第8章 sklearn机器学习实战

董付国

微信公众号：Python小屋

本章学习目标

- 了解机器学习常用的基本概念
- 了解如何根据实际问题类型选择合适的机学习算法
- 了解扩展库sklearn常用模块
- 理解并熟练运用线性回归算法
- 理解并熟练运用逻辑回归算法
- 理解并熟练运用KNN算法
- 理解并熟练运用KMeans算法
- 理解分层聚类算法

- 理解并熟练运用朴素贝叶斯算法
- 理解并熟练运用决策树与随机森林算法
- 理解并熟练运用DBSCAN算法
- 理解并熟练运用协同过滤算法
- 理解并熟练运用关联规则分析算法
- 理解并熟练运用支持向量机算法
- 理解数据降维的作用和主成分分析的应用
- 理解并熟练运用交叉验证评估模型泛化能力
- 理解并熟练运用网格搜索确定最佳参数

8.1 机器学习基本概念

- 机器学习（Machine Learning）根据已知数据来不断学习和积累经验，然后总结出规律并尝试预测未知数据的属性，是一门综合性非常强的多领域交叉学科，涉及线性代数、概率论、逼近论、凸分析、算法复杂度理论等多门学科。
- 目前机器学习已经有了十分广泛的应用，例如数据挖掘、计算机视觉、自然语言处理、生物特征识别、搜索引擎、医学诊断、检测信用卡欺诈、证券市场分析、DNA序列测序、语音和手写识别、推荐系统、战略游戏和机器人运用等。

8.1 机器学习基本概念

- 在有监督学习中，所有数据带有额外的属性（例如每个样本所属的类别），必须同时包含输入和预期输出（也就是特征和目标），通过大量已知的数据不断训练和减少错误来提高认知能力，最后根据积累的经验去预测未知数据的属性。分类和回归属于经典的有监督学习算法。
- 在分类算法中，样本属于两个或多个离散的类别之一，我们根据已贴标签的样本来学习如何预测未贴标签样本所属的类别。如果预期的输出是一个或多个连续变量，则分类问题变为回归问题。

8.1 机器学习基本概念

- 在无监督学习算法中，训练数据包含一组输入向量而没有相应的目标值。这类算法的目标可能是发现原始数据中相似样本的组合（称作聚类），或者确定数据的分布（称作密度估计），或者把数据从高维空间投影到低维空间（称作降维）以便进行分析或可视化。
- 一般地，不会把给定的整个数据集都用来训练模型，而是将其分成训练集和测试集两部分，模型使用训练集进行训练或学习，然后把测试集输入训练好的模型并评估其表现。
- 大多数模型都有若干参数可以设置，例如支持向量机模型的gamma参数，这些参数可以手动设置，也可以使用网格搜索（grid search）和交叉验证（cross validation）寻找最合适的值。

8.1 机器学习基本概念

（1）一维数组

一般用来表示向量，其shape属性的长度为1。例如

```
>>> import numpy as np
>>> data = np.array([1, 2, 3, 4, 5])
>>> data
array([1, 2, 3, 4, 5])
>>> data.shape
(5,)
>>> len(data.shape)          # shape属性长度为1
1
```

8.1 机器学习基本概念

（2）二维数组

一般用来表示矩阵或样本数据，每行表示一个样本，每列表示样本的一个特征。二维数组shape属性的长度为2。例如

```
>>> import numpy as np
>>> data = np.array([[1,2,3], [4,5,6]])
>>> data
array([[1, 2, 3],
       [4, 5, 6]])
>>> data.shape                # 2行3列
(2, 3)
>>> len(data.shape)           # shape属性长度为2
2
```

8.1 机器学习基本概念

（3）稀疏矩阵（sparse matrix）

其中大多数元素为0的二维数值型矩阵，扩展库scipy.sparse中实现了稀疏矩阵的高效表示、存储和相关运算。

（4）可调用对象（callable）

在Python中，可调用对象主要包括函数（function）、lambda表达式、类（class）、类的方法（method）、实现了特殊方法__call__()的类的对象，这些对象作为内置函数callable()的参数会使得该函数返回True。

8.1 机器学习基本概念

（5）样本（sample）

通常用来表示单个特征向量，其中每个分量表示样本的一个特征，这些特征组成的特征向量应该能够准确地描述一个样本并能够很好地区别于其他样本。

（6）特征（feature）、特征向量（feature vector）

抽象地讲，特征是用来把一个样本对象映射到一个数字或类别的函数，也常用来表示这些数字或类别（即一个样本若干特征组成的特征向量中的每个分量）。

在数据矩阵中，特征表示为列，每列包含把一个特征函数应用到一组样本上的结果，每行表示一个样本若干特征组成的特征向量。

8.1 机器学习基本概念

（7）特征提取器（feature extractor）

把样本映射到固定长度数组形式数据（如numpy数组、Python列表、元组以及只包含数值的pandas.DataFrame和pandas.Series对象）的转换器，至少应提供fit()、transform()和get_feature_names()方法。

（8）目标（target）

有监督学习或半监督学习中的因变量，一般作为参数y传递给评估器的拟合方法fit()，也称作结果变量、理想值或标签。

8.1 机器学习基本概念

偏差（也叫作离差）描述算法的期望预测结果与真实结果的偏离程度，反应了模型的拟合能力。在使用中往往使用偏差的平方，计算公式为：

$$bias^2(x) = \left(\bar{f}(x) - y\right)^2$$

其中，$\bar{f}(x)$ 表示模型的期望预测结果，y 表示真实结果。

方差用来描述数据的离散程度或者波动程度，比较分散的数据集的方差大，而相对集中的数据集的方差小。方差计算公式为：

$$\mathrm{var}(x) = E_D[(f(x;D) - \bar{f}(x))^2]$$

其中，$f(x;D)$ 表示在训练集 D 上得到的模型 f 在 x 上的预测输出，$\bar{f}(x) = E_D[f(x;D)]$ 表示在训练集 D 上得到的模型 f 在 x 上预测输出的期望值。

在机器学习中，方差可以用来描述模型的稳定性，过于依赖数据集的模型的方差大，对数据集依赖性不强的模型方差小。

8.1 机器学习基本概念

（10）维度（dimension）

一般指特征的数量，或者二维特征矩阵中列的数量，也是特定问题中每个样本特征向量的长度。

（11）早停法（early stopping）

把数据集分成训练集和测试集，使用训练集对模型进行训练，并周期性地使用测试集对模型进行测试和验证，如果模型在测试集上的表现开始变差就停止训练，避免过拟合问题。

（12）评估度量（evaluation metrics）

评估度量用来测量模型的表现有多好，也常指metrics模块中的函数。

8.1 机器学习基本概念

（13）拟合（fit）

拟合泛指一类数据处理的方式，包括回归、插值、逼近。简单地说，对于平面上若干已知点，拟合是构造一条光滑曲线，使得该曲线与这些点的分布最接近，曲线在整体上靠近这些点，使得某种意义下的误差最小。

（14）过拟合（overfit）

当模型设计过于复杂时，在拟合过程中过度考虑数据中的细节，甚至使用了过多的噪声，使得模型过分依赖训练数据（具有较高的方差和较低的偏差），导致新数据集上的表现很差，这种情况叫做过拟合。可以通过增加样本数量、简化模型、对数据进行降维减少使用的特征、早停、正则化或其他方法避免过拟合问题。

（15）欠拟合（underfit）

过于关注数据会导致过拟合，而忽略数据时容易导致欠拟合。模型不够复杂，没有充分考虑数据集中的特征，导致拟合能力不强，模型在训练数据和测试数据上的表现都很差，这种情况叫做欠拟合。

8.1 机器学习基本概念

（16）填充算法（imputation algorithms）

大多数机器学习算法要求输入没有缺失值，否则无法正常工作。试图填充缺失值的算法称作填充算法或插补算法。

（17）数据泄露（Data Leakage）

预测器在训练时使用了在实际预测时不可用的数据特征，或者误把预测结果的一部分甚至根据预测结果才能得到的结论当作特征，从而导致模型看起来非常精确但在实际使用中的表现却很差，此时称作存在数据泄露。如果发现建立的模型非常精确，很可能存在数据泄露，应仔细检查与目标target相关的特征。

8.1 机器学习基本概念

（18）有监督学习（supervised learning）

在训练模型时，如果每个样本都有预期的标签或理想值，称作有监督学习。例如分类与回归问题。

（19）半监督学习（semi-supervised learning）

在训练模型时，可能只有部分训练数据带有标签或理想值，这种情况称作半监督学习。在半监督学习中，一般给没有标签的样本统一设置标签为-1。

（20）直推式学习（transductive learning）

直推式学习可以看作半监督学习的一个子问题，或者是一种特殊的半监督学习。直推式学习假设没有标签的数据就是最终用来测试的数据，学习的目的就是在这些数据上取得最佳泛化能力。

（21）无监督学习（unsupervised learning）

在训练模型时，如果每个样本都没有预期的标签或理想值，并且模型和算法的目的是试图发现样本之间的关系，称作无监督学习，例如聚类、密度估计、降维、主成分分析和离群值检测。

8.1 机器学习基本概念

（22）分类器（classifier）

具有有限个可能的离散值作为结果的有监督（或半监督）预测器。

（23）聚类器（clusterer）

聚类属于无监督学习算法，具有有限个可能的离散输出结果。

（24）评估器（estimator）

表示一个模型以及这个模型被训练和评估的方式，例如分类器、回归器、聚类器。

（25）离群点检测器（outlier detector）

区分核心样本和偏远样本的无监督二分类预测器。

（26）预测器（predictor）

支持predict()和/或fit_predict()方法的评估器，可以是分类器、回归器、离群点检测器和聚类器等。

（27）回归器（regressor）

处理连续输出值的有监督或半监督预测器。

（28）转换器（transformer）

支持transform()和/或fit_transform()方法的评估器。

8.1 机器学习基本概念

（29）交叉验证生成器（cross-validation generator）

在不同的测试集上执行多重评估，然后组合这些评估的得分，这种技术叫做交叉验证。交叉验证生成器用来把数据集分成训练集和测试集部分，提供split()和get_n_splits()方法，不提供fit()、set_params()和get_params()方法，不属于评估器。

（30）交叉验证评估器（cross-validation estimator）

具有内置的交叉验证能力、能够自动选择最佳超参数的评估器，例如ElasticNetCV和LogisticRegressionCV。

（31）评分器（scorer）

可调用对象，不属于评估器，使用给定的测试数据评价评估器，返回一个数值，数值越大表示评估器的性能或表现越好。

8.1 机器学习基本概念

（32）损失函数（loss function）

用来计算单个样本的预测结果与实际值之间误差的函数。

（33）风险函数（risk function）

损失函数的期望，表示模型预测的平均质量。

（34）代价函数（cost function）

用来计算整个训练集上所有样本的预测结果与实际值之间误差平均值的函数，值越小表示模型的鲁棒性越好，预测结果越准确。

（35）目标函数（objective function）

目标函数=代价函数+惩罚项。

8.1 机器学习基本概念

（36）坐标下降法（Coordinate Descent）

算法在每次迭代中在当前位置沿某个坐标的方向进行一维搜索和优化以求得函数的局部极小值，在整个过程中循环使用不同的坐标方向。如果在某次迭代中函数没有得到优化，说明已经达到一个驻点。

8.1 机器学习基本概念

（37）梯度下降法（Gradient Descent）

对于可微函数，其在每个方向上的偏导数组成的向量称作梯度，梯度的方向表示变化的方向，梯度的模长或幅度表示变化的快慢。在求解机器学习算法的模型参数时，梯度下降是经常使用的方法之一。在求解损失函数的最小值时，可以通过梯度下降法进行迭代求解，沿梯度的反方向进行搜索，当梯度向量的幅度接近0时终止迭代，最终得到最小化的损失函数和模型参数值。

8.1 机器学习基本概念

（38）随机梯度下降（Stochastic Gradient Descent,SGD）

随机梯度下降是一个用于拟合线性模型的简单但非常有效的方法，每次迭代时随机抽取并使用一组样本，尤其适用于样本数量和特征数量都非常大的场合，其partial_fit()方法允许在线学习或核外学习。

8.1 机器学习基本概念

（39）感知器（perceptron）

感知器是一个适用于大规模学习的简单分类算法，不需要学习率（监督学习和深度学习中决定目标函数能否收敛到局部最小值以及何时收敛到最小值的超参数），也不进行正则化，只在出错时更新模型，速度比SGD略快，得到的模型更稀疏。

（40）被动攻击算法（Passive Aggressive Algorithms）

被动攻击算法是一组用于大规模学习的算法，和感知器一样不需要学习率，但进行正则化。

8.1 机器学习基本概念

（41）泛化（generalization）

使用通过对已知数据进行学习得到的模型对未知数据进行预测的过程。

（42）正则化（regularization）

正则化是指在损失函数的基础上加上一个约束项（或称作惩罚项、正则项），目的是为了防止过拟合。

8.1 机器学习基本概念

（43）学习曲线（learning curve）

一种判断模型性能的方法，描述数据集样本数量的增加时模型得分变化情况，通过绘制学习曲线可以比较直观了解模型的状态。

（44）召回率（recall rate）

召回率也称查全率。对于分类算法而言，也就是所有样本中被识别为A类的样本数量与实际属于A类的样本数量的比值。

（45）准确率（precision）

对于分类算法而言，准确率定义为被识别为A类的样本中有多少确实属于A类。

8.2 机器学习库sklearn简介

模块名称	简单描述
base	包含所有评估器的基类和常用函数，例如测试评估器是否为分类器的函数is_classifier()和测试评估器是否为回归器的函数is_regressor()
calibration	包含预测概率校准的类和函数
cluster	包含常用的无监督聚类算法的实现，例如AffinityPropagation、AggomerativeClustering、Birch、DBSCAN、FeatureAgglomeration、KMeans、MiniBatchKMeans、MeanShift、SpectralClustering
covariance	包含用来估计给定点集协方差的算法实现
cross_decomposition	交叉分解模块，主要包含偏最小二乘法（PLS）和经典相关分析（CCA）算法的实现
datasets	包含加载常用参考数据集和生成模拟数据的工具

8.2 机器学习库sklearn简介

decomposition	包含矩阵分解算法的实现，包括主成分分析（PCA）、非负矩阵分解（NMF）、独立成分分析（ICA）等，该模块中大部分算法可以用作降维技术
discriminant_analysis	主要包含线型判别分析（LDA）和二次判别分析（QDA）算法
dummy	包含使用简单规则的分类器和回归器，可以作为比较其他真实分类器和回归器好坏的基线，不直接用于实际问题
ensemble	包含用于分类、回归和异常检测的集成方法
feature_extraction	从文本和图像原始数据中提取特征
feature_selection	包含特征选择算法的实现，目前有单变量过滤选择方法和递归特征消除算法
gaussian_process	实现了基于高斯过程的分类与回归
isotonic	保序回归

