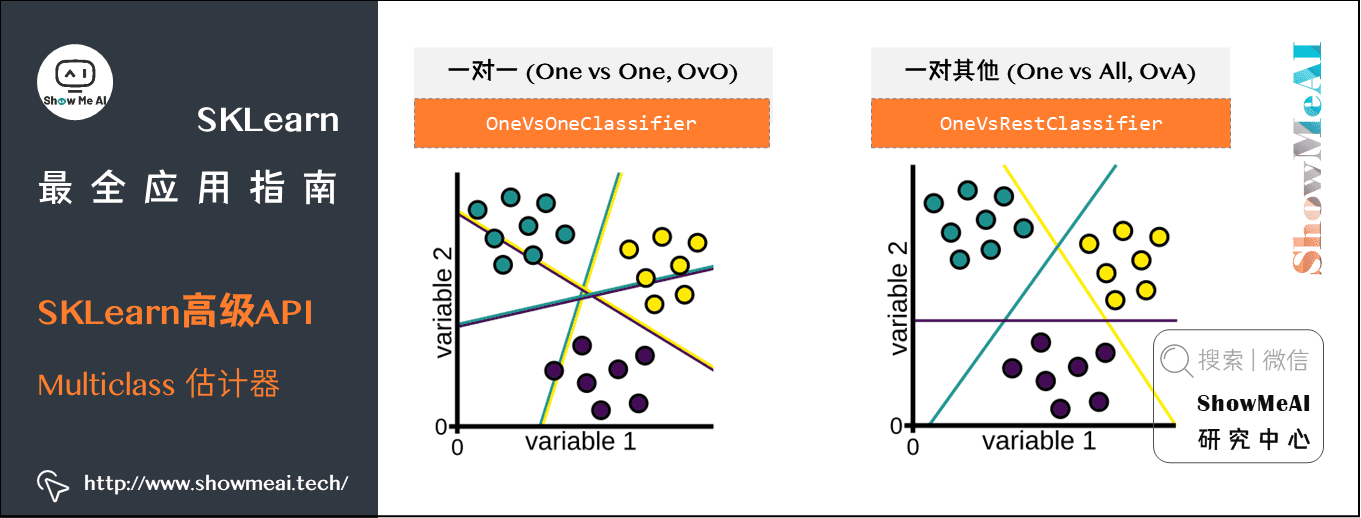
看看训练集中前100张图片和对应的标签（如下图）。像素很低，但基本上还是能看清。

|  |  |
| --- | --- |
|  | fig, axes = plt.subplots( 10, 16, figsize=(8, 8) ) |
|  | fig.subplots\_adjust( hspace=0.1, wspace=0.1 ) |
|  | for i, ax in enumerate( axes.flat ): |
|  | ax.imshow( X\_train[i,:].reshape(8,8), cmap='binary’, interpolation='nearest’) |
|  | ax.text( 0.05, 0.05, str(y\_train[i]), |
|  | transform=ax.transAxes, color='blue') |
|  | ax.set\_xticks([]) |
|  | ax.set\_yticks([]) |



**(1) 多类别分类**

手写数字有0-9十类，但手头上只有二分类估计器（比如像支撑向量机）怎么用呢？我们可以采取以下策略处理：



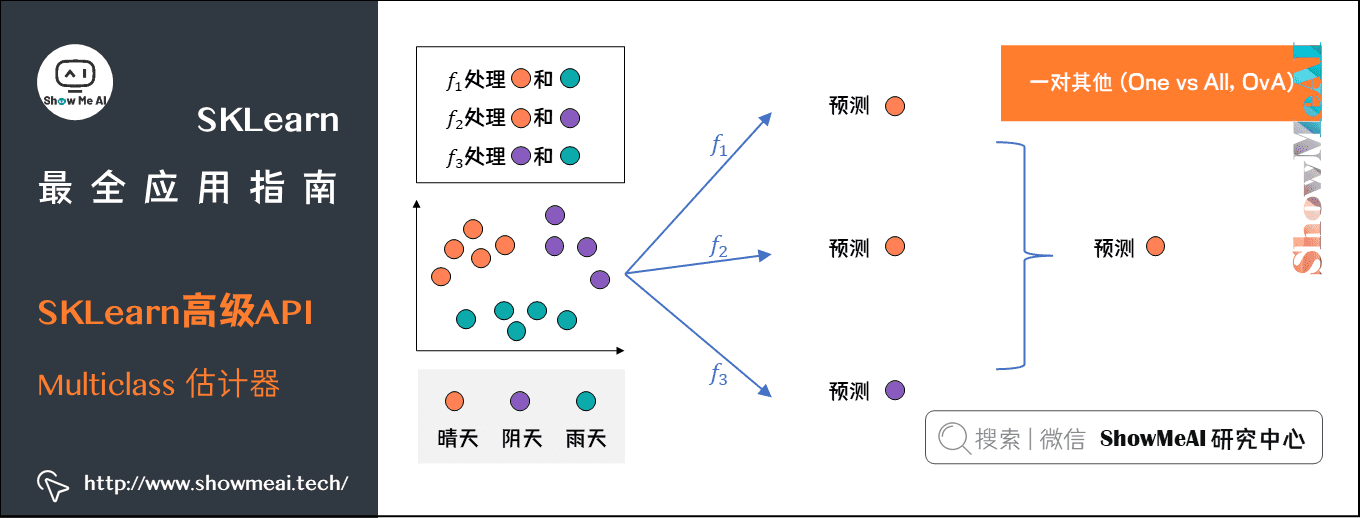
* **一对一**(One vs One，OvO)：一个分类器用来处理数字0和数字1，一个用来处理数字0和数字2，一个用来处理数字1和2，以此类推。N个类需要N(N-1)/2个分类器。
* **一对其他**(One vs All，OvA)：训练10个二分类器，每一个对应一个数字，第一个分类「1」和「非1」，第二个分类「2」和「非2」，以此类推。N个类需要N个分类器。

**① OneVsOneClassifier**

考虑一个具体天气多分类问题，天气可以是晴天、阴天和雨天，在OvO中，三个分类器为f1、f2和f3。

* f1负责区分橙色和绿色样本
* f2负责区分橙色和紫色样本
* f3负责区分绿色和紫色样本

在下图的例子中，f1和f2都预测为橙色，f3预测为紫色。根据多数原则得到的结合预测为橙色，如下图所示。



回到数字分类问题上，代码及结果如下：

|  |  |
| --- | --- |
|  | from sklearn.multiclass import OneVsOneClassifier |
|  | from sklearn.linear\_model import LogisticRegression |
|  | ovo\_lr = OneVsOneClassifier( LogisticRegression(solver='lbfgs', max\_iter=200) ) |
|  | ovo\_lr.fit( X\_train, y\_train ) |

|  |  |
| --- | --- |
|  | OnevsOneClassifier(estimator=LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True,intercept\_scaling=1, max\_iter=200, multi\_class=‘warn’,n\_jobs=None, penalty='12', random\_state=None, solver='lbfgs’,tol=0.0001, verbose=6, warm\_start=False),n\_jobs=None) |
|  |  |

10\*9/2=45，10类总共45个OvO分类器。

|  |  |
| --- | --- |
|  | print( len(ovo\_lr.estimators\_) ) |
|  | ovo\_lr.estimators\_ |

