DT0139
设计建议

决策树生成

主要元器件			
产品型号	**市场**	**系列**	**机器学习核心（MLC）功能**
LSM6DSOX	消费电子	iNEMO 惯性模块（IMU）	256 个节点
LSM6DSRX	消费电子	iNEMO 惯性模块（IMU）	512 个节点
ISM330DHCX	工业	iNEMO 惯性模块（IMU）	512 个节点
IIS2ICLX	工业	2 轴加速度计	512 个节点

目的和益处

机器学习是计算机算法的一个研究领域，可以使用数据构建算法，无需明确推导数学模型。有四种不同类型的机器学习方法被广泛使用：

- 监督式学习
- 无监督式学习
- 半监督式学习
- 强化学习

监督式学习将已被标记为相关的真实信息的训练数据用于机器学习算法。例如，可以从系在测试对象手腕上的数据收集设备采集加速度计数据，这些信息可以被标记为不同的活动，如静止、步行、跑步、游泳或骑自行车。无监督式机器学习算法不需要标记数据，可以通过在数据中确定模式来创建模型。半监督式学习混合使用标记数据和未标记数据来进行训练，以

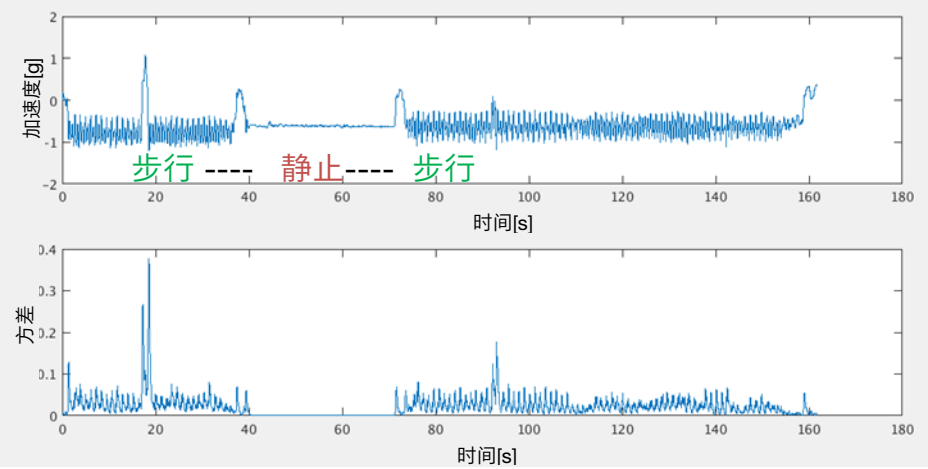
提高准确性。强化学习算法通过与现实世界的交互来推导和改进模型。更多关于机器学习算法的详细信息请参见机器学习。

说明

在本文中，我们将讨论一种监督式学习算法。最新一代意法半导体 MEMS 传感器内置一个基于决策树分类器的机器学习核心（MLC）。这些产品很容易通过后缀中的 X 来识别（例如，LSM6DSO**X**）。这种 MLC 可以在传感器中以极低的功耗执行程序化决策树。关于这些设备中机器学习核心的更多详细信息请参见相关应用笔记（LSM6DSOX 请参见 AN5259、LSM6DSRX 请参见 AN5393、ISM330DHCX 请参见 AN5392、IIS2ICLX 请参见 AN5536）。

一个配有机器学习核心的传感器可以处理传感器数据，并在一个数据窗口上计算 12 种不同类型的特征，如平均值、方差、带内能量、峰峰值、过零（正和负）、原始和过滤后的传感器数据峰值检测（正和负）。如果在所选的特征中观察到不同的模式，则可以使用决策树进行分类。考虑图 1 中所示的加速度数据示例。

图 1. 加速度数据及其方差



如图 1 所示，其为加速度范数在特定时间窗口上的方差图（本例中的特征值）。从第二个子图可以明显看出，“行走”和“静止”两个类别的方差特征表现出明显不同的值。在这种情况下，我们可以设计一个以方差作为特征之一的决策树来区分这两种不同类型的运动。

开发机器学习分类算法的一般流程如图 2 中所示。

图 2.　开发机器学习算法的基本过程

数据收集和数据标记（用于监督式学习）是前两个步骤。然后，使用训练数据（整个数据集合的子集）对模型进行训练。最后将模型传输并在配有 MLC 核心的设备中实现，利用嵌入式硬件进行实时验证测试。