8.2 机器学习库sklearn简介

impute	包含用于填充缺失值的转换器
kernel_approximation	实现了几个基于傅里叶变换的近似核特征映射
kernel_ridge	实现了核岭回归
linear_model	实现了广义线型模型，包括线性回归、岭回归、贝叶斯回归、使用最小角回归和坐标下降法计算的Lasso和弹性网络评估器，还实现了随机梯度下降（SGD）相关的算法
manifold	流形学习，实现了数据嵌入技术
metrics	包含评分函数、性能度量、成对度量和距离计算
mixture	实现了高斯混合建模算法
model_selection	实现了多个交叉验证器类以及用于学习曲线、数据集分割的函数
multiclass	实现了多类和多标签分类，该模块中的估计器都属于元估计器，需要使用基估计器类作为参数传递给构造器。例如可以用来把一个二分类器或回归器转换为多类分类器

8.2 机器学习库sklearn简介

multioutput	实现了多输出回归与分类
naive_bayes	实现了朴素贝叶斯算法
neighbors	实现了k近邻算法
neural_network	实现了神经网络模型
pipeline	实现了用来构建混合评估器的工具
inspection	包含用于模型检测的工具
preprocessing	包含缩放、居中、正则化、二值化和插补算法
svm	实现了支持向量机（SVM）算法
tree	包含用于分类和回归的决策树模型
utils	包含一些常用工具，例如查找所有正数中最小值的函数arrayfuncs.min_pos() 计算稀疏向量密度的函数extmath.density()

8.2 机器学习库sklearn简介

方法名称	简单描述
fit()	每个评估器都会提供这个方法，通常接收一些样本X作为参数，如果模型是有监督的还会接收目标y作为参数，以及其他用来描述样本属性的参数，例如sample_weight。该方法应该具备以下功能： • 清除评估器之前存储的任意属性，除非使用了热启动 • 验证和解释参数，如果参数非法则引发异常 • 对输入的数据进行验证 • 根据评价的参数和给定的数据评估并存储模型属性 • 返回本次拟合好的评估器
fit_predict()	该方法的参数与fit()相同，但适用于无监督的直推式评估器，训练模型并返回对给定训练数据的预测结果。在聚类器中，这些预测结果也会保存在labels_属性中，fit_predict(X)的输出通常等价于fit(X).predict(X)的结果

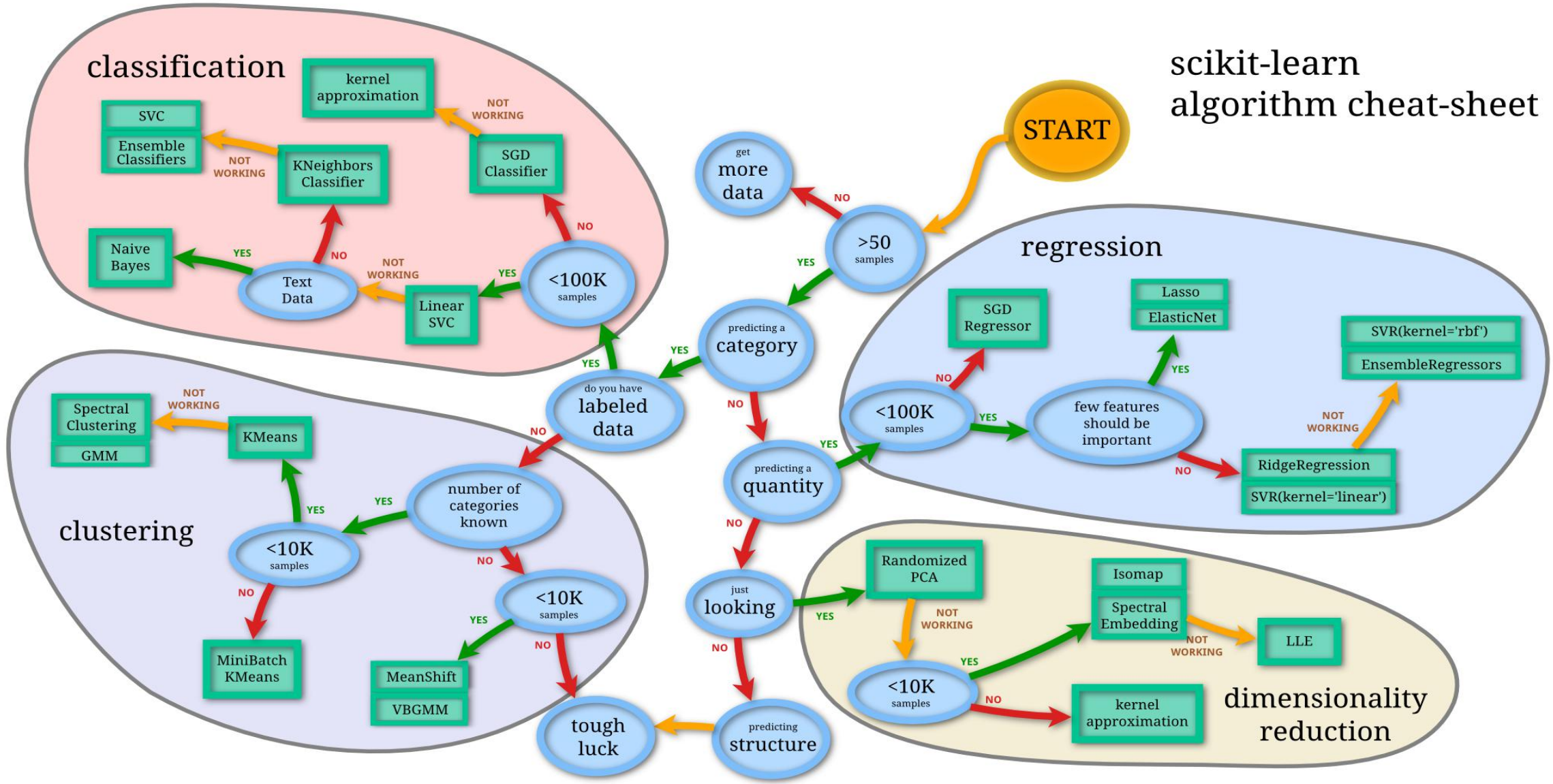
8.2 机器学习库sklearn简介

fit_transform()	转换器对象的方法，用来训练评估器并返回转换后的训练数据。该方法的参数与fit()相同，其输出应该和调用fit(X, ...).transform(X)具有相同的形状。但是在极个别情况中fit_transform(X, ...)和fit(X, ...).transform(X)的返回值并不一样，此时需要对训练数据进行不同的处理。 直推式转换器可能会提供fit_transform()方法而不提供transform()方法，这样比分别执行fit()和transform()更高效。 在归纳学习中，旨在从大量的经验数据中归纳抽取出一般的判定规则和模式，属于从特殊到一般的学习方法，目的是得到一个应用于新数据的广义模型，用户在进一步建模之前应该注意不要对整个数据集使用fit_transform()方法，否则会导致数据泄露。
get_feature_names()	特征提取器的基本方法，也适用于在评估器的transform()方法输出中获取每列名字的转换器。该方法输出包含若干字符串的列表，可以接收字符串列表作为输入用来指定输出的列名，默认情况下输入数据的特征被命名为x0, x1, ...
get_params()	获取所有参数和它们的值
partial_fit()	以在线方式训练评估器，重复调用该方法不会清除模型状态，而是使用给定的数据对模型进行更新

8.2 机器学习库sklearn简介

predict()	对每个样本进行预测，通常只接收样本矩阵X作为输入。在分类器或回归器中，预测结果必然是训练时使用的目标空间中的一个。尽管如此，即使传递给fit()方法的y是一个列表或其他类似于数组的数据，predict()方法的输出应该总是一个数组或稀疏矩阵。在聚类器或离群点检测中，预测结果是一个整数
predict_proba()	分类器和聚类器的方法，可以返回每个类的概率估计，其输入通常是包含样本特征向量的二维数组X
score()	预测器的方法，可以在给定数据集上评估预测结果，返回单个数值型得分，数值越大表示预测结果越好
split()	用于交叉验证而不是评估器，该方法接收参数(X, y, groups)，返回一个包含若干(train_idx, test_idx)元组的迭代器，用来把数据集分为训练集和测试集
transform()	在转换器中，对输入（通常只有X）进行转换，结果是一个长度为n_samples并且列数固定的数组或稀疏矩阵，评估器尚未完成时调用该方法会引发异常

8.2 机器学习库sklearn简介



8.3 线性回归算法原理与应用-8.3.1 线性回归模型原理

根据数学知识容易得知，对于平面上不重合的两个点 $P(x_1, y_1)$ 和 $Q(x_2, y_2)$，可以唯一确定一条直线 $\frac{y-y_1}{x-x_1}=\frac{y_2-y_1}{x_2-x_1}$，简单整理后得到 $y=\frac{y_2-y_1}{x_2-x_1}\times(x-x_1)+y_1$，根据这个公式就可以准确地计算任意 x 值在该直线上对应点的 y 值，如图 8-2 所示。

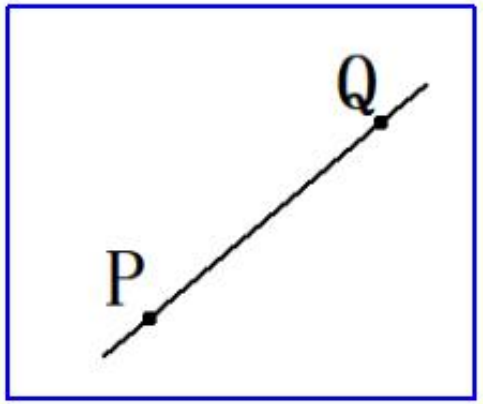


图 8-2　两点确定一条直线

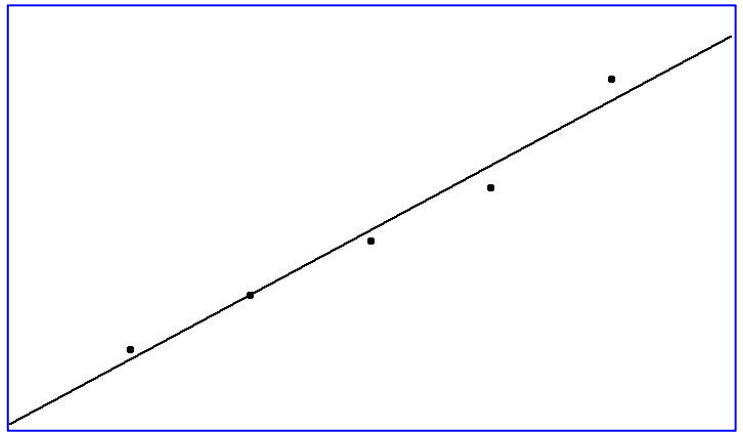
8.3.1 线性回归模型原理

如果平面上有若干样本点，并且这些点不共线。现在要求找到一条最佳回归直线，使得这些点的总离差最小，确定最佳回归系数 ω，满足下面的公式：

$$\min_{\omega} \|X\omega - y\|_2^2$$

其中，X 为包含若干 x 坐标的数组，$X\omega$ 为这些 x 坐标在回归直线上对应点的纵坐标，y 为样本点的实际纵坐标。

确定了最佳回归直线的方程之后，就可以对未知样本进行预测了，也就是计算任意 x 值在该直线上对应点的 y 值，如图 8-3 所示。



8.3.1 线性回归模型原理

使用线性回归算法对实际问题进行预测时，使用一个 $m \times n$ 的矩阵表示空间中的 m 个点（或样本）并作为参数 X 输入给模型的 fit()方法，每个样本是一个 $1 \times n$ 向量，该向量中每个分量表示样本的一个特征。根据观察值 X 和对应的目标 y 对线性回归模型进行训练和拟合，得到最佳“直线”并用来对未知的样本 x 进行预测。如果使用 $\bar{y}$ 表示预测结果，那么可以用下面的线性组合公式表示线性回归模型，

$$\bar{y}(w,x) = \omega_0 + \omega_1 x_1 + \omega_2 x_2 + \ldots + \omega_p x_p$$

其中，向量 $\omega = (\omega_1, \omega_2, \ldots, \omega_p)$ 为回归系数，使用 coef_表示；ω_0 为截距，使用 intercept_表示。在线性回归算法中，使用给定的观察值 X 和对应的目标 y 训练模型也就是计算最佳回归系数和截距的过程。

8.3.2 sklearn中线性回归模型的简单应用

```
>>> from sklearn import linear_model    # 导入线型模型模块
>>> regression = linear_model.LinearRegression()
                                        # 创建线型回归模型
>>> X = [[3], [8]]                      # 观察值的x坐标
>>> y = [1, 2]                          # 观察值的y坐标
>>> regression.fit(X, y)                # 拟合
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
>>> regression.intercept_               # 截距，以下画线结束
0.40000000000000036
>>> regression.coef_                    # 斜率，回归系数
                                        # 反映了x对y影响的大小
                                        # 以下画线结束，表示模型自身的属性
                                        # 区别于用户设置的参数

array([ 0.2])
>>> regression.predict([[6]])           # 对未知点进行预测，结果为数组
array([ 1.6])
```

8.3.3 岭回归原理与sklearn实现

岭回归是一种用于共线性数据分析的有偏估计回归方法，是一种改良的最小二乘估计法，通过放弃最小二乘法的无偏性，以损失部分信息、降低精度为代价从而获得更符合实际、更可靠的回归系数，对病态数据（这样的数据中某个元素的微小变动会导致计算结果误差很大）的拟合效果比最小二乘法好。岭回归通过在代价函数后面加上一个对参数的约束项（回归系数向量的l_2范数与一个常数α的乘积，称作 L2 正则化）来防止过拟合，即

$$\min_{\omega}\left\|X\omega-y\right\|_2^2+\alpha\left\|\omega\right\|_2^2$$

8.3.3 岭回归原理与sklearn实现

```
>>> from sklearn.linear_model import Ridge
>>> ridgeRegression = Ridge(alpha=10)          # 创建岭回归模型
                                               # 设置约束项系数为10
                                               # 数值越大，特征对结果的影响越小
>>> X = [[3], [8]]
>>> y = [1, 2]
>>> ridgeRegression.fit(X, y)                  # 拟合
Ridge(alpha=10, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001)
>>> ridgeRegression.predict([[6]])             # 预测
array([ 1.55555556])
>>> ridgeRegression.coef_                      # 查看回归系数
array([ 0.11111111])
>>> ridgeRegression.intercept_                 # 截距
0.8888888888888895
```

8.3.3 岭回归原理与sklearn实现

```
>>> ridgeRegression = Ridge(alpha=1.0)      # 设置约束项系数为1.0
>>> ridgeRegression.fit(X, y)
Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001)
>>> ridgeRegression.coef_
array([ 0.18518519])
>>> ridgeRegression.intercept_
0.48148148148148162
>>> ridgeRegression.predict([[6]])
array([ 1.59259259])
```

8.3.3 岭回归原理与sklearn实现

```
>>> ridgeRegression = Ridge(alpha=0.0)       # 约束项系数为0
                                             # 等价于线性回归
>>> ridgeRegression.fit(X, y)
Ridge(alpha=0.0, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001)
>>> ridgeRegression.coef_
array([ 0.2])
>>> ridgeRegression.intercept_
0.39999999999999991
>>> ridgeRegression.predict([[6]])
array([ 1.6])
```