结果如下：

|  |  |
| --- | --- |
|  | 45 |
|  |  |
|  | (LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True,intercept\_scaling=1, max\_iter=200, multi\_class='warn',n\_jobs=None, penalty='12', random\_state=None, solver='lbfgs',tol=60.0001, verbose=0, warm\_start=False), |
|  |  |
|  | LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True, intercept\_scaling=1, max\_iter=200, multi\_class='warn', n\_jobs=None, penalty='l2', random\_state=None, solver='lbfgs',tol=0.0001, verbose=0, warm\_start=False), |
|  |  |
|  | LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True, intercept\_scaling=1, max\_iter=200, multi\_class='warn', n\_jobs=None, penalty='12', random\_state=None, solver='lbfgs', tol=60.0001, verbose=0, warm\_start=False), |
|  |  |
|  | LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True, intercept\_scaling=1, max\_iter=200, multi\_class='warn', n\_jobs=None, penalty="12", random\_state=None, solver='lbfgs', tol=0.0001, verbose=0, warm\_start=False), |
|  |  |
|  | LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True, |
|  | ... |

训练集分类全对，测试集准确率98%。

|  |  |
| --- | --- |
|  | print ( “OvO LR - Accuracy (Train): %.4g" % metrics.accuracy\_score(y\_train, ovo\_Ir.predict(X\_train)) ) |
|  | print ( "OvO LR - Accuracy (Test): %.4g" % metrics.accuracy\_score(y\_test, ovo\_lr.predict(X\_test}) ) |
|  |  |

|  |  |
| --- | --- |
|  | # 运行结果 |
|  | OvO LR - Accuracy (Train): 1 |
|  | OvO LR - Accuracy (Test): 0.9806 |

**② OneVsRestClassifier**

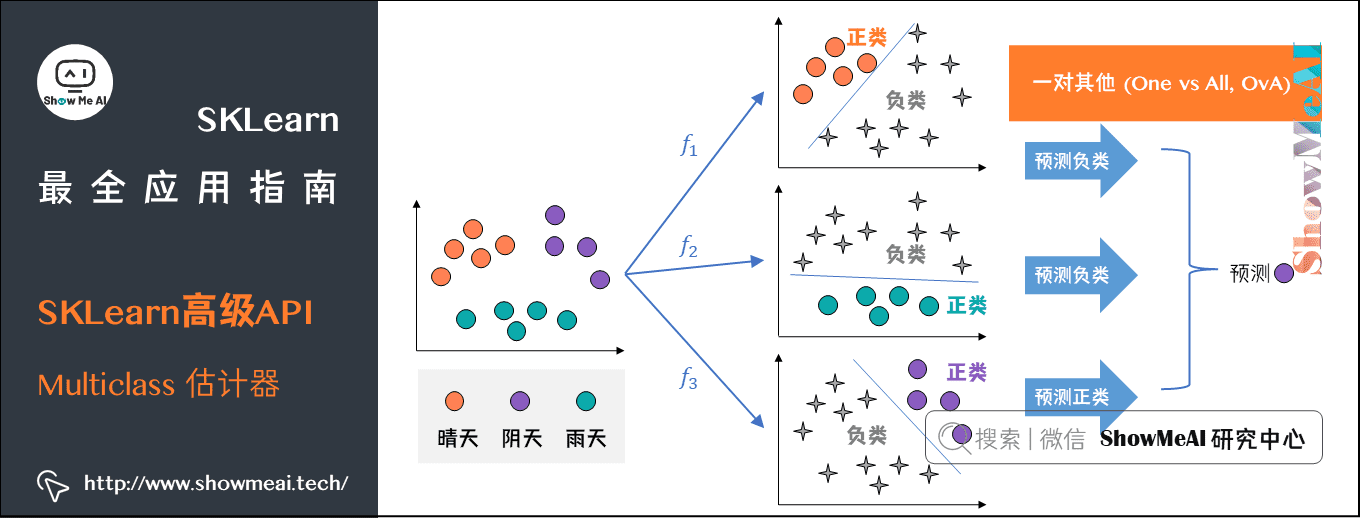
在OvA中，把数据分成“某个”和“其他”

* 图一，某个=橙色，其他=绿色和紫色
* 图二，某个=绿色，其他=橙色和紫色
* 图三，某个=紫色，其他=橙色和绿色

三分类分解成三个二分类，对应的分类器为f1、f2和f3。

* f1预测负类，即预测绿色和紫色
* f2预测负类，即预测橙色和紫色
* f3预测正类，即预测紫色

三个分类器都预测了紫色，根据多数原则得到的预测是紫色，即阴天。



回到数字分类问题上，代码和结果如下：

|  |  |
| --- | --- |
|  | from sklearn.multiclass import OneVsRestClassifier |
|  | ova\_lr = OneVsRestClassifier( LogisticRegression(solver='lbfgs', max\_iter=800) ) |
|  | ova\_lr.fit( X\_train, y\_train ) |

|  |  |
| --- | --- |
|  | OnevsRestClassifier(estimator=LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True, intercept\_scaling=1, max\_iter=800, multi\_class=‘warn’, n\_jobs=None, penalty='12', random\_state=None, solver='lbfgs’, tol=0.0001, verbose=6, warm\_start=False), n\_jobs=None) |
|  |  |

SQL 复制 全屏

10类总共10个OvA分类器。

|  |  |
| --- | --- |
|  | print( len(ova\_lr.estimators\_) ) |
|  | ova\_lr.estimators\_ |

结果如下：

|  |  |
| --- | --- |
|  | 10 |
|  |  |
|  | [LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True, intercept\_scaling=1, max\_iter=800, multi\_class='warn', n\_jobs=None, penalty='12', random\_state=None, solver='lbfgs',tol=0.0001, verbose=0, warm\_start=False), |
|  |  |
|  | LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True, intercept\_scaling=1, max\_iter=800, multi\_class='warn', n\_jobs=None, penalty='12', random\_state=None, solver='lbfgs', tol=0.0001, verbose=0, warm\_start=False), |
|  |  |
|  | LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True, |
|  | intercept\_scaling=1, max\_iter=800, multi\_class=‘warn', |
|  | n\_jobs=None, penalty='12', random\_state=None, solver="lbfgs', |
|  | tol=0.0001, verbose=0, warm\_start=False), |
|  |  |
|  | LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True, |
|  | intercept\_scaling=1, max\_iter=800, multi\_class='warn', n\_jobs=None, penalty='12', random\_state=None, solver='lbfgs', tol=0.0001, verbose=0, warm\_start=False), |
|  |  |
|  | LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True, |
|  | ... |

训练集准确率几乎100%，测试集准确率96%。代码与结果如下：

|  |  |
| --- | --- |
|  | print ( “OvA LR - Accuracy (Train): %.4g" % metrics.accuracy\_score(y\_train, ova\_Ir.predict(X\_train)) ) |
|  | print ( "OvA LR - Accuracy (Test): %.4g" % metrics.accuracy\_score(y\_test, ova\_lr.predict(X\_test}) ) |
|  |  |

|  |  |
| --- | --- |
|  | OvA LR - Accuracy (Train): 6.9993 |
|  | OvA LR - Accuracy (Test}: 6.9639 |