8.3.3 岭回归原理与sklearn实现

```
>>> import numpy as np
>>> from sklearn.linear_model import RidgeCV
>>> X = [[3], [8]]
>>> y = [1, 2]
>>> reg = RidgeCV(alphas=np.arange(-10,10,0.2))  # 指定alpha参数的范围
>>> reg.fit(X, y)                                # 拟合
RidgeCV(alphas=array([-10. ,  -9.8, ...,   9.6,   9.8]), cv=None,
    fit_intercept=True, gcv_mode=None, normalize=False, scoring=None,
    store_cv_values=False)
>>> reg.alpha_                                   # 最佳数值
                                                 # 拟合之后该值才可用
0.9999999999996092
>>> reg.predict([[6]])                           # 预测
array([ 1.59259259])
```

8.3.4 套索回归Lasso基本原理与sklearn实现

Lasso 是可以估计稀疏系数的线性模型，尤其适用于减少给定解决方案依赖的特征数量的场合。如果数据的特征过多，而其中只有一小部分是真正重要的，此时选择 Lasso 比较合适。

在数学表达上，Lasso 类似于岭回归，也是在代价函数基础上增加了一个惩罚项的线性模型，区别在于 Lasso 的正则项为系数向量的l_1范数与一个常数α的乘积（称作 L1 正则化），目标函数的形式为

$$\min_{\omega} \|X\omega - y\|_2^2 + \alpha\|\omega\|_1$$

8.3.4 套索回归Lasso基本原理与sklearn实现

```
>>> from sklearn.linear_model import Lasso
>>> X = [[3], [8]]
>>> y = [1, 2]
>>> reg = Lasso(alpha=3.0)                    # 惩罚项系数为3.0
>>> reg.fit(X, y)                             # 拟合
Lasso(alpha=3.0, copy_X=True, fit_intercept=True, max_iter=1000,
   normalize=False, positive=False, precompute=False, random_state=None,
   selection='cyclic', tol=0.0001, warm_start=False)
>>> reg.coef_                                 # 查看系数
array([ 0.])
>>> reg.intercept_                            # 查看截距
1.5
>>> reg.predict([[6]])
array([ 1.5])
```

8.3.4 套索回归Lasso基本原理与sklearn实现

```
>>> reg = Lasso(alpha=0.2)
>>> reg.fit(X, y)
Lasso(alpha=0.2, copy_X=True, fit_intercept=True, max_iter=1000,
   normalize=False, positive=False, precompute=False, random_state=None,
   selection='cyclic', tol=0.0001, warm_start=False)
>>> reg.coef_
array([ 0.168])
>>> reg.intercept_
0.5759999999999996
>>> reg.predict([[6]])
array([ 1.584])
```

8.3.5 弹性网络基本原理与sklearn实现

弹性网络 ElasticNet 是同时使用了系数向量的 l_1 范数和 l_2 范数的线性回归模型，使得可以学习得到类似于 Lasso 的一个稀疏模型，同时还保留了 Ridge 的正则化属性，结合了二者的优点，尤其适用于有多个特征彼此相关的场合。在数学上，目标函数形式为

$$\min_{\omega} \frac{1}{2n_{samples}} \|X\omega - y\|_2^2 + \alpha\rho\|\omega\|_1 + \frac{\alpha(1-\rho)}{2}\|\omega\|_2^2$$

8.3.5 弹性网络基本原理与sklearn实现

```
>>> from sklearn.linear_model import ElasticNet
>>> reg = ElasticNet(alpha=1.0, l1_ratio=0.7)
>>> X = [[3], [8]]
>>> y = [1, 2]
>>> reg.fit(X, y)
ElasticNet(alpha=1.0, copy_X=True, fit_intercept=True, l1_ratio=0.7,
  max_iter=1000, normalize=False, positive=False, precompute=False,
  random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
>>> reg.predict([[6]])
array([ 1.54198473])
>>> reg.coef_
array([ 0.08396947])
>>> reg.intercept_
1.038167938931297 7
```

8.3.5 弹性网络基本原理与sklearn实现

```
>>> reg = ElasticNet(alpha=1.0, l1_ratio=0.3)    # 修改参数，进行对比
>>> reg.fit(X, y)
ElasticNet(alpha=1.0, copy_X=True, fit_intercept=True, l1_ratio=0.3,
  max_iter=1000, normalize=False, positive=False, precompute=False,
  random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
>>> reg.predict([[6]])
array([ 1.56834532])
>>> reg.coef_
array([ 0.13669065])
>>> reg.intercept_
0.74820143884892099
```

8.3.6 使用线性回归模型预测儿童身高

- 理论上，一个人的身高除了随年龄变大而增长之外，在一定程度上还受到遗传和饮食习惯以及其他因素的影响。在这里我们把问题简化一下，假定一个人的身高只受年龄、性别、父母身高、祖父母身高和外祖父母身高这几个因素的影响，并假定大致符合线性关系。
- 也就是说，在其他条件不变的情况下，随着年龄的增长，会越来越高；同样，对于其他条件都相同的儿童，其父母身高较大的话，儿童也会略高一些。但在实际应用时要考虑到一个情况，人的身高不是一直在变高的，到了一定年龄之后就不再生长了，然后身高会长期保持固定而不再变化（不考虑年龄太大之后会稍微变矮一点的情况）。为了简化问题，我们假设18岁之后身高不再变化。

8.3.6 使用线性回归模型预测儿童身高

```
import copy
import numpy as np
from sklearn import linear_model

# 训练数据，每一行表示一个样本，包含的信息分别为：
# 儿童年龄,性别（0女1男）
# 父亲、母亲、祖父、祖母、外祖父、外祖母的身高
x = np.array([[1, 0, 180, 165, 175, 165, 170, 165],
              [3, 0, 180, 165, 175, 165, 173, 165],
              [4, 0, 180, 165, 175, 165, 170, 165],
              [6, 0, 180, 165, 175, 165, 170, 165],
              [8, 1, 180, 165, 175, 167, 170, 165],
              [10, 0, 180, 166, 175, 165, 170, 165],
              [11, 0, 180, 165, 175, 165, 170, 165],
              [12, 0, 180, 165, 175, 165, 170, 165],
              [13, 1, 180, 165, 175, 165, 170, 165],
              [14, 0, 180, 165, 175, 165, 170, 165],
              [17, 0, 170, 165, 175, 165, 170, 165]])

# 儿童身高，单位：cm
y = np.array([60, 90, 100, 110, 130, 140, 150, 164, 160, 163, 168])
```

8.3.6 使用线性回归模型预测儿童身高

```
# 创建线性回归模型
lr = linear_model.LinearRegression()
# 根据已知数据拟合最佳直线
lr.fit(x, y)

# 待测的未知数据，其中每个分量的含义和训练数据相同
xs = np.array([[10, 0, 180, 165, 175, 165, 170, 165],
               [17, 1, 173, 153, 175, 161, 170, 161],
               [34, 0, 170, 165, 170, 165, 170, 165]])

for item in xs:
    # 为不改变原始数据，进行深复制，并假设超过18岁以后就不再长高了
    # 对于18岁以后的年龄，返回18岁时的身高
    item1 = copy.deepcopy(item)
    if item1[0] > 18:
        item1[0] = 18
    print(item, ':', lr.predict(item1.reshape(1,-1)))
```

8.4 逻辑回归算法原理与应用-8.4.1 逻辑回归算法原理与sklearn实现

- 虽然名字中带有“回归”二字，但实际上逻辑回归是一个用于分类的线性模型，通常也称作最大熵分类或对数线性分类器。
- 逻辑回归的因变量既可以是二分类的，也可以是多分类的，但是二分类更常用一些。逻辑回归常用于数据挖掘、疾病自动诊断、经济预测等领域，例如可以挖掘引发疾病的主要因素，或根据这些因素来预测发生疾病的概率。

8.4.1 逻辑回归算法原理与sklearn实现

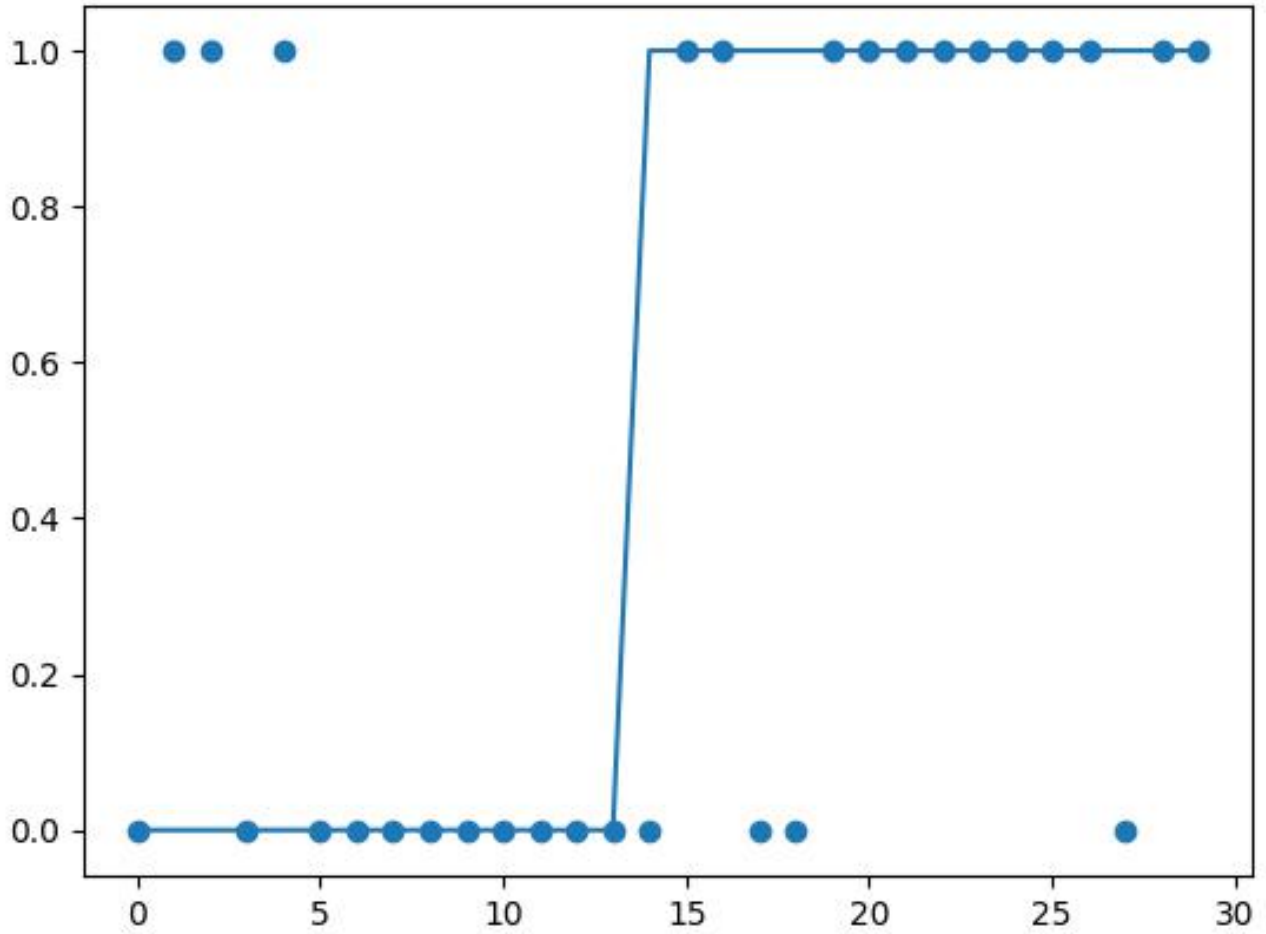
参数名称	含义
penalty	用来指定惩罚时的范数，默认为'l2'，也可以为'l1'，但求解器'newton-cg'、'sag'和'lbfgs'只支持'l2'
C	用来指定正则化强度的逆，必须为正实数，值越小表示正则化强度越大（这一点和支持向量机类似），默认值为1.0
solver	用来指定优化时使用的算法，该参数可用的值有'newton-cg'、'lbfgs'、'liblinear'、'sag'、'saga'，默认值为'liblinear'
multi_class	取值可以为'ovr'或'multinomial'，默认值为'ovr'。如果设置为'ovr'，对于每个标签拟合二分类问题，否则在整个概率分布中使用多项式损失进行拟合，该参数不适用于'liblinear'求解器
n_jobs	用来指定当参数multi_class='ovr'时使用的CPU核的数量，值为-1时表示使用所有的核

8.4.1 逻辑回归算法原理与sklearn实现

方法	功能
fit(self, X, y, sample_weight=None)	根据给定的训练数据对模型进行拟合
predict_log_proba(self, X)	对数概率估计，返回的估计值按分类的标签进行排序
predict_proba(self, X)	概率估计，返回的估计值按分类的标签进行排序
predict(self, X)	预测X中样本所属类的标签
score(self, X, y, sample_weight=None)	返回给定测试数据和实际标签相匹配的平均准确率
densify(self)	把系数矩阵转换为密集数组格式
sparsify(self)	把系数矩阵转换为稀疏矩阵格式

8.4.1 逻辑回归算法原理与sklearn实现

```
import numpy as np
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt

# 构造测试数据
X = np.array([[i] for i in range(30)])
y = np.array([0]*15+[1]*15)
# 人为修改部分样本的值
y[np.random.randint(0,15,3)] = 1
y[np.random.randint(15,30,4)] = 0
print(y[:15])
print(y[15:])
# 根据原始数据绘制散点图
plt.scatter(X, y)

# 创建并训练逻辑回归模型
reg = LogisticRegression('l2', C=3.0)
reg.fit(X, y)

# 对未知数据进行预测
print(reg.predict([[5], [19]]))
# 未知数据属于某个类别的概率
print(reg.predict_proba([[5], [19]]))

# 对原始观察点进行预测
yy = reg.predict(X)
# 根据预测结果绘制折线图
plt.plot(X, yy)

plt.show()
```



8.4.2 使用逻辑回归算法预测考试能否及格

```
from sklearn.linear_model import LogisticRegression

# 复习情况，格式为(时长,效率)，其中时长单位为小时
# 效率为[0,1]之间的小数，数值越大表示效率越高
X_train = [(0,0), (2,0.9), (3,0.4), (4,0.9), (5,0.4), (6,0.4), (6,0.8), (6,0.7), (7,0.2), (7.5,0.8),
           (7,0.9), (8,0.1), (8,0.6), (8,0.8)]
# 0表示不及格，1表示及格
y_train = [0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1]

# 创建并训练逻辑回归模型
reg = LogisticRegression()
reg.fit(X_train, y_train)

# 测试模型
X_test = [(3,0.9), (8,0.5), (7,0.2), (4,0.5), (4,0.7)]
y_test = [0, 1, 0, 0, 1]
score = reg.score(X_test, y_test)

# 预测并输出预测结果
learning = [(8, 0.9)]
result = reg.predict_proba(learning)
msg = '''模型得分：{0}
复习时长为：{1[0]}，效率为：{1[1]}
您不及格的概率为：{2[0]}
您及格的概率为：{2[1]}
综合判断，您会：{3}'''.format(score, learning[0], result[0], '不及格' if result[0][0]>0.5 else '及格')
print(msg)
```

8.5 朴素贝叶斯算法原理与应用-8.5.1 基本概念

（1）随机试验

随机试验是指这样的试验，可以在相同条件下重复试验多次，所有可能发生的结果都是已知的，但每次试验到底会发生其中哪一种结果是无法预先确定的。

（2）事件与空间

在一个特定的试验中，每一个可能出现的结果称作一个基本事件，全体基本事件组成的集合称为基本空间。

在一定条件下必然会发生的事件称为必然事件，可能发生也可能不发生的事件称为随机事件，不可能发生的事件称为不可能事件，不可能同时发生的两个事件称为互斥事件，二者必有其一发生的事件称为对立事件。

例如，在水平地面上投掷硬币的试验中，正面朝上是一个基本事件，反面朝上是一个基本事件，基本空间中只包含这两个随机事件，并且二者既为互斥事件又是对立事件。

8.5.1 基本概念

（3）概率

概率是用来描述在特定试验中一个事件发生的可能性大小的指标，是介于0和1之间的实数，可以定义为某个事件发生的次数与试验总次数的比值，即

$$p(x)=\frac{n_x}{n}$$

其中 n_x 表示事件 x 发生的次数， n 表示实验总次数。

例如，在投掷硬币的试验中，对于材质均匀的硬币，在水平地面上投掷足够多次，那么正面朝上和反面朝上这两个事件的概率都是0.5。

8.5.1 基本概念

（4）先验概率

先验概率是指，根据以往的经验和分析得到的概率。

例如，上一段关于投掷硬币的试验描述中，0.5 就是先验概率。再例如，有 5 张卡片，上面分别写着数字 1、2、3、4、5，随机抽取一张，取到偶数卡片的概率是 $\frac{2}{5}$，这也是先验概率。

8.5.1 基本概念

（5）条件概率

条件概率也称后验概率，是指在另一个事件 B 已经发生的情况下事件 A 发生的概率，记为 $p(A \mid B)$。如果基本空间只有两个事件 A 和 B 的话，有

$$p(A \cap B) = p(A \mid B)p(B) = p(B \mid A)p(A)$$

或

$$p(A \mid B) = \frac{p(A \cap B)}{p(B)}$$

以及

$$p(B \mid A) = \frac{p(A \cap B)}{p(A)}$$

8.5.1 基本概念

（6）全概率公式

已知若干互不相容的事件B_i，其中$i=1,2,3,...,n$，并且所有事件B_i构成基本空间，那么对于任意事件A，有

$$p(A)=\sum_{i=1}^{n}p(A|B_i)p(B_i)$$

这个公式称为全概率公式，可以把复杂事件A的概率计算转化为不同情况下发生的简单事件的概率求和问题。

8.5.1 基本概念

（7）贝叶斯理论

贝叶斯理论用来根据一个已发生事件的概率计算另一个事件发生的概率，即

$$p(A|B)p(B) = p(B|A)p(A)$$

或

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)}$$

8.5.2 朴素贝叶斯算法分类原理与sklearn实现

朴素贝叶斯算法之所以说“朴素”，是指在整个过程中只做最原始、最简单的假设，例如假设特征之间互相独立并且所有特征同等重要。

使用朴素贝叶斯算法进行分类时，计算未知样本属于所有已知类的概率，然后选择其中概率最大的类作为分类结果。根据贝叶斯理论，样本x属于某个类c_i的概率计算公式为

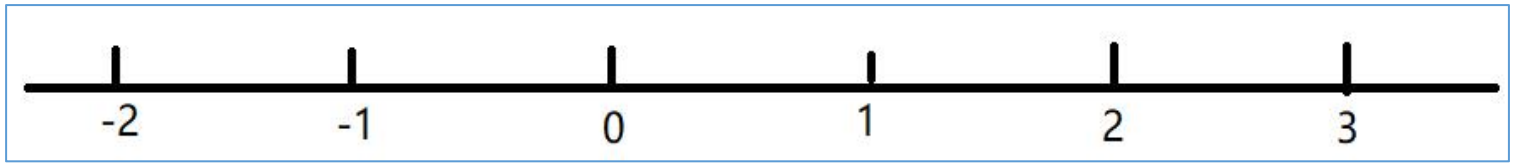
$$p(c_i \mid x) = \frac{p(x \mid c_i)p(c_i)}{p(x)}$$

然后在所有条件概率$p(c_1 \mid x)$、$p(c_2 \mid x)$、$p(c_3 \mid x)$、...、$p(c_n \mid x)$中选择最大的那个，例如$p(c_k \mid x)$，并判定样本x属于类c_k。

例如，如果邮件中包含“发票”、“促销”、“微信”或“电话”之类的词汇，并且占比较高或组合出现，那么这封邮件是垃圾邮件的概率会比没有这些词汇的邮件要大一些。

8.5.2 朴素贝叶斯算法分类原理与sklearn实现

```
>>> import numpy as np
>>> X = np.array([[-1, -1], [-2, -1], [-3, -2],
                  [1, 1], [2, 1], [3, 2]])
>>> y = np.array([1, 1, 1, 2, 2, 2])
>>> from sklearn.naive_bayes import GaussianNB
>>> clf = GaussianNB()                          # 创建高斯朴素贝叶斯模型
>>> clf.fit(X, y)                               # 拟合
GaussianNB(priors=None)
>>> clf.predict([[-0.8, -1]])                   # 分类
array([1])
>>> clf.predict_proba([[-0.8, -1]])             # 样本属于不同类别的概率
array([[  9.99999949e-01,   5.05653254e-08]])
>>> clf.score([[-0.8,-1]], [1])                 # 评分
1.0
>>> clf.score([[-0.8,-1], [0,0]], [1,2])        # 评分
0.5
```



8.5.3 使用朴素贝叶斯算法对中文邮件进行分类

1）从电子邮箱中收集足够多的垃圾邮件和非垃圾邮件的内容作为训练集。

2）读取全部训练集，删除其中的干扰字符，例如【】*。、，等等，然后分词，再删除长度为1的单个字，这样的单个字对于文本分类没有贡献，剩下的词汇认为是有效词汇。

3）统计全部训练集中每个有效词汇的出现次数，截取出现次数最多的前N（可以根据实际情况进行调整）个。

4）根据每个经过第2步预处理后的垃圾邮件和非垃圾邮件内容生成特征向量，统计第3步中得到的N个词语分别在该邮件中的出现频率。每个邮件对应于一个特征向量，特征向量长度为N，每个分量的值表示对应的词语在本邮件中出现的次数。例如，特征向量[3, 0, 0, 5]表示第一个词语在本邮件中出现了3次，第二个和第三个词语没有出现，第四个词语出现了5次。

5）根据第4步中得到特征向量和已知邮件分类创建并训练朴素贝叶斯模型。

6）读取测试邮件，参考第2步，对邮件文本进行预处理，提取特征向量。

7）使用第5步中训练好的模型，根据第6步提取的特征向量对邮件进行分类。

8.5.3 使用朴素贝叶斯算法对中文邮件进行分类

- 参考代码：

code\贝叶斯中文邮件分类\贝叶斯垃圾邮件分类器.py

补充：保存和加载训练结果

code\贝叶斯中文邮件分类\get_words_from_file.py

code\贝叶斯中文邮件分类\贝叶斯垃圾邮件分类器_训练并保存结果.py

code\贝叶斯中文邮件分类\贝叶斯垃圾邮件分类器_加载并使用训练结果.py

补充：文本中词频向量化

```
>>> from sklearn.feature_extraction.text import CountVectorizer
>>> cv = CountVectorizer()
>>> cv.fit_transform(["aaa aaa aaa bbb","bbb bbb bbb ccc ccc"]).toarray()
array([[3, 1, 0],
       [0, 3, 2]], dtype=int64)
>>> cv.fit_transform(["aaa aaa aaa bbb ddd eee","bbb bbb bbb ccc ccc"]) # 稀疏矩阵
<2x5 sparse matrix of type '<class 'numpy.int64'>'
       with 6 stored elements in Compressed Sparse Row format>
>>> cv.fit_transform(["aaa aaa aaa bbb ddd eee","bbb bbb bbb ccc ccc"]).toarray()
array([[3, 1, 0, 1, 1],
       [0, 3, 2, 0, 0]], dtype=int64)
```

8.6 决策树与随机森林算法应用-8.6.1 基本概念

（1）熵

熵表示的是数据中包含的信息量大小或着数据的混乱程度。熵越小，数据的纯度越高，数据越趋于一致，混乱程度越低；熵越大，数据的纯度越低，数据混乱程度越高。熵的计算公式为

$$Entropy = -\sum_{i=1}^{n} p_i \cdot \log(p_i)$$

其中，p_i 表示第 i 个类样本的出现概率或数量占比。

（2）信息增益

信息增益 = 分裂前熵 - 分裂后熵。信息增益越大，则意味着使用某几个属性来进行分裂节点创建子节点所获得的 “纯度提升” 越大。分裂后所有子节点的纯度都应高于父节点。

8.6.1 基本概念

（3）信息增益率

信息增益率 = 信息增益 / 分裂前熵。信息增益率越高，说明分裂的效果越好。

（4）基尼值

基尼值也称基尼指数，计算公式为

$$Gini = 1 - \sum_{i=1}^{n} p_i^2$$

其中，p_i 表示第 i 个类样本的数量占比。

容易得知，基尼值越大，表示数据纯度越低，也表示从样本空间中随机选取两个样本时这两个样本所属类别不一样的概率越大。

8.6.1 基本概念

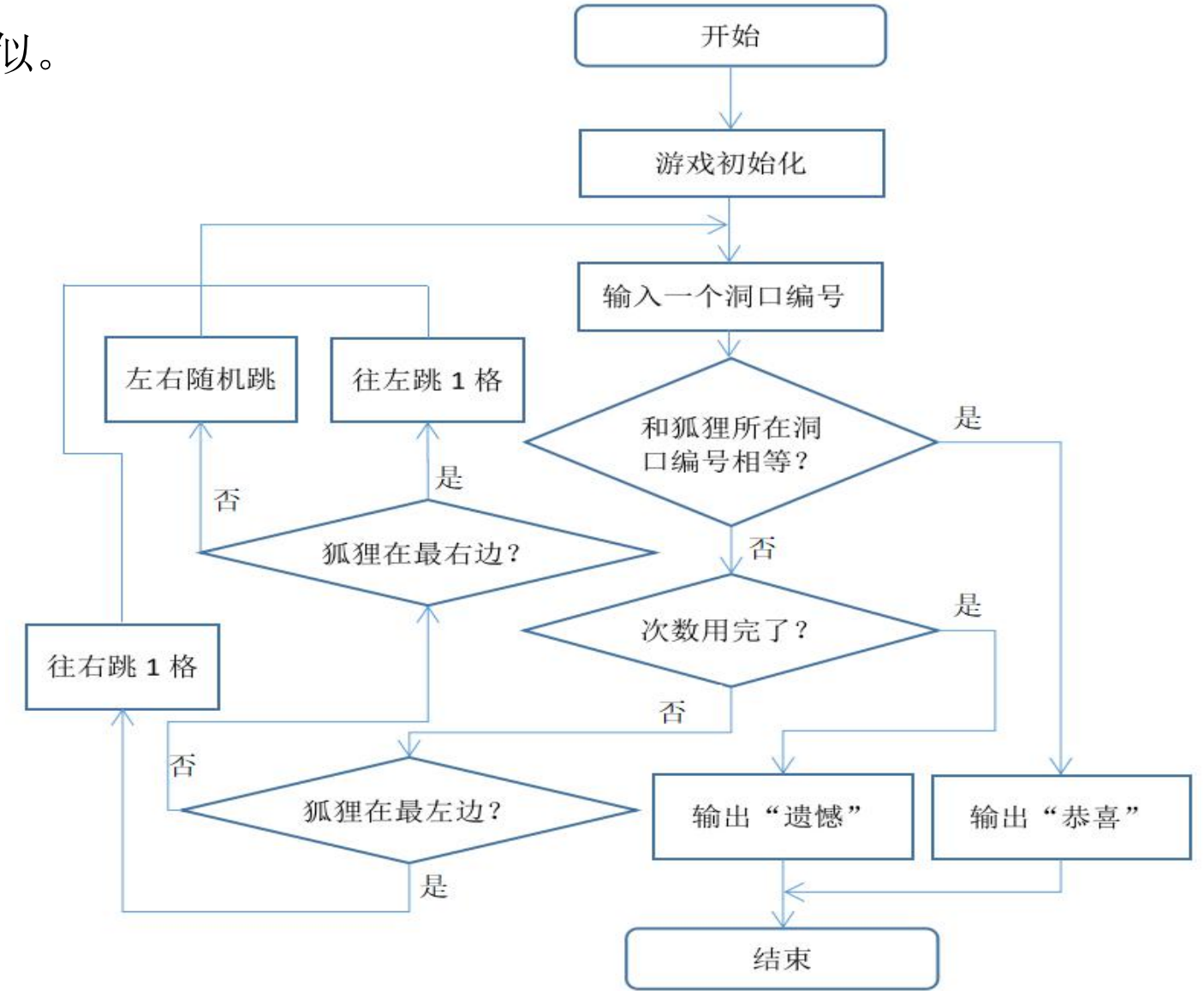
例如，有 10 个样本，如果这 10 个样本全部属于类别 A，那么基尼值为$1-(1^2+0^2)=0$，从这 10 个样本中任选 2 个样本属于不同类别的概率为 0，此时数据的纯度最高；如果其中 7 个属于类别 A 而另外 3 个属于类别 B，那么基尼值为$1-(0.7^2+0.3^2)=0.42$；如果有 5 个属于类别 A 而另外 5 个属于类别 B，那么基尼值为$1-(0.5^2+0.5^2)=0.5$；如果有 3 个属于类别 A、3 个属于类别 B、4 个属于类别 C，那么基尼值为$1-(0.3^2+0.3^2+0.4^2)=0.66$；如果这 10 个样本分别属于 10 个不同的类别，那么基尼值为$1-10\times0.1^2=0.9$。

8.6.2 决策树算法原理与sklearn实现

- 简单地说，决策树算法相当于一个多级嵌套的选择结构，通过回答一系列问题来不停地选择树上的路径，最终到达一个表示某个结论或类别的叶子节点，例如有无贷款意向、能够承担的理财风险等级、根据高考时各科成绩填报最合适的学校和专业、寻找最佳伴侣、判断一个人的诚信度、判断商场是否应该引进某种商品、发现不好意思申请补助的贫困生、预测明天是晴天还是阴天、预测一条狗主动攻击人的可能性、使用专家系统进行疾病诊断。

8.6.2 决策树算法原理与sklearn实现

- 程序流程图和决策树类似。



8.6.2 决策树算法原理与sklearn实现

- 决策树属于有监督学习算法，需要根据已知样本数据及其目标来训练并得到一个可以工作的模型，然后再使用该模型对未知样本进行分类。
- 在决策树算法中，构造一棵完整的树并用来分类所需要的计算量和空间复杂度都非常高，可以采用剪枝算法在保证模型性能的前提下删除不必要的分支。剪枝有预先剪枝和后剪枝两大类方法
  - ✓预先剪枝算法是指在树的生长过程中设定一个指标，当达到指标时就停止生长，当前节点确定为叶子节点并不再分裂。适合大样本集的情况，但有可能会导致模型的误差比较大。
  - ✓后剪枝算法可以充分利用全部训练集的信息，但计算量和空间复杂度都要大很多，一般用于小样本的场合。