**(2) 多标签分类**

到目前为止，所有的样例都总是被分配到仅一个类。有些情况下，你也许想让分类器给一个样例输出多个类别。在无人驾驶的应用中，在下图识别出有车和指示牌，没有交通灯和人。



物体识别是一个复杂的深度学习问题，我们在这里暂且不深入探讨。我们先看一个简单点的例子，在手写数字的例子上，我们特意为每个数字设计了两个标签：

* 标签1：奇数、偶数
* 标签2：小于等于4，大于4

我们构建多标签y\_train\_multilabel，代码如下（OneVsRestClassifier也可以用来做多标签分类）：

|  |  |
| --- | --- |
|  | from sklearn.multiclass import OneVsRestClassifier |
|  | y\_train\_multilabel = np.c\_[y\_train%2==0, y\_train<=4 ] |
|  | print(y\_train\_multilabel) |

|  |  |
| --- | --- |
|  | [[ True True] [False False] [False False] |
|  | ... |
|  | [False False] [False False] [False False]] |

看下图训练集第1和2个图片是数字4和5，对应上面两种标签结果为：

* [True True]：4是偶数，小于等于4
* [False False]：5不是偶数，大于4



我们这次用y\_train\_multilabel来训练模型。代码如下

|  |  |
| --- | --- |
|  | ova\_ml = OneVsRestClassifier( LogisticRegression(solver='lbfgs', max\_iter=800) ) |
|  | ova\_ml.fit( X\_train, y\_train\_multilabel ) |

|  |  |
| --- | --- |
|  | # 运行结果 |
|  | OnevsRestClassifier(estimator=LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True, intercept\_scaling=1, max\_iter=800, multi\_class=‘warn’, n\_jobs=None, penalty='12', random\_state=None, solver='lbfgs', tol=0.0001, verbose=6, warm\_start=False), n\_jobs=None) |

有两个估计器，每个对应一个标签。

|  |  |
| --- | --- |
|  | print( len(ova\_ml.estimators\_) ) |
|  | ova\_ml.estimators\_ |

运行结果如下：

|  |  |
| --- | --- |
|  | 2 |
|  |  |
|  | [LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True, intercept\_scaling=1, max\_iter=800, multi\_class=‘warn', n\_jobs=None, penalty='12°, random\_state=None, solver='lbfgs', tol=0.0001, verbose=0, warm\_start=False), |
|  |  |
|  | LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True, intercept\_scaling=1, max\_iter=800, multi\_class='warn', n\_jobs=None, penalty='l2', random\_state=None, solver='lbfgs', tol=0.0001, verbose=0, warm\_start=False) ] |

展示一下测试集上100张图片。

|  |  |
| --- | --- |
|  | fig, axes = plt.subplots( 10, 10, figsize=(8, 8) ) |
|  | fig.subplots\_adjust( hspace=0.1, wspace=0.1 ) |
|  |  |
|  | for i, ax in enumerate( axes.flat ): |
|  | ax.imshow( X\_test[i,:].reshape(8,8), cmap='binary', interpolation='nearest') |
|  | ax.text( 6.05, 0.05, str(y\_test[i]), transform=ax.transAxes, color='blue') |
|  | ax.set\_xticks([]) |
|  | ax.set\_yticks([]) |



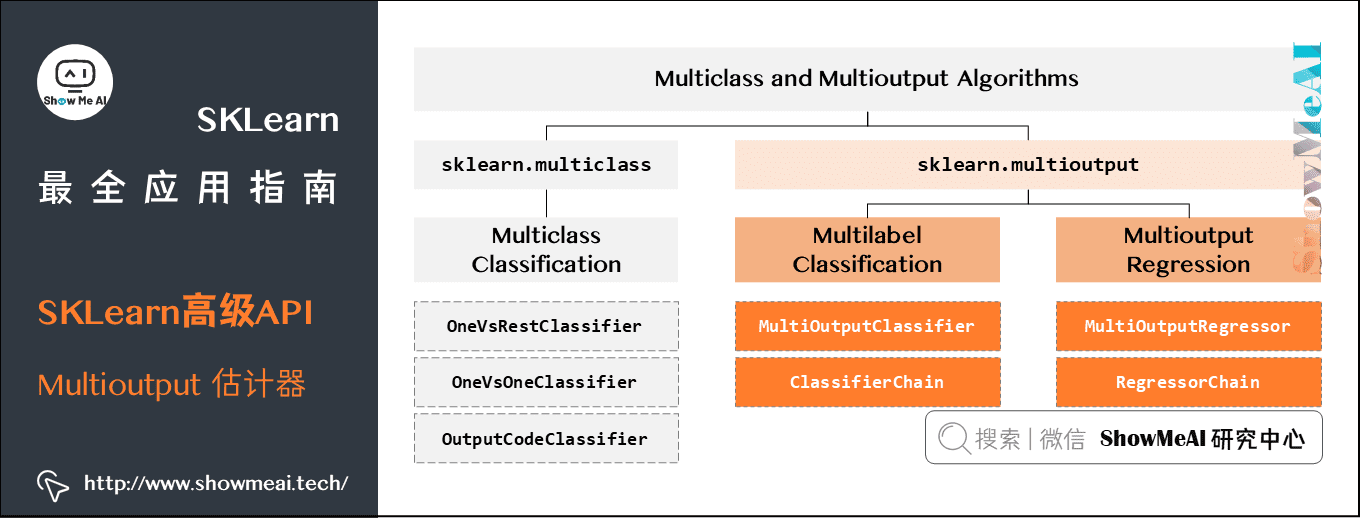
第一张图片是数字2，它是偶数(标签1为true)，小于等于4(标签2为true)。

|  |  |
| --- | --- |
|  | print( y\_test[:1] ) |
|  | print( ova\_ml.predict(X\_test[:1,:]) ) |

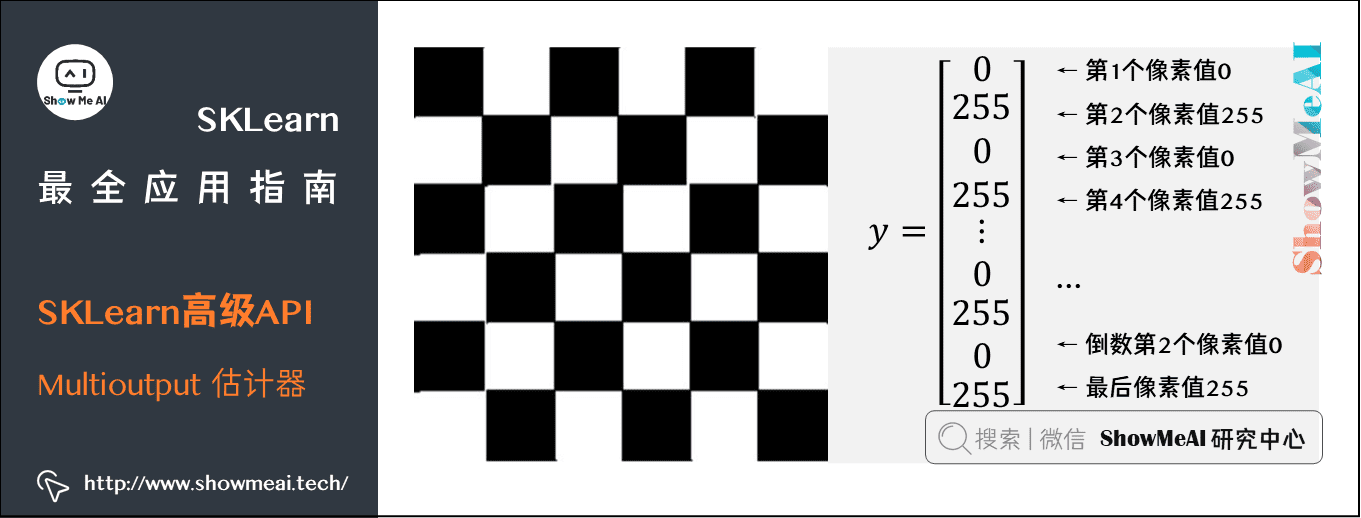
|  |  |
| --- | --- |
|  | [2] |
|  | [[1 1]] |