8.6.2 决策树算法原理与sklearn实现

- 决策树有多种实现，常见的有ID3（Iterative Dichotomiser 3）、C4.5、C5.0和CART，其中ID3、C4.5、C5.0是属于分类树，CART属于分类回归树。
- ID3以信息论为基础，以信息熵和信息增益为衡量标准，实现对数据的归纳分类。ID3算法从根节点开始，在每个节点上计算所有可能的特征的信息增益，选择信息增益最大的一个特征作为该节点的特征进行分裂并创建子节点，不断递归这个过程直到完成决策树的构建。ID3适合二分类问题，且仅能处理离散属性。
- C4.5是对ID3的一种改进，根据信息增益率选择属性，在构造树的过程中进行剪枝操作，能够对连续属性进行离散化。该算法先将特征值排序，以连续两个值的中间值作为划分标准。尝试每一种划分，并计算修正后的信息增益，选择信息增益率最大的分裂点作为该属性的分裂点。

8.6.2 决策树算法原理与sklearn实现

- 分类决策树DecisionTreeClassifier类构造方法的语法为：

```
__init__(self, criterion='gini', splitter='best', max_depth=None,
min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
max_features=None, random_state=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None,
presort=False)
```

8.6.2 决策树算法原理与sklearn实现

参数名称	含义
criterion	用来指定衡量分裂（创建子节点）质量的标准，取值为'gini'时使用基尼值，为'entropy'时使用信息增益
splitter	用来指定在每个节点选择划分的策略，可以为'best'或'random'
max_depth	用来指定树的最大深度，如果不指定则一直扩展节点，直到所有叶子包含的样本数量少于min_samples_split，或者所有叶子节点都不再可分
min_samples_split	用来指定分裂节点时要求的最小样本数量，值为实数时表示百分比
min_samples_leaf	叶子节点要求的最小样本数量
max_features	用来指定在寻找最佳分裂时考虑的特征数量
max_leaf_nodes	用来设置叶子最大数量
min_impurity_decrease	如果一个节点分裂后可以使得不纯度减少的值大于或等于min_impurity_decrease，则对该节点进行分裂
min_impurity_split	用来设置树的生长过程中早停的阈值，如果一个节点的不纯度高于这个阈值则进行分裂，否则为一个叶子不再分裂
presort	用来设置在拟合时是否对数据进行预排序来加速寻找最佳分裂的过程

8.6.2 决策树算法原理与sklearn实现

方法	功能
fit(self, X, y, sample_weight=None, check_input=True, X_idx_sorted=None)	根据给定的训练集构建决策树分类器
predict_log_proba(self, X)	预测样本集X属于不同类别的对数概率
predict_proba(self, X, check_input=True)	预测样本集X属于不同类别的概率
apply(self, X, check_input=True)	返回每个样本被预测的叶子索引
decision_path(self, X, check_input=True)	返回树中的决策路径
predict(self, X, check_input=True)	返回样本集X的类别或回归值
score(self, X, y, sample_weight=None)	根据给定的数据和标签计算模型精度的平均值

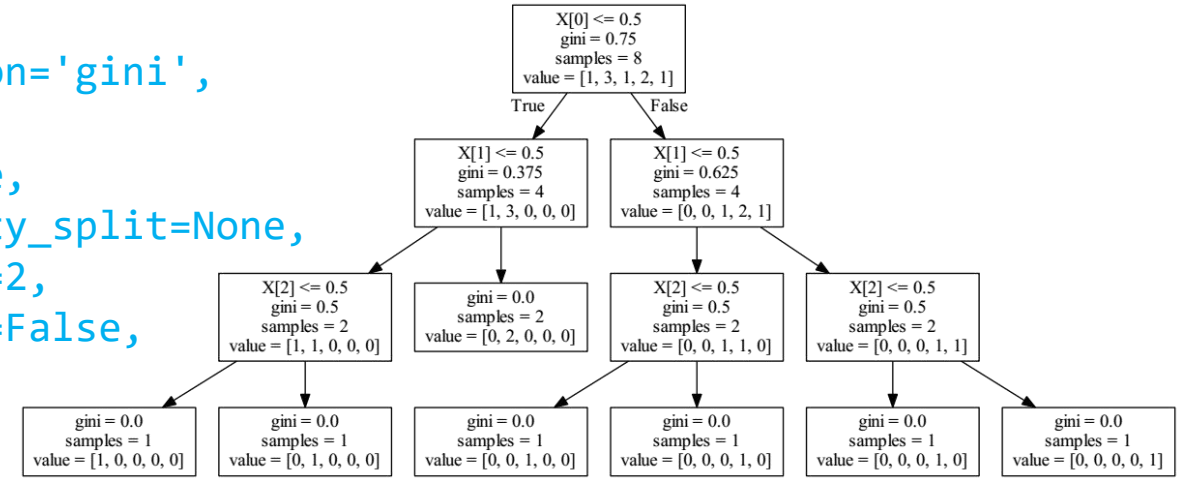
8.6.2 决策树算法原理与sklearn实现

- sklearn.tree模块的函数export_graphviz()可以用来把训练好的决策树数据导出，然后再使用扩展库graphviz中的功能绘制决策树图形，export_graphviz()函数语法为：

```
export_graphviz(decision_tree, out_file="tree.dot", max_depth=None,
feature_names=None, class_names=None, label='all', filled=False,
leaves_parallel=False, impurity=True, node_ids=False, proportion=False,
rotate=False, rounded=False, special_characters=False, precision=3)
```

8.6.2 决策树算法原理与sklearn实现

```
>>> import numpy as np
>>> from sklearn import tree
>>> X = np.array([[0, 0, 0], [0, 0, 1], [0, 1, 0], [0, 1, 1],
                  [1, 0, 0], [1, 0, 1], [1, 1, 0], [1, 1, 1]])
>>> y = [0, 1, 1, 1, 2, 3, 3, 4]
>>> clf = tree.DecisionTreeClassifier()   # 创建决策树分类器
>>> clf.fit(X, y)                         # 拟合
DecisionTreeClassifier(class_weight=None, criterion='gini',
            max_depth=None,
            max_features=None, 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=None, splitter='best')
>>> clf.predict([[1, 0, 0]])                    # 分类
array([2])
>>> import graphviz
>>> dot_data = tree.export_graphviz(clf, out_file=None)  # 导出决策树
>>> graph = graphviz.Source(dot_data)                    # 创建图形
>>> graph.render('result')                               # 输出PDF文件
'result.pdf'
```



8.6.3 随机森林算法原理与sklearn实现

- 随机森林是一种集成学习方法，基本思想是把几棵不同参数的决策树打包到一起，每棵决策树单独进行预测，然后计算所有决策树预测结果的平均值（适用于回归分析）或所有决策树“投票”得到最终结果（适用于分类）。
- 在随机森林算法中，不会让每棵树都生成最佳的节点，而是在每个节点上随机选择一个特征进行分裂。

8.6.3 随机森林算法原理与sklearn实现

- 扩展库sklearn在ensemble模块中提供了随机森林分类器RandomForestClassifier和随机森林回归器RandomForestRegressor。本节重点介绍随机森林分类器的用法，该类构造方法语法为

```
__init__(self, n_estimators=10, criterion='gini', max_depth=None,
min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1,
random_state=None, verbose=0, warm_start=False, class_weight=None)
```

8.6.3 随机森林算法原理与sklearn实现

参数名称	含义
n_estimators	用来指定森林中树的数量，默认为10
criterion	用来指定衡量分裂（创建子节点）质量的函数，取值为'gini'时使用基尼值，为'entropy'时使用信息增益
max_features	用来指定寻找最佳分裂时考虑的特征数量，可以是整数，也可以是实数（表示百分比）、'auto'（相当于max_features=sqrt(n_features)）、'sqrt'（与'auto'含义相同）、'log2'（相当于max_features=log2(n_features)）、None（相当于max_features=n_features）
max_depth	用来指定树的最大深度
min_samples_split	用来指定分裂节点时要求的样本数量最小值，值为实数时表示百分比
min_samples_leaf	用来指定叶子节点要求的样本数量最小值
max_leaf_nodes	在最佳优先方式中使用该参数生成树
min_impurity_split	在树的生长过程中早停的阈值，如果一个节点的不纯度高于该阈值则进行分裂，否则为叶子节点
min_impurity_decrease	如果一个节点分裂后带来的不纯度减少的量大于等于该参数的值，就对该节点进行分裂
bootstrap	用来设置在构建树时是否可以重复使用同一个样本
oob_score	用来设置是否使用out-of-bag样本（本次没有使用的样本）估计泛化精度

8.6.3 随机森林算法原理与sklearn实现

方法	功能
predict(self, X)	预测样本集X中样本的目标
apply(self, X)	把森林中的树应用到样本集X，返回叶子索引
decision_path(self, X)	返回森林中的决策路径
fit(self, X, y, sample_weight=None)	根据训练集(X, y)构建包含若干决策树的森林
score(self, X, y, sample_weight=None)	根据样本集和对应的真实值计算并返回模型得分

8.6.3 随机森林算法原理与sklearn实现

```
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier

# 生成测试数据
X, y = make_classification(n_samples=800,      # 800个样本
                           n_features=6,       # 每个样本6个特征
                           n_informative=4,    # 4个有用特征
                           n_redundant=2,      # 2个冗余特征
                           n_classes=2,        # 全部样本分2类
                           shuffle=True)
clf = RandomForestClassifier(n_estimators=4,   # 4个决策树
                             max_depth=3,      # 最多3层
                             criterion='gini', max_features=0.1, min_samples_split=5)
clf.fit(X, y)
# 包含拟合好的决策树的列表
print('决策树列表：\n', clf.estimators_)
# 类标签
print('类标签列表：\n', clf.classes_)
# 执行fit()时特征的数量
print('特征数量：\n', clf.n_features_)
# 包含每个特征重要性的列表，值越大表示该特征越重要
print('每个特征的重要性：\n', clf.feature_importances_)

x = [[1]*6]
print('预测结果：\n', clf.predict(x))
```

8.6.4 使用决策树算法判断学员的Python水平

```
from sklearn import tree
import numpy as np

questions = ('《Python程序设计基础（第2版）》',
             '《Python程序设计基础与应用》',
             '《Python程序设计（第2版）》',
             '《大数据的Python基础》',
             '《Python程序设计开发宝典》',
             '《Python可以这样学》',
             '《中学生可以这样学Python》',
             '《Python编程基础与案例集锦（中学版）》',
             '《玩转Python轻松过二级》',
             '微信公众号“Python小屋”的免费资料',)
```

8.6.4 使用决策树算法判断学员的Python水平

```
# 每个样本的数据含义：
# 0没看过，1很多看不懂，2大部分可以看懂，3没压力
answers = [[3, 3, 3, 3, 3, 3, 3, 3, 3, 3],
           [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
           [1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
           [0, 0, 0, 0, 0, 0, 2, 2, 0, 1],
           [0, 0, 0, 0, 3, 3, 0, 0, 0, 3],

           [3, 3, 0, 3, 0, 0, 0, 0, 3, 1],
           [3, 0, 3, 0, 3, 0, 0, 3, 3, 2],
           [0, 0, 3, 0, 3, 3, 0, 0, 0, 3],
           [2, 2, 0, 2, 0, 0, 0, 0, 0, 1],
           [0, 2, 1, 3, 1, 1, 0, 0, 2, 1]
          ]
```

8.6.4 使用决策树算法判断学员的Python水平

```
labels = ['超级高手', '门外汉', '初级选手', '初级选手', '高级选手',
          '中级选手', '高级选手', '超级高手', '初级选手', '初级选手']

clf = tree.DecisionTreeClassifier().fit(answers, labels) # 训练

yourAnswer = []
# 显示调查问卷，并接收用户输入
for question in questions:
    print('==========\n你看过董付国老师的', question, '吗？')
    # 确保输入有效
    while True:
        print('没看过输入0，很多看不懂输入1，'
              '大部分可以看懂输入2，没压力输入3')
        try:
            answer = int(input('请输入：'))
            assert 0<=answer<=3
            break
        except:
            print('输入无效，请重新输入。')
            pass
    yourAnswer.append(answer)

print(clf.predict(np.array(yourAnswer).reshape(1,-1)))   # 分类
```

8.7 支持向量机算法原理与应用-8.7.1 支持向量机算法基本原理与sklearn实现

在学习支持向量机算法之前，先看一个脑筋急转弯，在图 8-7 中画一条直线对多边形进行分隔得到两个三角形，应该怎样画这条直线呢？

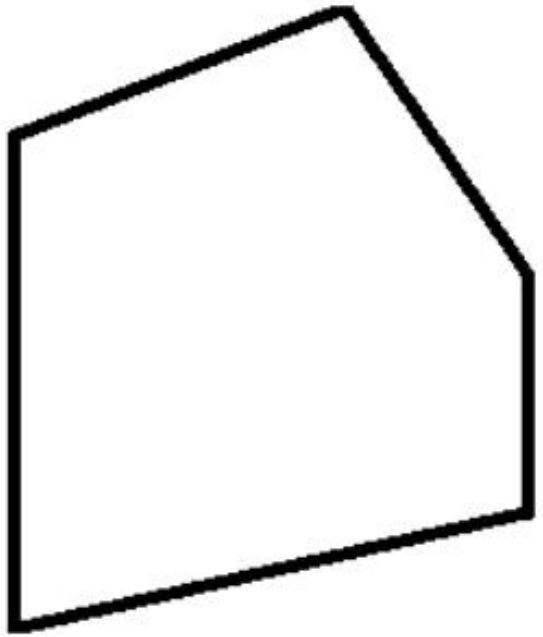


图 8-7　多边形

8.7.1 支持向量机算法基本原理与sklearn实现

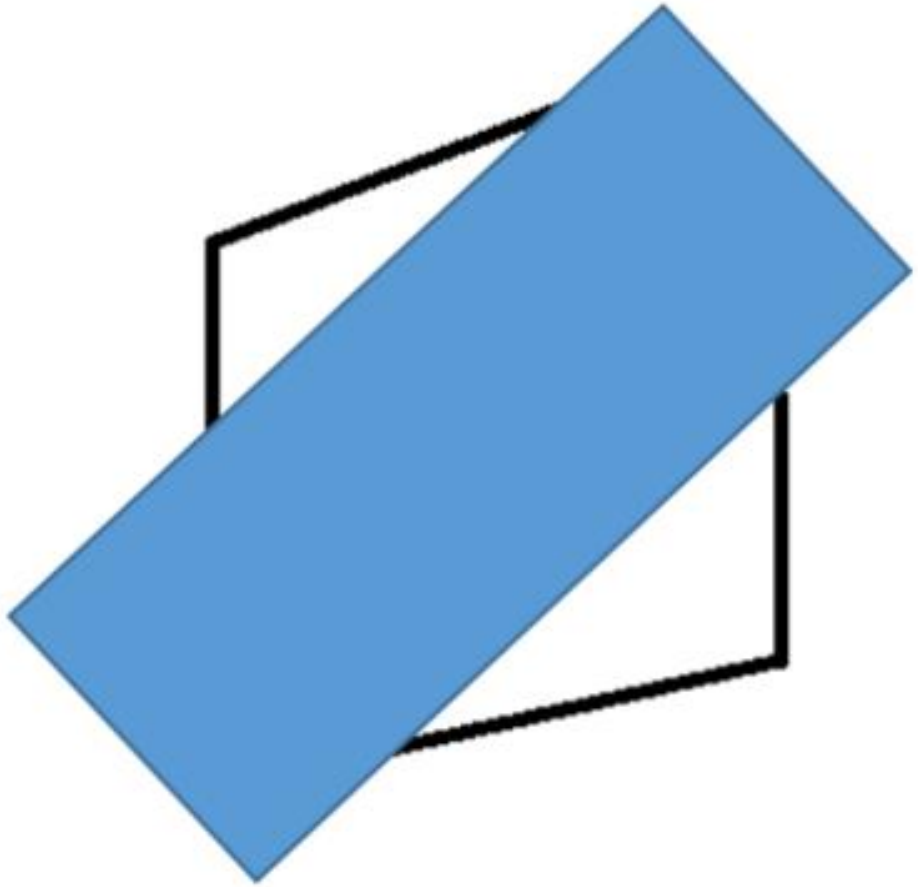


图 8-8　画一条直线把分隔多边形得到两个三角形

8.7.1 支持向量机算法基本原理与sklearn实现

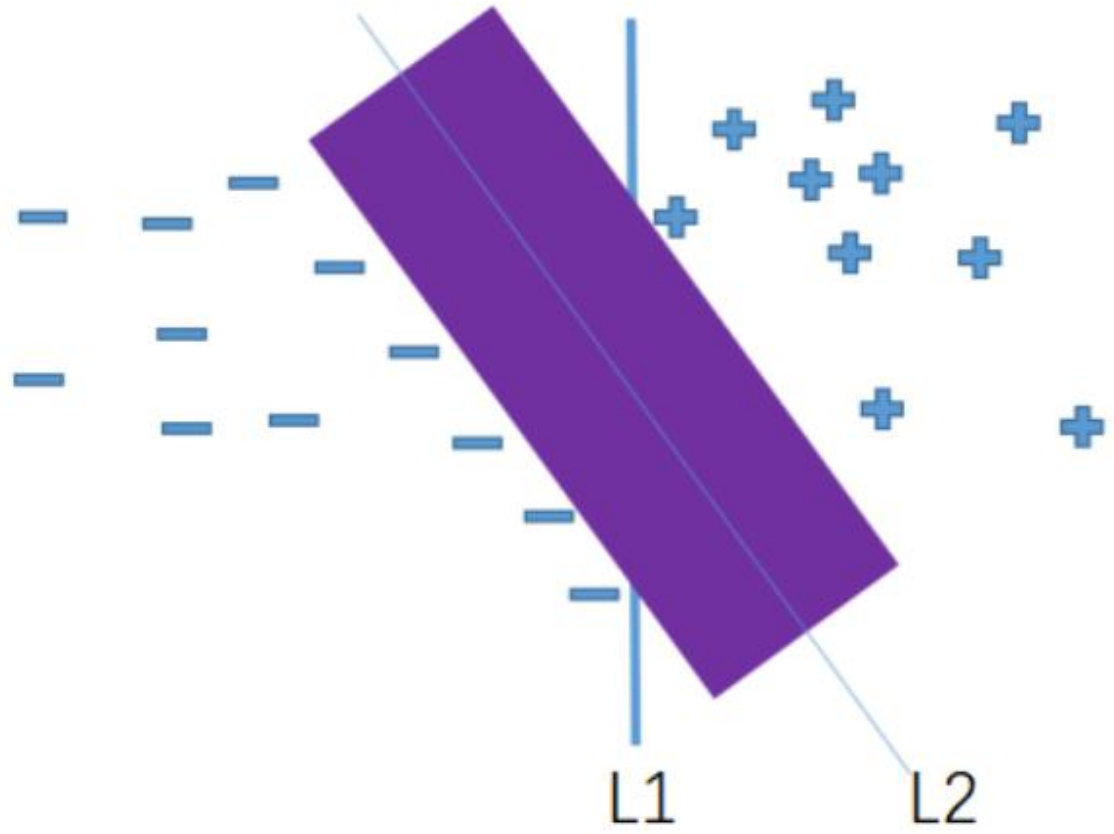


图 8-9　分隔平面上随意摆放的两类物体

8.7.1 支持向量机算法基本原理与sklearn实现

- 支持向量机（Support Vector Machine, SVM）是通过寻找超平面对样本进行分隔从而实现分类或预测的算法，分隔样本时的原则是使得间隔最大化，寻找间隔最大的支持向量。在二维平面上相当于寻找一条“最粗的直线”把不同类别的物体分隔开，或者说寻找两条平行直线对物体进行分隔并使得这两条平行直线之间的距离最大。如果样本在二维平面上不是线性可分的，无法使用一条简单的直线将其完美分隔开，就尝试着通过某种变换把所有样本都投射到三维空间（例如，把一类物体沿z轴正方向移动靠近用户，另一类物体沿z轴负方向移动远离用户）然后使用一个平面（例如，屏幕所在的平面）进行分隔。如果样本在三维空间仍不是线性可分的，就尝试着投射到更高维空间使用超平面进行分隔，以此类推。

8.7.1 支持向量机算法基本原理与sklearn实现

- 如果样本在原来维度的空间中不是线性可分的，就投影到更高维的特征空间进行处理，支持向量机的核决定了如何投影到更高维空间，这也是支持向量机的关键所在。常用的核有线性核、多项式核（Polynomial kernel）、径向基函数核（Radial Basis Function kernel）、拉普拉斯核和Sigmoid核。支持向量机在人脸识别、文本分类、图像分类、手写识别、生物序列分析等模式识别应用中取得了较大成功。
- 在支持向量机算法中核函数和正则化参数的选择非常重要。另外，核函数的参数决定了边界的形状，对模型也有较大影响。例如，RBF核的gamma参数用来调节内核宽度，gamma值越小，RBF核的直径越大，模型越简单，但容易出现欠拟合；gamma值越大，模型越复杂，容易出现过拟合问题；C越小表示单个数据点对模型影响越小，模型越简单。在多项式核的SVC中起决定作用的则是degree和正则化参数C。

8.7.1 支持向量机算法基本原理与sklearn实现

参数名称	含义
C	用来设置错误项的惩罚参数C，值越大对误分类的惩罚越大，间隔越小，对错误的容忍度越低
kernel	用来指定算法中使用的核函数类型，可用的值有'linear'、'poly'、'rbf'、'sigmoid'、'precomputed'或可调用对象。如果样本在原始空间中就是线形可分的，可以直接使用kernel='linear'，如果样本在原始空间中不是线性可分的，再根据实际情况选择使用其他的核
degree	用来设置kernel='poly'时多项式核函数的度，kernel为其他值时忽略degree参数
gamma	用来设置kernel值为'rbf'、'poly'和'sigmoid'时的核系数，gamma='auto'使用1/n_features作为系数
coef0	用来设置核函数中的独立项，仅适用于kernel为'poly'和'sigmoid'的场合
probability	用来设置是否启用概率估计，必须在调用fit()方法之前启用，启用之后会降低fit()方法的执行速度
shrinking	用来设置是否使用启发式收缩方式
tol	用来设置停止训练的误差精度
max_iter	用来设置最大迭代次数，-1表示不限制
decision_function_shape	用来设置决策函数的形状，可以用的值有'ovr'或'ovo'，前者表示one-vs-rest，决策函数形状为(n_samples, n_classes)；后者表示one-vs-one，决策函数形状为(n_samples, n_classes * (n_classes - 1) / 2)

8.7.1 支持向量机算法基本原理与sklearn实现

属性	含义
support_	支持向量的索引
support_vectors_	支持向量
n_support_	每个类的支持向量的数量
dual_coef_	决策函数中支持向量的系数
coef_	为特征设置的权重，仅适用于线性核

8.7.1 支持向量机算法基本原理与sklearn实现

方法	功能
decision_function(self, X)	计算样本集X到分隔超平面的函数距离
predict(self, X)	对X中的样本进行分类
predict_proba(self, X)	返回X中样本属于不同类的概率
fit(self, X, y, sample_weight=None)	根据训练集对支持向量机分类器进行训练
score(self, X, y, sample_weight=None)	根据给定的测试集和标签计算并返回分类准确度平均值

8.7.2 使用支持向量机对手写数字图像进行分类

```
from os import mkdir, listdir
from os.path import isdir, basename
from random import choice, randrange
from string import digits
from PIL import Image, ImageDraw
from PIL.ImageFont import truetype
from sklearn import svm
from sklearn.model_selection import train_test_split
```

8.7.2 使用支持向量机对手写数字图像进行分类

```
# 图像尺寸、图片中的数字字体大小、噪点比例
width, height = 30, 60
fontSize = 40
noiseRate = 8

def generateDigits(dstDir='datasets', num=40000):
    # 生成num个包含数字的图片文件存放于当前目录下的datasets子目录
    if not isdir(dstDir):
        mkdir(dstDir)
    # digits.txt用来存储每个图片对应的数字
    with open(dstDir+'\\digits.txt', 'w') as fp:
        for i in range(num):
            # 随机选择一个数字，生成对应的彩色图像文件
            digit = choice(digits)
            im = Image.new('RGB', (width,height), (255,255,255))
            imDraw = ImageDraw.Draw(im)
            font = truetype('c:\\windows\\fonts\\TIMESBD.TTF', fontSize)
            # 写入黑色数字
            imDraw.text((0,0), digit, font=font, fill=(0,0,0))
            # 加入随机干扰
            for j in range(int(noiseRate*width*height)):
                w, h = randrange(1, width-1), randrange(height)
                # 水平交换两个相邻像素的颜色
                c1 = im.getpixel((w,h))
                c2 = im.getpixel((w+1,h))
                imDraw.point((w,h), fill=c2)
                imDraw.point((w+1,h), fill=c1)
            im.save(dstDir+'\\'+str(i)+'.jpg')
            fp.write(digit+'\n')
```

8.7.2 使用支持向量机对手写数字图像进行分类

```
def loadDigits(dstDir='datasets'):
    # 获取所有图像文件名
    digitsFile = [dstDir+'\\'+fn for fn in listdir(dstDir) if fn.endswith('.jpg')]
    # 按编号排序
    digitsFile.sort(key=lambda fn: int(basename(fn)[:-4]))
    # digitsData用于存放读取的图片中数字信息
    # 每个图片中所有像素值存放于digitsData中的一行数据
    digitsData = []
    for fn in digitsFile:
        with Image.open(fn) as im:
            # getpixel()方法用来读取指定位置像素的颜色值
            data = [sum(im.getpixel((w,h)))/len(im.getpixel((w,h)))
                    for w in range(width)
                    for h in range(height)]
            digitsData.append(data)
    # digitsLabel用于存放图片中数字的标准分类
    with open(dstDir+'\\digits.txt') as fp:
        digitsLabel = fp.readlines()
    # 删除数字字符两侧的空白字符
    digitsLabel = [label.strip() for label in digitsLabel]
    return (digitsData, digitsLabel)
```

8.7.2 使用支持向量机对手写数字图像进行分类

```
# 生成图片文件
generateDigits(num=8000)
# 加载数据
data = loadDigits()
print('数据加载完成。')

# 随机划分训练集和测试集，其中参数test_size用来指定测试集大小
X_train, X_test, y_train, y_test = train_test_split(data[0], data[1], test_size=0.1)
# 创建并训练模型
svcClassifier = svm.SVC(kernel="linear", C=1000, gamma=0.001)
svcClassifier.fit(X_train, y_train)
print('模型训练完成。')

# 使用测试集对模型进行评分
score = svcClassifier.score(X_test, y_test)
print('模型测试得分：', score)
```

8.8 KNN算法原理与应用-8.8.1 KNN算法基本原理与sklearn实现

- KNN算法是k-Nearest Neighbor的简称，叫作k近邻算法，属于有监督学习算法，既可以用于分类，也可以用于回归，本节重点介绍分类的用法。
- k近邻分类算法的基本思路是在样本空间中查找k个最相似或者距离最近的样本，然后根据k个最相似的样本对未知样本进行分类。

8.8.1 KNN算法基本原理与sklearn实现

- 使用KNN算法进行分类的基本步骤为：

1)对数据进行预处理，提取特征向量，对原始数据进行重新表达；

2)确定距离计算公式，并计算已知样本空间中所有样本与未知样本的距离；

3)对所有距离按升序排列；

4)确定并选取与未知样本距离最小的k个样本；

5)统计选取的k个样本中每个样本所属类别的出现频率；

6)把出现频率最高的类别作为预测结果，认为未知样本属于这个类别。

8.8.1 KNN算法基本原理与sklearn实现

```
>>> from sklearn.neighbors import KNeighborsClassifier
>>> X = [[1,5], [2,4], [2.2,5],
         [4,1.5], [5,1], [5,2], [5,3], [6,2],
         [7.5,4.5], [8.5,4], [7.9,5.1], [8.2,5]]
>>> y = [0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2]
>>> knn = KNeighborsClassifier(n_neighbors=3)  # 创建模型，k=3
>>> knn.fit(X, y)                              # 训练模型
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=3, p=2, weights='uniform')
>>> knn.predict([[4.8,5.1]])                   # 分类
array([0])
>>> knn = KNeighborsClassifier(n_neighbors=9)  # 设置参数k=9
>>> knn.fit(X, y)
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=9, p=2, weights='uniform')
>>> knn.predict([[4.8,5.1]])                   # 分类
array([1])
>>> knn.predict_proba([[4.8,5.1]])             # 属于不同类别的概率
array([[ 0.22222222,  0.44444444,  0.33333333]])
```

8.8.2 使用KNN算法判断交通工具类型

- 源码见配套资源。

8.9 KMeans聚类算法原理与应用-8.9.1 KMeans聚类算法基本原理与sklearn实现

- KMeans（K-均值聚类）是一种比较简单但广泛使用的聚类算法，属于无监督学习算法，在数据预处理时使用较多。在初始状态下，样本没有都标签或目标值，由聚类算法发现样本之间的关系，然后自动把在某种意义下相似的样本归为一类并贴上相应的标签。
- KMeans算法的基本思想是：选择样本空间中k个样本（点）为初始中心，然后对剩余样本进行聚类，每个中心把距离自己最近的样本“吸引”过来，然后更新聚类中心的值，依次把每个样本归到距离最近的类中，重复上面的过程，直至得到某种条件下最好的聚类结果。