**4.3 Multioutput 估计器**

sklearn.multioutput可以处理多输出(multi-output)的分类问题。



多输出分类是多标签分类的泛化，在这里每一个标签可以是多类别(大于两个类别)的。一个例子就是预测图片每一个像素(标签)的像素值是多少(从0到255的256个类别)。



Multioutput估计器有两个：

* MultiOutputRegressor：多输出回归
* MultiOutputClassifier：多输出分类

这里我们只关注多输出分类。

**(1) MultiOutputClassifier**

首先引入MultiOutputClassifier和RandomForestClassifier。

|  |  |
| --- | --- |
|  | from sklearn.multioutput import MultiOutputClassifier |
|  | from sklearn.ensemble import RandomForestClassifier |

在手写数字的例子上，我们也为特意每个数字设计了多标签而且每个标签的类别都大于二。

* 标签1：小于等于4，4和7之间，大于等于7(三类)
* 标签2：数字本身(十类)

代码如下：

|  |  |
| --- | --- |
|  | y\_train\_1st = y\_train.copy() |
|  | y\_train\_1st[ y\_train<=4 ] = 0 |
|  | y\_train\_1st[ np.logical\_and{y\_train>4, y\_train<7) ] = 1 |
|  | y\_train\_ist[ y\_train>=7 ] = 2 |
|  |  |
|  | y\_train\_multioutput = np.c\_[y\_train\_1st, y\_train] |
|  | y\_train\_multioutput |

|  |  |
| --- | --- |
|  | # 运行结果 |
|  | array( [[0, 4], |
|  | [1, 5], |
|  | [2, 7], |
|  | [1, 5], |
|  | [2, 9], |
|  | [2, 9]]) |

用含有100棵决策树的随机森林来解决这个多输入分类问题。

|  |  |
| --- | --- |
|  | MO = MultiOutputClassifier( RandomForestClassifier(n\_estimators=100) ) |
|  | MO.fit( X\_train, y\_train\_multioutput ) |

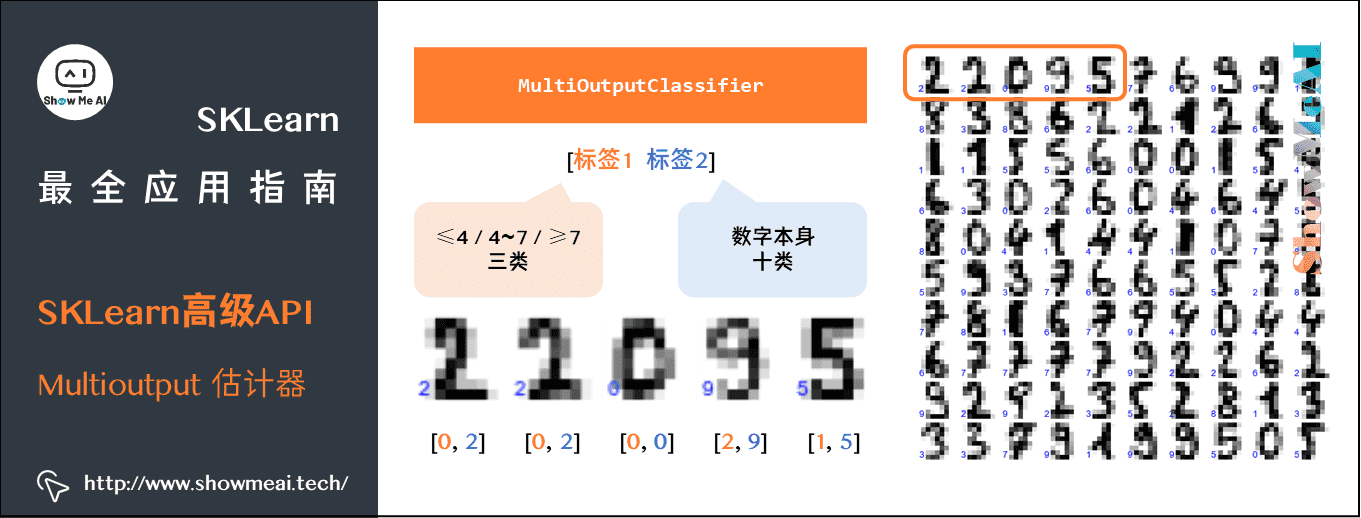
|  |  |
| --- | --- |
|  | # 结果 |
|  | MultiOutputClassifier(estimator=RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='gini', max\_depth=None, max\_features='auto', max\_leaf\_nodes=None, min\_impurity\_decrease=0.0, min\_impurity\_split=None, min\_samples\_leaf=1, min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0, n\_estimators=100, n\_jobs=None, oob\_score=False, random\_state=None, verbose=0, warm\_start=False), n\_jobs=None) |

看看这个模型在测试集前五张照片上的预测。

|  |  |
| --- | --- |
|  | MO.predict( X\_test[:5,:] ) |

|  |  |
| --- | --- |
|  | array([[0, 2],[0, 2],[0, 0],[2, 9],[1, 5]]) |

这个ndarray第一列是标签1的类别，第二列是标签2的类别。预测结果是这五张照片分别显示数字2、2、0、9、5(标签2)，它们前三个数2、2、0都小于等于4(标签1第一类)，第四个数9大于等于7(标签1第二类)，而第五个数5在4和7之间(标签1第三类)。



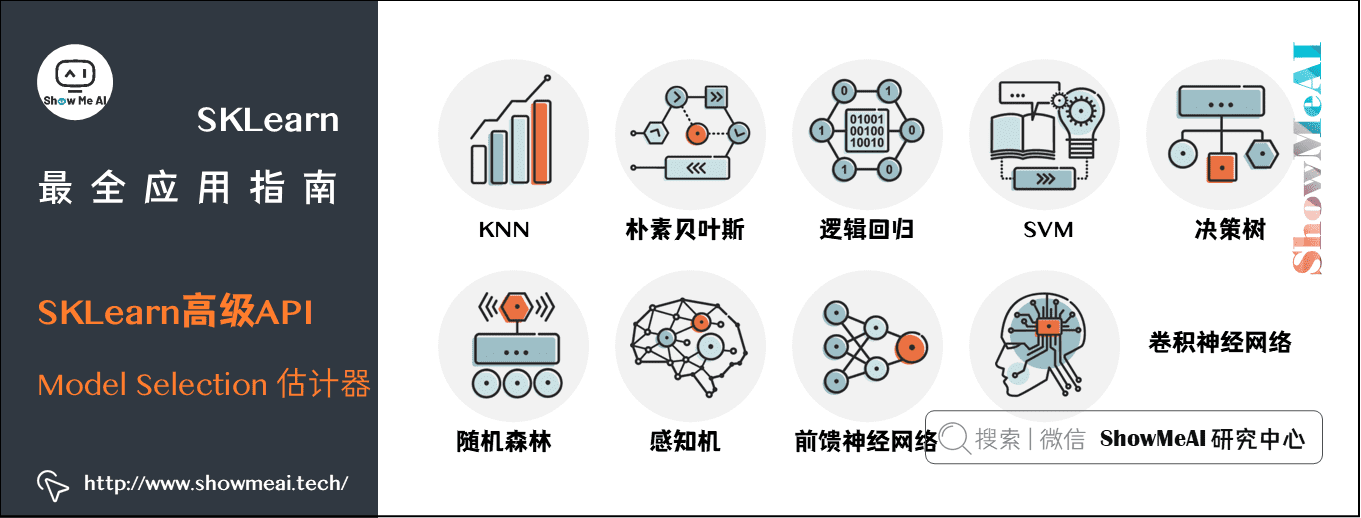
再看看真实标签。

|  |  |
| --- | --- |
|  | y\_test\_1st = y\_test.copy() |
|  | y\_test\_1st[ y\_test<=4 ] = 0 |
|  | y\_test\_1st[ np.logical\_and(y\_test>4, y\_test<7) ] = 1 |
|  | y\_test\_1st[ y\_test>=7 ] = 2 |
|  | y\_test\_multioutput = np.c\_[ y\_test\_1st, y\_test ] |
|  | y\_test\_multioutput[:5] |

|  |  |
| --- | --- |
|  | array([[0, 2],[0, 2],[0, 0],[2, 9],[1, 5]]) |

对比参考结果标签，模型预测的结果还是很不错的。

**4.4 Model Selection 估计器**



模型选择(Model Selction)在机器学习非常重要，它主要用于评估模型表现，常见的Model Selection估计器有以下几个：

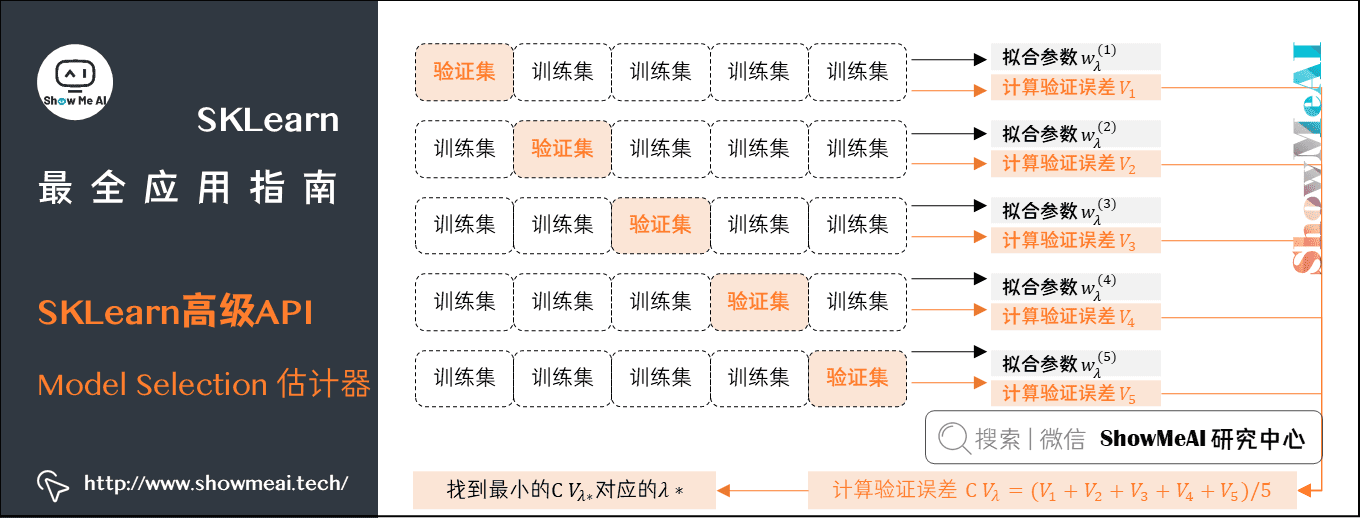
* cross\_validate：评估交叉验证的结果。
* learning\_curve：构建与绘制学习曲线。
* GridSearchCV：用交叉验证从超参数候选网格中搜索出最佳超参数。
* RandomizedSearchCV：用交叉验证从一组随机超参数搜索出最佳超参数。

这里我们只关注调节超参数的两个估计器，即GridSearchCV和RandomizedSearchCV。我们先回顾一下交叉验证（更详细的讲解请查看[ShowMeAI](https://www.showmeai.tech/)文章 [**图解机器学习 | 模型评估方法与准则**](https://www.showmeai.tech/article-detail/186)）。