8.9.1 KMeans聚类算法基本原理与sklearn实现

- 假设要把样本集分为k个类别，算法描述如下：

（1）按照定义好的规则选择k个样本作为每个类的初始中心；

（2）在第i次迭代中，对任意一个样本，计算该样本到k个类中心的距离，将该样本归到距离最小的中心所在的类；

（3）利用均值或其他方法更新该类的中心值；

（4）重复上面的过程，直到没有样本被重新分配到不同的类或者没有聚类中心再发生变化，停止迭代。

8.9.1 KMeans聚类算法基本原理与sklearn实现

```
from numpy import array
from random import randrange
from sklearn.cluster import KMeans

# 原始数据
X = array([[1,1,1,1,1,1,1], [2,3,2,2,2,2,2], [3,2,3,3,3,3,3],
           [1,2,1,2,2,1,2], [2,1,3,3,3,2,1], [6,2,30,3,33,2,71]])
# 训练模型，选择3个样本作为中心，把所有样本划分为3个类
kmeansPredicter = KMeans(n_clusters=3).fit(X)
print('原始数据：\n', X)
# 原始数据每个样本所属的类别标签
category = kmeansPredicter.labels_
print('聚类结果：', category)
print('='*30)
print('聚类中心：\n', kmeansPredicter.cluster_centers_)
print('='*30)

def predict(element):
    result = kmeansPredicter.predict(element)
    print('预测结果：', result)
    print('相似元素：\n', X[category==result])

# 测试
predict([[1,2,3,3,1,3,1]])
print('='*30)
predict([[5,2,23,2,21,5,51]])
```

8.9.2 使用KMeans算法压缩图像颜色

- 源码见配套资源。

8.10 分层聚类算法原理与应用

- 分层聚类又称系统聚类算法或系谱聚类，该算法首先把所有样本看作各自一类，定义类间距离计算方式，选择距离最小的一对元素合并成一个新的类，重新计算各类之间的距离并重复上面的步骤，直到将所有原始元素划分为指定数量的类。
- 该算法的计算复杂度非常高，不适合大数据聚类问题。

8.10 分层聚类算法原理与应用

- 扩展库sklearn.cluster中提供了分层聚类算法AgglomerativeClustering，该类构造方法的语法为：

```
__init__(self, n_clusters=2, affinity='euclidean', memory=None,
connectivity=None, compute_full_tree='auto', linkage='ward',
pooling_func=<function mean at 0x0000028EBB2F7EA0>)
```

8.10 分层聚类算法原理与应用

参数名称	含义
n_clusters	最终要确定的聚类的数量
affinity	用来设置距离计算方法，可以为"euclidean"、"l1"、"l2"、"manhattan"、"cosine"或"precomputed"，当参数linkage="ward"时，affinity的值只能为"euclidean"
connectivity	用来设置连通矩阵，定义样本之间的连接关系，可以设置为连通矩阵或者能把数据转换为连通矩阵的可调用对象
compute_full_tree	用来设置是否构建完整的层次树
linkage	用来设置使用哪种连通标准，连通标准用来定义集合之间使用哪种距离，该算法将合并使得该标准最小的两个类，可用的值有： • "ward"：使得要合并的聚类的方差最小 • "complete"：使用两个集合中所有观察点的最大距离 • "average"：使用两个集合中每个观察点之间距离的平均值

8.10 分层聚类算法原理与应用

属性	含义
labels_	每个样本的聚类标签
n_leaves_	层次树中叶子的数量
n_components_	图中连接部分的数量估计值
children_	每个非叶子节点的孩子节点，(n_nodes-1,2)形式的数组

8.10 分层聚类算法原理与应用

方法	功能
fit(self, X, y=None)	对数据进行拟合
get_params(self, deep=True)	返回估计器的参数
set_params(self, **params)	设置估计器的参数
fit_predict(self, X, y=None)	对数据进行聚类并返回聚类后的标签

8.10 分层聚类算法原理与应用

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import AgglomerativeClustering

def AgglomerativeTest(n_clusters):
    assert 1 <= n_clusters <= 4
    predictResult = AgglomerativeClustering(n_clusters=n_clusters, affinity='euclidean',
                                            linkage='ward').fit_predict(data)
    # 定义绘制散点图时使用的颜色和散点符号
    colors = 'rgby'
    markers = 'o*v+'
    # 依次使用不同的颜色和符号绘制每个类的散点图
    for i in range(n_clusters):
        subData = data[predictResult==i]
        plt.scatter(subData[:,0], subData[:,1], c=colors[i], marker=markers[i], s=40)
    plt.show()


# 生成随机数据，200个点，分成4类，返回样本及标签
data, labels = make_blobs(n_samples=200, centers=4)
print(data)
AgglomerativeTest(3)
AgglomerativeTest(4)
```

8.11 DBSCAN算法原理与应用

- DBSCAN（Density-Based Spatial Clustering of Applications with Noise）属于密度聚类算法，把类定义为密度相连对象的最大集合，通过在样本空间中不断搜索高密度的核心样本并扩展得到最大集合完成聚类，能够在带有噪点的样本空间中发现任意形状的聚类并排除噪点。

8.11 DBSCAN算法原理与应用

- 基本概念：

✓核心样本：如果给定样本的邻域（最大距离为eps）内样本数量超过阈值min_samples，则称为核心样本。

✓边界样本：在eps邻域内样本的数量小于min_samples，但是落在核心样本的邻域内的样本。

✓噪声样本：既不是核心样本也不是边界样本的样本。

✓直接密度可达：如果样本q在核心样本p的eps邻域内，则称q从p出发是直接密度可达的。

✓密度可达：集合中的样本链p1、p2、p3、...、pn，如果每个样本pi+1从pi出发都是直接密度可达的，则称pn从p1出发是密度可达的。

✓密度相连：集合中如果存在样本o使得样本p和q从o出发都是密度可达的，则称样本p和q是互相密度相连的。

8.11 DBSCAN算法原理与应用

- DBSCAN聚类算法的工作过程如下：

1）定义邻域半径eps和样本数量阈值min_samples。

2）从样本空间中抽取一个尚未访问过的样本p。

3）如果样本p是核心样本，进入第4）步；否则根据实际情况将其标记为噪声样本或某个类的边界样本，返回第2）步。

4）找出样本p出发的所有密度相连样本，构成一个聚类Cp（该聚类的边界样本都是非核心样本），并标记这些样本为已访问。

5）如果全部样本都已访问，算法结束；否则返回第2）步。

8.11 DBSCAN算法原理与应用

- 扩展库sklearn.cluster实现了DBSCAN聚类算法，其构造方法语法为：

```
__init__(self, eps=0.5, min_samples=5, metric='euclidean',
metric_params=None, algorithm='auto', leaf_size=30, p=None, n_jobs=1)
```

8.11 DBSCAN算法原理与应用

参数名称	含义
eps	用来设置邻域内样本之间的最大距离，如果两个样本之间的距离小于eps，则认为属于同一个领域。参数eps的值越大，聚类覆盖的样本越多
min_samples	用来设置核心样本的邻域内样本数量的阈值，如果一个样本的eps邻域内样本数量超过min_samples，则认为该样本为核心样本。参数min_samples的值越大，核心样本越少，噪声越多
metric	用来设置样本之间距离的计算方式
algorithm	用来计算样本之间距离和寻找最近样本的算法，可用的值有'auto'、'ball_tree'、'kd_tree'或'brute'
leaf_size	传递给BallTree或cKDTree的叶子大小，会影响树的构造和查询速度以及占用内存的大小
p	用来设置使用闵科夫斯基距离公式计算样本距离时的幂

8.11 DBSCAN算法原理与应用

属性	含义
core_sample_indices_	核心样本的索引
components_	通过训练得到的每个核心样本的副本
labels_	数据集中每个点的聚类标签，其中-1表示噪声样本

8.11 DBSCAN算法原理与应用

方法	功能
fit(self, X, y=None, sample_weight=None)	对数据进行拟合，如果构造DBSCAN聚类器时设置了metric='precomputed'则要求参数X为样本之间的距离数组
fit_predict(self, X, y=None, sample_weight=None	对X进行聚类并返回聚类标签

8.11 DBSCAN算法原理与应用

参考代码见配套资源

8.12 使用协同过滤算法进行电影推荐

- 协同过滤算法常用于商品推荐或者类似的场合，根据用户之间或商品之间的相似性进行精准推荐，可以分为基于用户的协同过滤算法和基于商品的协同过滤算法。
- 以电影推荐为例，假设用户1喜欢看电影A、B、C、D、G，用户2喜欢看电影A、D、E、F，用户3喜欢看电影A、B、D，现在用户3想再看个没看过的电影，向用户1和用户2寻求推荐。简单分析易知，与用户2相比，用户1和用户3更相似，所以根据用户1喜欢的电影进行推荐，也就是用户1看过但用户3还没看过的电影C或G。
- 常用来计算相似性的方法有欧几里德距离、余弦距离和杰卡德相似度。在上面的例子中采用的相似性度量标准是杰卡德公式，也就是两个集合交集中元素数量与它们的并集中元素数量的比值，这个比值越大表示两个集合的相似度越高。用户3和用户1的交集为{A, B, D}，并集为{A, B, C, D, G}，相似度为3/5=0.6；用户3和用户2的交集为{A, D}，并集为{A,B,D,E,F}，相似度为2/5=0.4。

8.12 使用协同过滤算法进行电影推荐

```
from random import randrange

# 模拟历史电影打分数据，共10个用户，每个用户打分的电影数量不等
data = {'user'+str(i):{'film'+str(randrange(1, 15)):randrange(1, 6) for j in range(randrange(3, 10))}
          for i in range(10)}
# 寻求推荐的用户对电影打分的数据
user = {'film'+str(randrange(1, 15)):randrange(1,6) for i in range(5)}

# 最相似的用户及其对电影打分情况
# 两个最相似的用户共同打分的电影最多，同时所有电影打分差值的平方和最小
rule = lambda item:(-len(item[1].keys()&user),
                    sum(((item[1].get(film)-user.get(film))**2 for film in user.keys()&item[1].keys())))
similarUser, films = min(data.items(), key=rule)
# 输出信息以便验证，每行数据有3列
# 分别为该用户与当前用户共同打分的电影数量、打分差的平方和、该用户打分数据
print('known data'.center(50, '='))
for item in data.items():
    print(len(item[1].keys()&user.keys()),
          sum(((item[1].get(film)-user.get(film))**2 for film in user.keys()&item[1].keys())),
          item, sep=':')
print('current user'.center(50, '='), user, sep='\n')
print('most similar user and his films'.center(50, '='))
print(similarUser, films, sep=':')
print('recommended film'.center(50, '='))
# 在当前用户没看过的电影中选择打分最高的进行推荐
print(max(films.keys()-user.keys(), key=lambda film: films[film]))
```

8.13 关联规则分析原理与应用-8.13.1 关联规则分析原理与基本概念

- 关联规则分析或者关联规则学习主要用于从大规模数据中寻找物品之间隐含的或者可能存在的联系，从而实现某种意义上的预测。
- 例如，捡到鼠标垫的幸运者3个月内是否有可能购买笔记本电脑；正在浏览某商品页面的用户还可能对什么商品感兴趣；一个特别爱吃炒花生的人喜欢喝酒的可能性有多大；在饭店吃饭时点了糖醋里脊和红烧茄子的客人再点红烧排骨的可能性有多大；通过调整商场内商品的柜台分布来方便顾客同时提高整体营业额。

8.13.1 关联规则分析原理与基本概念

- 常用概念：

1）项集：包含若干物品或条目的集合。包含k的物品的集合称作k-项集。

2）频繁项集：经常一起出现的物品的集合。如果某个项集是频繁的，那么它的所有子集都是频繁的；如果某个项集不是频繁的，那么它的所有超集都不是频繁的。这一点是避免项集数量过多的重要基础，使得快速计算频繁项集成为可能。

3）关联规则：可以表示为一个蕴含式R:X==>Y，其中X&Y为空集。这样一条关联规则的含义是，如果X发生，那么Y很可能也会发生。

4）支持度：一个项集X的支持度是指包含该项集的记录数量在整个数据集中所占的比例，也就是项集X的概率P(X)。

5）置信度：用来表示某条规则可信度的大小，用来检验一个推测是否靠谱。对于某条关联规则X==>Y，置信度是指同时包含X和Y的项集X|Y（表示X和Y的并集）的支持度与项集X的支持度的比值P(X|Y)/P(X)。如果某条关联规则不满足最小置信度要求，那么该规则的所有子集也不会满足最小置信度。根据这一点可以减少要测试的规则数量。

6）强关联规则：同时满足最小支持度和最小置信度的关联规则。根据不同的支持度和置信度阈值设置，关联规则分析的结果会有所不同。

8.13.1 关联规则分析原理与基本概念

- 在使用时，关联规则分析主要分两步来完成。第一步是查找满足最小支持度要求的所有频繁项集，首先得到所有1-频繁项集（包含1个元素的集合），然后根据这些1-频繁项集生成2-频繁项集，再根据2-频繁项集生成3-频繁项集，依次类推。为减少无效搜索和空间占用，每次迭代时可以对k-项集进行过滤，如果一个k-项集是频繁项集的话，它的所有k-1子集都应该是频繁项集。反之，如果一个k-项集有不是频繁项集的k-1子集，就删掉这个项集。第二步是在频繁项集中查找满足最小置信度的所有关联规则。

8.13.2 使用关联规则分析演员关系

- 源码见配套资源。

	A	B	C
1	电影名称	导演	演员
2	电影1	导演1	演员1，演员2，演员3，演员4
3	电影2	导演2	演员3，演员2，演员4，演员5
4	电影3	导演3	演员1，演员5，演员3，演员6
5	电影4	导演1	演员1，演员4，演员3，演员7
6	电影5	导演2	演员1，演员2，演员3，演员8
7	电影6	导演3	演员5，演员7，演员3，演员9
8	电影7	导演4	演员1，演员4，演员6，演员7
9	电影8	导演1	演员1，演员4，演员3，演员8
10	电影9	导演2	演员5，演员4，演员3，演员9
11	电影10	导演3	演员1，演员4，演员5，演员10
12	电影11	导演1	演员1，演员4，演员3，演员11
13	电影12	导演2	演员7，演员4，演员9，演员12
14	电影13	导演3	演员1，演员7，演员3，演员13
15	电影14	导演4	演员10，演员4，演员9，演员14
16	电影15	导演5	演员1，演员8，演员11，演员15
17	电影16	导演6	演员14，演员4，演员13，演员16
18	电影17	导演7	演员3，演员4，演员9
19	电影18	导演8	演员3，演员4，演员10

8.14 数据降维

- 数据降维是指采取某种映射方法，把高维空间中可能包含冗余信息或噪声的数据点映射到低维空间中，在低维空间中重新表示高维空间中的数据，从而可以挖掘数据内部本质结构特征、提高识别精度以及减少计算量和空间复杂度。从广义上看，前面几节介绍的各种聚类算法也可以看作是数据降维技术。
- 主成分分析（Principal Component Analysis, PCA）是一种比较常用的线性降维方法，该方法通过对矩阵进行奇异值分解（参考本书6.7节）把高维空间中的数据映射到低维空间中重新表示，并期望在投影后的维度上方差最大，使得投影后的维度尽可能少，同时又保留尽可能多的原数据特征。除了PCA，其他常用的降维算法还有线性判别分析（Linear Discriminant Analysis，LDA）、局部线性嵌入（Locally Linear Embedding，LLE）和拉普拉斯特征映射。

8.14 数据降维

- 扩展库sklearn在decomposition模块中提供了主成分分析PCA的实现，该类构造方法语法为：