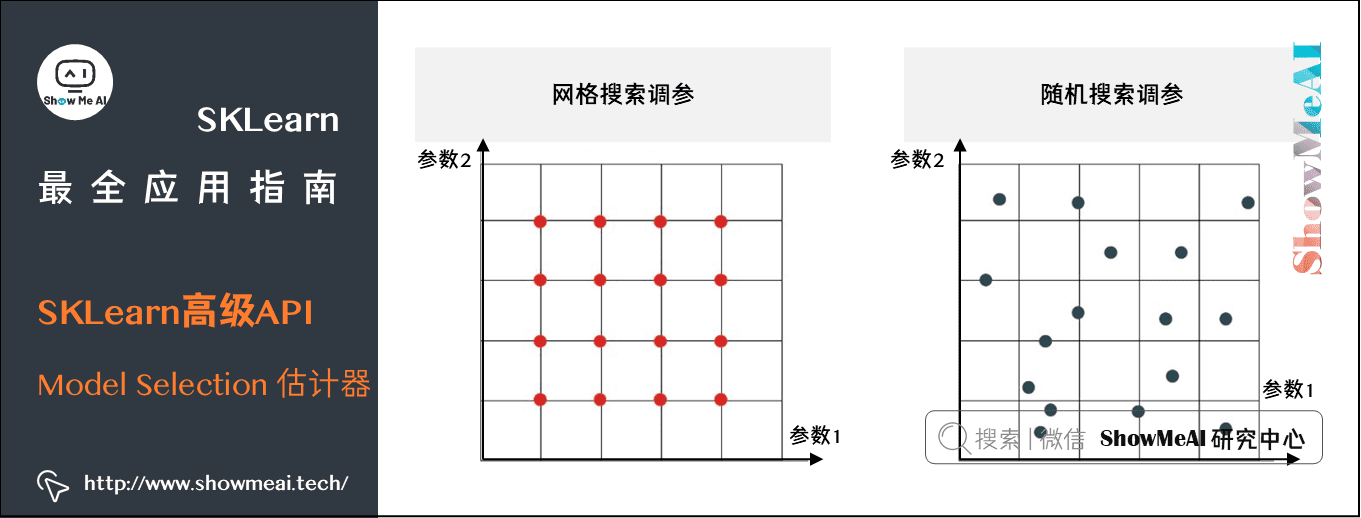
**(1) 交叉验证**

K-折交叉验证(K-fold cross validation set)，指的是把整个数据集平均但随机分成K份，每份大概包含m/K个数据(m 是总数据数)。

在这K份，每次选K-1份作为训练集拟合参数，在剩下1份验证集上进行评估计算。由于遍历了这K份数据，因此该操作称为交叉验证。操作如下图所示



下图展示了两个调参的估计器：「**网格搜索**」和「**随机搜索**」。



**网格搜索调参**：参数1在[1,10,100,1000]中取值，参数2在[0.01, 0.1, 1 10] 中取值，注意并不是等间距取值。模型在所有16组超参数上实验，选取交叉验证误差最小的参数。

**随机搜索调参**：根据指定分布随机搜索，可以选择独立于参数个数，比如log(参数1)服从0到3的均匀分布，log(参数2)服从-2到1的均匀分布。

应用方式与参考代码如下：

|  |  |
| --- | --- |
|  | from time import time |
|  | from scipy.stats import randint |
|  | from sklearn.model\_selection import GridSearchCv |
|  | from sklearn.model\_selection import RandomizedSearchcCv |
|  | from sklearn.ensemble import RandomForestClassifier |
|  |  |
|  | X, y = digits.data, digits.target |
|  | RFC = RandomForestClassifier(n\_estimators=20) |
|  |  |
|  | # 随机搜索/Randomized Search |
|  | param\_dist = { "max\_depth": [3, 5], |
|  | "max\_features": randint(1, 11), |
|  | "min\_samples\_split": randint(2, 11), |
|  | "criterion": ["gini", "entropy"]} |
|  | n\_iter\_search = 20 |
|  | random\_search = RandomizedSearchCv( RFC, param\_distributions=param\_dist, n\_iter=n\_iter\_search, cv=5 )} |
|  | start = time() |
|  | random\_search.fit(X, y) |
|  | print("RandomizedSearchCv took %.2f seconds for %d candidates，parameter settings." % ((time() - start), n\_iter\_search)) |
|  | print( random\_search.best\_params\_ ) |
|  | print( random\_search.best\_score\_ ) |
|  |  |
|  | # 网格搜索/Grid Search |
|  | param\_grid = { "max\_depth": [3, 5], |
|  | "max\_features": [1, 3, 10], |
|  | "min\_samples\_ split": [2, 3, 10], |
|  | "criterion": ["gini", "entropy"]} |
|  | grid\_search = GridSearchCV( RF, param\_grid=param\_grid, cv=5 ) |
|  | start = time() |
|  | grid\_search.fit(X, y) |
|  |  |
|  | print("\nGridSearchcv took %.2f seconds for %d candidate parameter settings." % (time() - start, len(grid\_search.cv\_results\_['params']))) |
|  | print( grid\_search.best\_params\_ ) |
|  | print( grid\_search.best\_score\_ ) |

输出结果如下：

|  |  |
| --- | --- |
|  | RandomizedSearchCv took 3.73 seconds for 20 candidates parameter settings. |
|  | {'criterion': 'entropy', '\*max\_depth': 5, 'max\_features': 6, 'min\_samples\_split': 4} |
|  | 0.8898163606010017 |
|  |  |
|  | GridSearchCV took 2.30 seconds for 36 candidate parameter settings. |
|  | {'criterion': 'entropy', 'max\_depth': 5, 'max\_features': 10, 'min\_samples\_ split': 10} |
|  | 0.841402337228714S5 |

这里我们对代码做一个解释：

* 前5行引入相应工具库。
* 第7-8行准备好数据X和y，创建一个含20个决策树的随机森林模型。
* 第10-14和23-27行为对随机森林的超参数「最大树深、最多特征数、最小可分裂样本数、分裂标准」构建候选参数分布与参数网格。
* 第15-18行是运行**随机搜索**。
* 第18-30行是运行**网格搜索**。

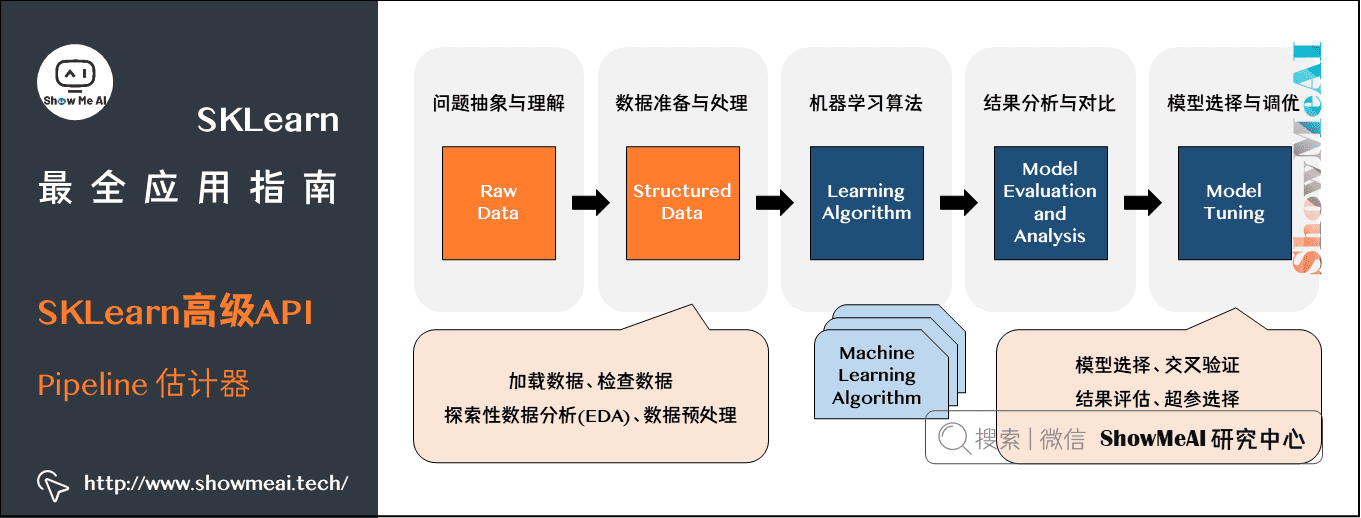
运行结果里：

* 第一行输出每种追踪法运行的多少次和花的时间。
* 第二行输出最佳超参数的组合。
* 第三行输出最高得分。

在本例中，**随机搜索**比**网格搜索**用更短时间内找到一组超参数，获得了更高的得分。

**4.5 Pipeline 估计器**

Pipeline估计器又叫流水线，把各种估计器串联(Pipeline)或并联(FeatureUnion)的方式组成一条龙服务。用好了它真的能大大提高效率。



**(1) Pipeline**

Pipeline将若干个估计器按顺序连在一起，比如：特征提取 → 降维 → 拟合 → 预测

Pipeline的属性永远和最后一个估计器属性一样：

* 如果最后一个估计器是预测器，那么Pipeline是预测器。
* 如果最后一个估计器是转换器，那么Pipeline是转换器。

下面是一个简单示例，使用Pipeline来完成「填补缺失值-标准化」这两步的。我们先构建含缺失值NaN的数据X。

|  |  |
| --- | --- |
|  | X = np.array([[56,40,30,5,7,10,9,np.NaN,12], |
|  | [1.68,1.83,1.77,np.NaN,1.9,1.65,1.88,np.NaN,1.75]]) |
|  | X = np.transpose(X) |

我们用以下流程组件构建Pipeline：

* 处理缺失值的转换器SimpleImputer。
* 做规划化的转换器MinMaxScaler。

|  |  |
| --- | --- |
|  | from sklearn.pipeline import Pipeline |
|  | from sklearn.impute import SimpleImputer |
|  | from sklearn.preprocessing import MinMaxScaler |
|  |  |
|  | pipe = Pipeline([ |
|  | ('impute', SimpleImputer(missing\_values=np.nan, strategy='mean')), |
|  | ('normalize', MinMaxScaler())]) |

第5-7行创建了流水线，使用方式非常简单，在Pipeline()里输入(名称,估计器)这个元组构建的流水线列表。在本例中SimpleImputer起名叫impute，MinMaxScaler起名叫normalize。

因为最后一个估计器是转换器，因此pipeline也是个转换器。下面我们来运行一下，我们发现值都被填满了，而且两列也被标准化了。

|  |  |
| --- | --- |
|  | X\_proc = pipe.fit\_transform( X ) |

来验证上面流水线的参数，我们可以按顺序来运行这两个转换器，结果是一样的。

|  |  |
| --- | --- |
|  | X\_impute = SimpleImputer(missing values=np.nan, strategy='mean').fit\_transform( X ) |
|  | X\_impute |

|  |  |
| --- | --- |
|  | # 运行结果 |
|  | array( [[50, 1.68], |
|  | [40, 1.83], |
|  | [30, 1.77], |
|  | [5, 1.78], |
|  | [7, 1.9 ], |
|  | [10, 1.65], |
|  | [9, 1.88], |
|  | [20.375, 1.78], |
|  | [12, 1.75 ]]) |

|  |  |
| --- | --- |
|  | X\_normalize = MinMaxScaler().fit\_transform( X\_impute ) |
|  | X\_normalize |

运行结果

|  |  |
| --- | --- |
|  | array( [[1., 0.12 ], |
|  | [0.77777778, 0.72], |
|  | [0.55555556, 6.48], |
|  | [0.52, 1], |
|  | [0.04444444, 1.], |
|  | [0.11111111, 9.], |
|  | [0.08888889, 6.92], |
|  | [0.34166667, 6.52], |
|  | [0.15555556, 0.4 ]]) |

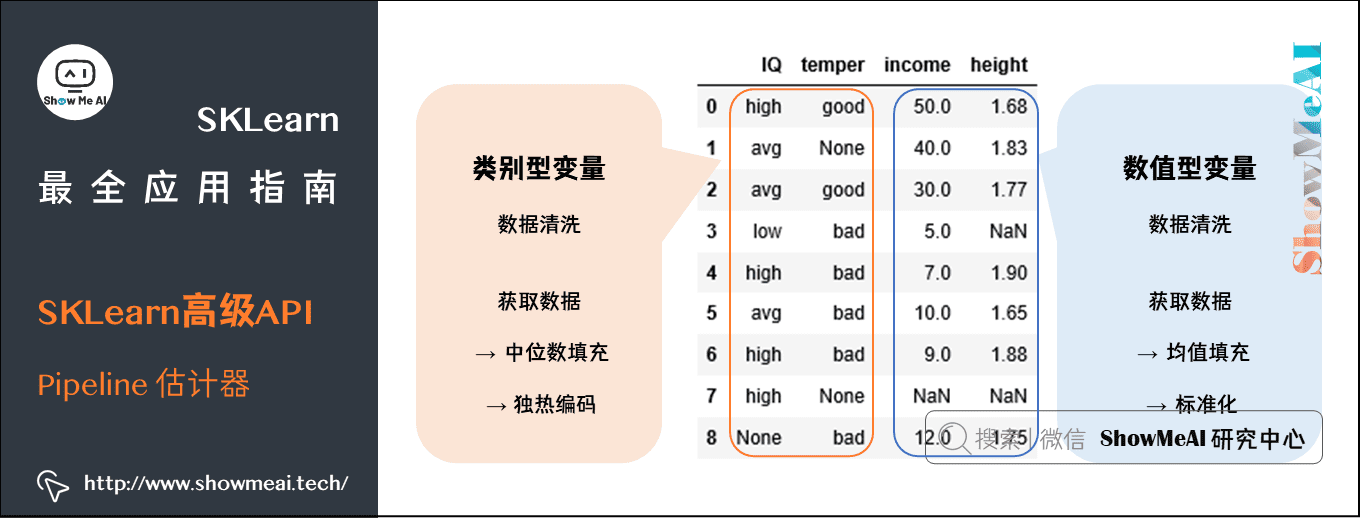
**(2) FeatureUnion**

如果我们想在一个节点同时运行几个估计器，我们可用FeatureUnion。在下面的例子中，我们首先建立一个DataFrame数据，它有如下特点：

* 前两列字段「智力IQ」和「脾气temper」都是类别型变量。
* 后两列字段「收入income」和「身高height」都是数值型变量。
* 每列中都有缺失值。

|  |  |
| --- | --- |
|  | d= { 'IQ' : ['high','avg','avg','low', high', avg', 'high', 'high',None], |
|  | 'temper' : ['good', None,'good', 'bad', 'bad','bad', 'bad', None, 'bad'], |
|  | 'income' : [50,40,30,5,7,10,9,np.NaN,12], |
|  | 'height' : [1.68,1.83,1.77,np.NaN,1.9,1.65,1.88,np.NaN,1.75]} |
|  |  |
|  | X = pd.DataFrame( d ) |
|  | X |

结果如下：



我们现在按下列步骤来清洗数据。

* **对类别型变量**：获取数据 → 中位数填充 → 独热编码
* **对数值型变量**：获取数据 → 均值填充 → 标准化

上面两步是并行进行的。

首先我们自己定义一个从DataFrame里面获取数据列的类，起名叫DataFrameSelector。

|  |  |
| --- | --- |
|  | from sklearn.base import BaseEstimator, TransformerMixin |
|  |  |
|  | class DataFrameSelector( BaseEstimator, TransformerMixin ): |
|  | def \_init\_( self, attribute\_names ): |
|  | self.attribute\_names = attribute\_names |
|  | def fit( self, X, y=None ): |
|  | return self |
|  | def transform( self, X ): |
|  | return X[self.attribute\_names].values |

上述代码在transform函数中，我们将输入的DataFrame X根据属性名称来获取其值。

接下来建立流水线full\_pipe，它并联着两个流水线

* **categorical\_pipe处理分类型变量**
  + DataFrameSelector用来获取
  + SimpleImputer用出现最多的值来填充None
  + OneHotEncoder来编码返回非稀疏矩阵
* **numeric\_pipe处理数值型变量**
  + DataFrameSelector用来获取
  + SimpleImputer用均值来填充NaN
  + normalize来规范化数值

代码如下：

|  |  |
| --- | --- |
|  | from sklearn.pipeline import Pipeline |
|  | from sklearn.pipeline import FeatureUnion |
|  | from sklearn.impute import SimpleImputer |
|  | from sklearn.preprocessing import MinMaxScaler |
|  | from sklearn.preprocessing import OneHotEncoder |
|  |  |
|  | categorical features = ['IQ', 'temper'] |
|  | numeric\_features = ['income', 'height'] |
|  |  |
|  | categorical pipe = Pipeline([ |
|  | ('select', DataFrameSelector(categorical\_features)), |
|  | ('impute', SimpleImputer(missing values=None, strategy='most\_frequent')), |
|  | ('one\_hot\_encode', OneHotEncoder(sparse=False))]) |
|  |  |
|  | numeric\_pipe = Pipeline([ |
|  | ('select', DataFrameSelector(numeric\_features)), |
|  | ('impute', SimpleImputer(missing values=np.nan, strategy='mean')), |
|  | ('normalize', MinMaxScaler())]) |
|  |  |
|  | full\_pipe = FeatureUnion( transformer\_list=[ |
|  | ('numeric\_pipe', numeric\_pipe), |
|  | ('categorical\_pipe', categorical\_pipe)]) |

我们打印结果如下：

|  |  |
| --- | --- |
|  | X\_proc = full\_pipe.fit\_transform( X ) |
|  | print( X\_proc ) |

|  |  |
| --- | --- |
|  | [[1. 0.12 0. 1. 0. 0. 1. ] |
|  | [0.77777778 0.72 1. 0. 0. 1. 0. ] |
|  | [0.55555556 0.48 1. 0. 0. 0. 1. ] |
|  | [0. 0.52 0. 0. 1. 1. 0. ] |
|  | [0.04444444 1. 0. 1. 0. 1. 0. ] |
|  | [0.11111111 0. 1. 0. 0. 1. 0. ] |
|  | [0.08888889 0.92 0. 1. 0. 1. 0. ] |
|  | [0.34166667 0.52 0. 1. 0. 1. 0. ] |
|  | [0.15555556 0.4 0. 1. 0. 1. 0. ]] |