```
__init__(self, n_components=None, copy=True, whiten=False,
svd_solver='auto', tol=0.0, iterated_power='auto', random_state=None)
```

8.14 数据降维

参数名称	含义
n_components	用来设置要保留的成分的数量
whiten	用来设置是否白化。设置为True时，components_向量乘以n_samples的平方根然后再除以奇异值，降低冗余，降低样本特征之间的相关性，使得各特征具有相同的方差（此时协方差矩阵为单位矩阵）
svd_solver	用来设置奇异值分解的方法，可用的值有'auto'、'full'、'arpack'、'randomized'
iterated_power	用来设置svd_solver='randomized'时SVD算法的迭代次数

8.14 数据降维

属性	含义
components_	(n_components, n_features)形状的数组，特征空间中的主成分，表示数据中最大方差的方向
explained_variance_	(n_components,)形状的数组，降维后各主成分的方差值
explained_variance_ratio_	降维后各主成分的方差在总方差中的百分比
singular_values_	(n_components,)形状的数组，降维后各主成分的奇异值
mean_	(n_features,)形状的数组，每个特征的经验平均值，等价于X.mean(axis=1)
n_components_	主成分的数量
noise_variance_	噪声方差

8.14 数据降维

方法	功能
fit(self, X, y=None)	使用X训练模型
fit_transform(self, X, y=None)	使用X拟合模型并对X进行降维
transform(self, X)	对X进行降维

8.14 数据降维

```
>>> import numpy as np
>>> from sklearn.decomposition import PCA
>>> X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
>>> pca = PCA(n_components=2)
>>> pca.fit(X)
PCA(copy=True, iterated_power='auto', n_components=2,
    random_state=None, svd_solver='auto', tol=0.0, whiten=False)
>>> pca.components_
array([[-0.83849224, -0.54491354],
       [ 0.54491354, -0.83849224]])
>>> pca.explained_variance_
array([ 7.93954312,  0.06045688])
>>> pca.explained_variance_ratio_
array([ 0.99244289,  0.00755711])
>>> pca.singular_values_
array([ 6.30061232,  0.54980396])
>>> pca.transform(X)
array([[ 1.38340578,  0.2935787 ],
       [ 2.22189802, -0.25133484],
       [ 3.6053038 ,  0.04224385],
       [-1.38340578, -0.2935787 ],
       [-2.22189802,  0.25133484],
       [-3.6053038 , -0.04224385]])
```

8.14 数据降维

```
>>> pca = PCA(n_components=2, whiten=True)      # 白化
>>> pca.fit(X)
PCA(copy=True, iterated_power='auto', n_components=2, random_state=None,
  svd_solver='auto', tol=0.0, whiten=True)
>>> pca.transform(X)
array([[ 0.49096647,  1.19399271],
       [ 0.78854479, -1.02218579],
       [ 1.27951125,  0.17180692],
       [-0.49096647, -1.19399271],
       [-0.78854479,  1.02218579],
       [-1.27951125, -0.17180692]])
```

8.14 数据降维

```
>>> pca = PCA(n_components=1, svd_solver='arpack')
>>> pca.fit(X)
PCA(copy=True, iterated_power='auto', n_components=1, random_state=None,
  svd_solver='arpack', tol=0.0, whiten=False)
>>> pca.singular_values_
array([ 6.30061232])
>>> pca.components_
array([[-0.83849224, -0.54491354]])
>>> pca.transform(X)
array([[ 1.38340578],
       [ 2.22189802],
       [ 3.6053038 ],
       [-1.38340578],
       [-2.22189802],
       [-3.6053038 ]])
```

8.15 交叉验证与网格搜索-8.15.1 使用交叉验证评估模型泛化能力

- 交叉验证（Cross Validation）会反复对数据集进行划分，并使用不同的划分对模型进行评分，可以更好地评估模型的泛化质量。

8.15.1 使用交叉验证评估模型泛化能力

- 扩展库sklearn在model_selection模块中提供了用来实现交叉验证的函数cross_val_score()，其语法为：

```
cross_val_score(estimator, X, y=None, groups=None, scoring=None, cv=None,
n_jobs=1, verbose=0, fit_params=None, pre_dispatch='2*n_jobs')
```

其中，1）参数estimator用来指定要评估的模型；2）参数X和y分别用来指定数据集及其对应的标签；3）参数cv用来指定划分策略，常设置为整数表示把数据集拆分成几个部分对模型进行训练和评分。该函数返回实数数组，数组中每个实数分别表示每次评分的结果，在实际使用时往往使用这些得分的平均值作为最终结果。

8.15.1 使用交叉验证评估模型泛化能力

- 函数cross_val_score()使用k折叠（k-folds）交叉验证，把数据集拆分为k个部分，然后使用k个数据集对模型进行训练和评分。另外，sklearn.model_selection模块中还提供了随机拆分交叉验证ShuffleSplit和逐个测试交叉验证LeaveOneOut。
- 源码见配套资源。

8.15.2 使用网格搜索确定模型最佳参数

- 选定了合适的模型之后，参数的设置也非常重要，不同的参数对模型泛化的性能也有所影响。可以编写代码循环测试可能的参数取值，对于每一组参数使用交叉验证对模型进行评分，然后从中选择最佳的参数，也可以使用扩展库sklearn提供的网格搜索GridSearchCV来完成这个功能，使用网格搜索更加简洁方便一些。扩展库sklearnd的model_selection模块中GridSearchCV类的构造方法语法为：

```
__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=True)
```

8.15.2 使用网格搜索确定模型最佳参数

参数名称	含义
estimator	用来设置待选择参数的评估器
param_grid	用来设置待测试和选择的参数
scoring	用来设置选择参数时使用的评分函数
cv	用来设置交叉验证划分策略，取值可以为： • None：默认使用3-折叠交叉验证 • 整数：指定折叠的数量 • 可以作为交叉验证生成器使用的可调用对象
refit	用来设置是否使用在整个数据集上发现的最佳参数对模型进行重新拟合

8.15.2 使用网格搜索确定模型最佳参数

属性	含义
cv_results_	交叉验证结果
best_estimator_	得分最高的评估器，仅refit=True时可用
best_score_	最佳评估器的交叉验证平均得分
best_params_	最佳参数
scorer_	使用的评分函数
n_splits_	交叉验证时折叠或迭代的数量

8.15.2 使用网格搜索确定模型最佳参数

方法	功能
fit(self, X, y=None, groups=None, **fit_params)	使用所有参数对模型进行拟合
predict(self, X)	使用发现的最佳参数调用模型的predict()方法
score(self, X, y=None)	如果评估器进行了重新拟合，返回评估器在给定数据上的得分
transform(self, X)	使用发现的最佳参数调用模型的transform()方法

8.15.2 使用网格搜索确定模型最佳参数

- 源码见配套资源。

补充：数据预处理

- 二值化：以指定阈值为界，把阈值两侧的值分别使用固定值0、1表示。可以使用训练集进行fit()，然后对待测数据进行transform()。

```
>>> from sklearn import preprocessing
>>> import numpy as np
>>> data = np.random.randint(1,100,(3,8))
>>> data
array([[92, 21, 81,  7, 88, 54, 68, 89],
       [36, 35, 29, 83, 13, 30, 13,  4],
       [87, 11, 93, 50, 56, 50, 36, 17]])
>>> preprocessing.Binarizer(threshold=50).fit_transform(data)
array([[1, 0, 1, 0, 1, 1, 1, 1],
       [0, 0, 0, 1, 0, 0, 0, 0],
       [1, 0, 1, 0, 1, 0, 0, 0]])
>>> preprocessing.Binarizer(threshold=50).fit(data).transform(data)
array([[1, 0, 1, 0, 1, 1, 1, 1],
       [0, 0, 0, 1, 0, 0, 0, 0],
       [1, 0, 1, 0, 1, 0, 0, 0]])
```

```
>>> preprocessing.binarize(data, threshold=50)
array([[1, 0, 1, 0, 1, 1, 1, 1],
       [0, 0, 0, 1, 0, 0, 0, 0],
       [1, 0, 1, 0, 1, 0, 0, 0]])
```

补充：数据预处理

- 最小最大化：把原始数据映射到指定区间。下面的代码等价于 preprocessing.minmax_scale(data.astype(np.float64), feature_range=(1,5))

```
>>> data = data.astype(np.float64)
>>> preprocessing.MinMaxScaler(feature_range=(1,5)).fit_transform(data)
array([[5.        , 2.66666667, 4.25      , 1.        , 5.        ,
        5.        , 5.        , 5.        ],
       [1.        , 5.        , 1.        , 5.        , 1.        ,
        1.        , 1.        , 1.        ],
       [4.64285714, 1.        , 5.        , 3.26315789, 3.29333333,
        4.33333333, 2.67272727, 1.61176471]])
>>> preprocessing.MinMaxScaler(feature_range=(1,5)).fit(data).transform(data)
array([[5.        , 2.66666667, 4.25      , 1.        , 5.        ,
        5.        , 5.        , 5.        ],
       [1.        , 5.        , 1.        , 5.        , 1.        ,
        1.        , 1.        , 1.        ],
       [4.64285714, 1.        , 5.        , 3.26315789, 3.29333333,
        4.33333333, 2.67272727, 1.61176471]])
```

补充：数据预处理

```
>>> preprocessing.MinMaxScaler().fit_transform(data.astype(np.float64))
                            # 把原始数据缩放到[0,1]区间
                            # 可以消除不同单位的影响
                            # 例如以cm为单位的身高（一般小于200），
                            # 以ml为单位的肺活量（往往是几千）
array([[1.        , 0.41666667, 0.8125    , 0.        , 1.        ,
        1.        , 1.        , 1.        ],
       [0.        , 1.        , 0.        , 1.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.91071429, 0.        , 1.        , 0.56578947, 0.57333333,
        0.83333333, 0.41818182, 0.15294118]])
```

补充：数据预处理

- 正则化：对包含至少一个非0分量的样本都独立于其他样本进行缩放，使其l1或l2范数为1。

```
>>> preprocessing.Normalizer(norm='l1').fit_transform(data)   # l1正则化
array([[0.184     , 0.042     , 0.162     , 0.014     , 0.176     ,
        0.108     , 0.136     , 0.178     ],
       [0.14814815, 0.14403292, 0.11934156, 0.34156379, 0.05349794,
        0.12345679, 0.05349794, 0.01646091],
       [0.2175    , 0.0275    , 0.2325    , 0.125     , 0.14      ,
        0.125     , 0.09      , 0.0425    ]])
>>> preprocessing.Normalizer(norm='l2').fit_transform(data)   # l2正则化
                                                  # 等价于preprocessing.normalize(data)
array([[0.46754149, 0.10672143, 0.41163979, 0.03557381, 0.4472136 ,
        0.27442652, 0.34557414, 0.45229557],
       [0.33562878, 0.32630575, 0.27036763, 0.77381079, 0.12119928,
        0.27969065, 0.12119928, 0.03729209],
       [0.53892961, 0.06814053, 0.57609717, 0.30972966, 0.34689722,
        0.30972966, 0.22300536, 0.10530809]])
```

补充：数据预处理

- 标准化：使得数据符合标准正态分布（均值为0，方差为1），以满足某些模型对数据的要求。

```
>>> preprocessing.scale(data)    # 如果数据稀疏，可以加参数with_mean=False
array([[ 0.80360607, -0.13545709,  0.48001536, -1.27478864,  1.16071451,
         0.88900089,  1.2858235 ,  1.39988983],
       [-1.40960409,  1.28684238, -1.39204455,  1.16766354, -1.2800403 ,
        -1.3970014 , -1.15280728, -0.87381658],
       [ 0.60599802, -1.15138528,  0.91202919,  0.1071251 ,  0.11932579,
         0.50800051, -0.13301622, -0.52607325]])
>>> preprocessing.StandardScaler().fit_transform(data)
array([[ 0.80360607, -0.13545709,  0.48001536, -1.27478864,  1.16071451,
         0.88900089,  1.2858235 ,  1.39988983],
       [-1.40960409,  1.28684238, -1.39204455,  1.16766354, -1.2800403 ,
        -1.3970014 , -1.15280728, -0.87381658],
       [ 0.60599802, -1.15138528,  0.91202919,  0.1071251 ,  0.11932579,
         0.50800051, -0.13301622, -0.52607325]]
```

补充：数据预处理

```
>>> preprocessing.scale(data).mean()           # 所有特征的均值为0
1.5612511283791264e-17
>>> preprocessing.scale(data).mean(axis=0)
array([-1.48029737e-16,  7.40148683e-17, -1.11022302e-16,  6.01370805e-17,
       -7.40148683e-17,  2.22044605e-16, -9.25185854e-18,  1.11022302e-16])
>>> preprocessing.scale(data).var()            # 方差为1
1.0
>>> preprocessing.scale(data).std()            # 标准差为1
1.0
```

补充：数据预处理

- 填充缺失值

```
>>> data1 = np.random.randint(1,10,(5,5)).astype(np.float64)
>>> data1[2,3] = np.NaN
>>> data1
array([[ 1.,  4.,  9.,  1.,  7.],
       [ 8.,  7.,  2.,  1.,  6.],
       [ 3.,  8.,  3., nan,  8.],
       [ 7.,  2.,  4.,  3.,  4.],
       [ 5.,  8.,  3.,  6.,  7.]])
>>> preprocessing.Imputer().fit_transform(data1)    # 使用缺失值所在列的均值填充
array([[1.  , 4.  , 9.  , 1.  , 7.  ],
       [8.  , 7.  , 2.  , 1.  , 6.  ],
       [3.  , 8.  , 3.  , 2.75, 8.  ],
       [7.  , 2.  , 4.  , 3.  , 4.  ],
       [5.  , 8.  , 3.  , 6.  , 7.  ]])
```

补充：数据预处理

```
>>> preprocessing.Imputer(strategy='median').fit_transform(data1)
                                                    # 使用缺失值所在列的中值填充
array([[1., 4., 9., 1., 7.],
       [8., 7., 2., 1., 6.],
       [3., 8., 3., 2., 8.],
       [7., 2., 4., 3., 4.],
       [5., 8., 3., 6., 7.]])
>>> preprocessing.Imputer(strategy='most_frequent').fit_transform(data1)
                                                    # 使用缺失值所在列出现次数最多的值填充
array([[1., 4., 9., 1., 7.],
       [8., 7., 2., 1., 6.],
       [3., 8., 3., 1., 8.],
       [7., 2., 4., 3., 4.],
       [5., 8., 3., 6., 7.]])
```

补充：数据预处理

```
>>> preprocessing.Imputer(strategy='most_frequent',
                          axis=1).fit_transform(data1) # 缺失值所在行出现次数最多的值填充
array([[1., 4., 9., 1., 7.],
       [8., 7., 2., 1., 6.],
       [3., 8., 3., 3., 8.],
       [7., 2., 4., 3., 4.],
       [5., 8., 3., 6., 7.]])
>>> preprocessing.Imputer(strategy='most_frequent', axis=1,
                          missing_values=4).fit_transform(data1) # 把4当作缺失值
array([[ 1.,  1.,  9.,  1.,  7.],
       [ 8.,  7.,  2.,  1.,  6.],
       [ 3.,  8.,  3., nan,  8.],
       [ 7.,  2.,  2.,  3.,  2.],
       [ 5.,  8.,  3.,  6.,  7.]])
```

补充：绘制支持向量机用于手写数字识别的学习曲线

- 所谓学习曲线，是指随着样本数量增加时模型的表现，例如模型在训练样本和验证样本上的得分。

code\绘制支持向量机手写数字识别学习曲线.py