**5.总结**

下面我们对上面讲解到的sklearn工具库应用知识做一个总结。

**5.1 SKLearn五大原则**

SKLearn的设计下，它的主要API遵循五大原则

**(1) 一致性**

所有对象的接口一致且简单，在「估计器」中

* 创建：model = Constructor(hyperparam)
* 拟参：
  + 有监督学习：model.fit(X\_train, y\_train)
  + 无监督学习：model.fit(X\_train)

在「预测器」中

* 有监督学习里预测标签：y\_pred = model.predict(X\_test)
* 无监督学习里识别模式：idx\_pred = model.predict( Xtest)

在「转换器」中

* 创建：trm = Constructor(hyperparam)
* 获参：trm.fit(X\_train)
* 转换：X\_trm = trm.transform(X\_train)

**(2) 可检验**

所有估计器里设置的超参数和学到的参数都可以通过实例的变量直接访问来检验其值，区别是超参数的名称最后没有下划线\_，而参数的名称最后有下划线\_。举例如下：

* 通例：model.hyperparameter
* 特例：SVC.kernel
* 通例：model.parameter\_
* 特例：SVC.support\_vectors\_

**(3) 标准类**

SKLearn模型接受的数据集的格式只能是「Numpy数组」和「Scipy稀疏矩阵」。超参数的格式只能是「字符」和「数值」。  
不接受其他的类！

**(4) 可组成**

模块都能重复「连在一起」或「并在一起」使用，比如两种形式流水线(pipeline)

* 任意转换器序列
* 任意转换器序列+估计器

**(5) 有默认**

SKLearn给大多超参数提供了合理的默认值，大大降低了建模的难度。

**5.2 SKLearn框架流程**

sklearn的建模应用流程框架大概如下：

**(1) 确定任务**

是「有监督」的分类或回归？还是「无监督」的聚类或降维？确定好后基本就能知道用Sklearn里哪些模型了。

**(2) 数据预处理**

这步最繁琐，要处理缺失值、异常值；要编码类别型变量；要正规化或标准化数值型变量，等等。但是有了Pipeline神器一切变得简单高效。

**(3) 训练和评估**

这步最简单，训练用估计器fit()先拟合，评估用预测器predict()来评估。

**(4) 选择模型**

启动ModelSelection估计器里的GridSearchCV和RandomizedSearchCV，选择得分最高的那组超参数(即模型)。

**参考资料**

* [图解机器学习算法 | 从入门到精通系列](https://www.showmeai.tech/tutorials/34)
* SKLearn官网：<https://scikit-learn.org/stable/>
* [AI建模工具速查|Scikit-learn使用指南](https://www.showmeai.tech/article-detail/108)

**机器学习【算法】系列教程**

* [图解机器学习 | 机器学习基础知识](https://www.showmeai.tech/article-detail/185)
* [图解机器学习 | 模型评估方法与准则](https://www.showmeai.tech/article-detail/186)
* [图解机器学习 | KNN算法及其应用](https://www.showmeai.tech/article-detail/187)
* [图解机器学习 | 逻辑回归算法详解](https://www.showmeai.tech/article-detail/188)
* [图解机器学习 | 朴素贝叶斯算法详解](https://www.showmeai.tech/article-detail/189)
* [图解机器学习 | 决策树模型详解](https://www.showmeai.tech/article-detail/190)
* [图解机器学习 | 随机森林分类模型详解](https://www.showmeai.tech/article-detail/191)
* [图解机器学习 | 回归树模型详解](https://www.showmeai.tech/article-detail/192)
* [图解机器学习 | GBDT模型详解](https://www.showmeai.tech/article-detail/193)
* [图解机器学习 | XGBoost模型最全解析](https://www.showmeai.tech/article-detail/194)
* [图解机器学习 | LightGBM模型详解](https://www.showmeai.tech/article-detail/195)
* [图解机器学习 | 支持向量机模型详解](https://www.showmeai.tech/article-detail/196)
* [图解机器学习 | 聚类算法详解](https://www.showmeai.tech/article-detail/197)
* [图解机器学习 | PCA降维算法详解](https://www.showmeai.tech/article-detail/198)

**机器学习【实战】系列教程**

* [机器学习实战 | Python机器学习算法应用实践](https://www.showmeai.tech/article-detail/201)
* [机器学习实战 | SKLearn入门与简单应用案例](https://www.showmeai.tech/article-detail/202)
* [机器学习实战 | SKLearn最全应用指南](https://www.showmeai.tech/article-detail/203)
* [机器学习实战 | XGBoost建模应用详解](https://www.showmeai.tech/article-detail/204)
* [机器学习实战 | LightGBM建模应用详解](https://www.showmeai.tech/article-detail/205)
* [机器学习实战 | Python机器学习综合项目-电商销量预估](https://www.showmeai.tech/article-detail/206)
* [机器学习实战 | Python机器学习综合项目-电商销量预估<进阶方案>](https://www.showmeai.tech/article-detail/207)
* [机器学习实战 | 机器学习特征工程最全解读](https://www.showmeai.tech/article-detail/208)
* [机器学习实战 | 自动化特征工程工具Featuretools应用](https://www.showmeai.tech/article-detail/209)
* [机器学习实战 | AutoML自动化机器学习建模](https://www.showmeai.tech/article-detail/210)

**ShowMeAI系列教程推荐**

* [大厂技术实现方案系列](https://www.showmeai.tech/tutorials/50)
* [图解Python编程：从入门到精通系列教程](https://www.showmeai.tech/tutorials/56)
* [图解数据分析：从入门到精通系列教程](https://www.showmeai.tech/tutorials/33)
* [图解AI数学基础：从入门到精通系列教程](https://www.showmeai.tech/tutorials/83)
* [图解大数据技术：从入门到精通系列教程](https://www.showmeai.tech/tutorials/84)
* [图解机器学习算法：从入门到精通系列教程](https://www.showmeai.tech/tutorials/34)
* [机器学习实战：手把手教你玩转机器学习系列](https://www.showmeai.tech/tutorials/41)
* [深度学习教程：吴恩达专项课程 · 全套笔记解读](https://www.showmeai.tech/tutorials/35)
* [自然语言处理教程：斯坦福CS224n课程 · 课程带学与全套笔记解读](https://www.showmeai.tech/tutorials/36)
* [深度学习与计算机视觉教程：斯坦福CS231n · 全套笔记解读](https://www.showmeai.tech/tutorials/37)

分类: [机器学习实战通关指南 ⛵ 全场景覆盖AI解决方案](https://www.cnblogs.com/showmeai/category/2129373.html)

标签: [sklearn](https://www.cnblogs.com/showmeai/tag/sklearn/), [机器学习](https://www.cnblogs.com/showmeai/tag/%E6%9C%BA%E5%99%A8%E5%AD%A6%E4%B9%A0/), [人工智能](https://www.cnblogs.com/showmeai/tag/%E4%BA%BA%E5%B7%A5%E6%99%BA%E8%83%BD/), [算法](https://www.cnblogs.com/showmeai/tag/%E7%AE%97%E6%B3%95/)