图 3.　说明机器学习核心的实现过程

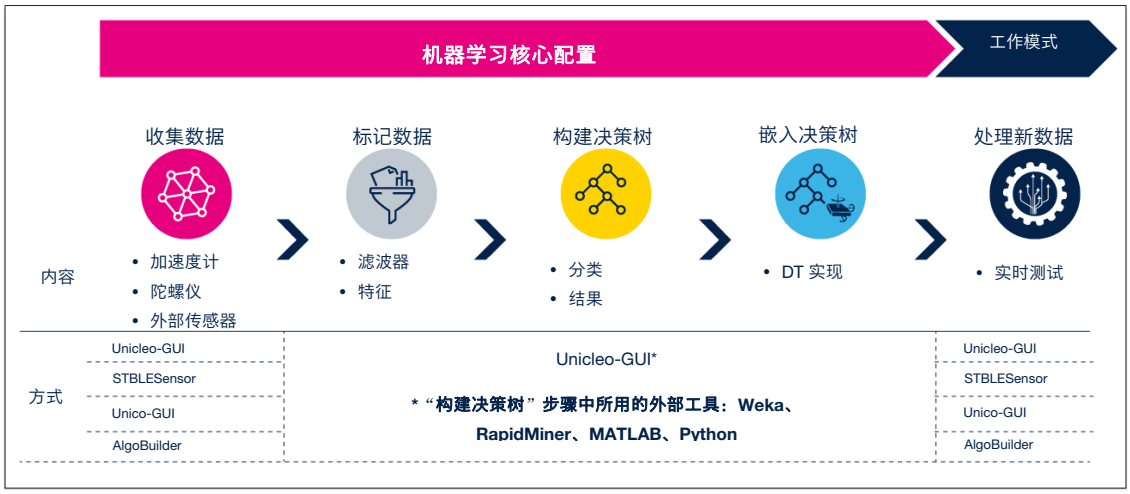


图 3 显示了 MLC 的处理步骤和一些可用工具。在这种情况下，可以使用（例如）意法半导体的 Unico-GUI 来处理前两个步骤，收集数据、标记数据并将每个类存储在特定的日志文件中。然后，可以使用任何可用的工具（如 WEKA、RapidMiner、MATLAB、Python（例如 sklearn））或 Unico-GUI 内置的决策树生成功能计算决策树模型。

数据收集和预处理将在本文档的第一节中介绍。第 2 节介绍如何进一步管理训练数据集，以及如何建立系统（特征选择、窗口长度定义、传感器 ODR 等）以便构建模型。关于决策树生成的详细信息在第 3 节中介绍。评估模型的方法和优化分类性能的方法将在第 4 节中讲述。

1. 数据收集和预处理

任何 ML 分类问题的第一步都是收集数据。可以使用各种应用程序（如 PC 上的 Unico-GUI、ST BLE Sensor app、AlgoBuilder、 FP-SNS-DATALOG1 等）以及不同的硬件设备选项（如 ProfiMEMS 板、SensorTile.Box、Nucleo 板和 STWIN）收集和标记传感器数据。

图 4. 机器学习核心体验的开发套件和 GUI

关于适当的数据收集活动的一般指南基于以下几点：

- **类别定义。**分类问题必须与待识别的类一起定义。此外，还应该明确当输入不属于任何已定义的类时会发生什么。在这种情况下，考虑将“other（其他）/idle（空闲）”类添加到类列表中。
- **传感器。**一些分类问题可以通过使用加速度计信号来解决，有一些需要使用陀螺仪信号，还有一些需要两个传感器输入。建议对两个传感器信号进行记录，以研究在使用 6 轴惯性模块产品的情况下，哪种配置可以提供最佳精度。也可以通过 6 轴 MLC 系列传感器的传感器集线器功能访问 3 轴磁力计。
- **传感器配置。**在机器学习核心配置工具（Unico-GUI）中，可以设置所有 MLC 参数，包括传感器速率（ODR）和满量程（FS）：
  - 确保以足够的 ODR 和带宽收集传感器数据，以捕获所需的信息。带宽（表示系统捕获的最大频率信号）通常是 ODR 的一半（BW ~ ODR/2）。如果不能预先精确选择

ODR，最好设置一个较高的 ODR（例如 104 Hz）最大带宽，然后使用抽样/重采样方法。还要考虑到必须为整个过程维护已定义的配置，并且 ML 算法将以相同的方式处理字段数据。

- 应根据应用要求定义满量程（FS）。例如，如果传感器用于车辆或手腕应用，陀螺仪满量程可能会迥然不同。

可以通过改变传感器配置来优化解决方案。

- **数据一致性和平衡。**根据应用需求收集能反映实际使用场景的足够数据。如果应用基于手腕式设备，不要包括智能手机上记录的数据。**为每个类收集类似数量的数据也很重要。机器学习训练算法会尝试达到最大总精确率，因此数据量小的类得到的权重更低，模型可能会偏向数据量大的类。**例如，如果“A”类有 90%的数据，“B”类有 10%的数据，那么模型将所有输出分配给“A”类，精确率为 90%。
- **数据记录格式。**如果使用其他工具进行数据记录，数据必须以 Unico-GUI 中定义的相同惯例记录在文本文件（.txt）中（详细信息请参见应用笔记 AN5259）。

第一阶段数据收集过后，需要一系列操作来处理分类问题。数据应进行验证和清洗，以避免异常值和标记不当。数据可视化对于数据验证、选择特征、标记等都是一种最佳实践。在下一节中，我们将讨论这些操作。

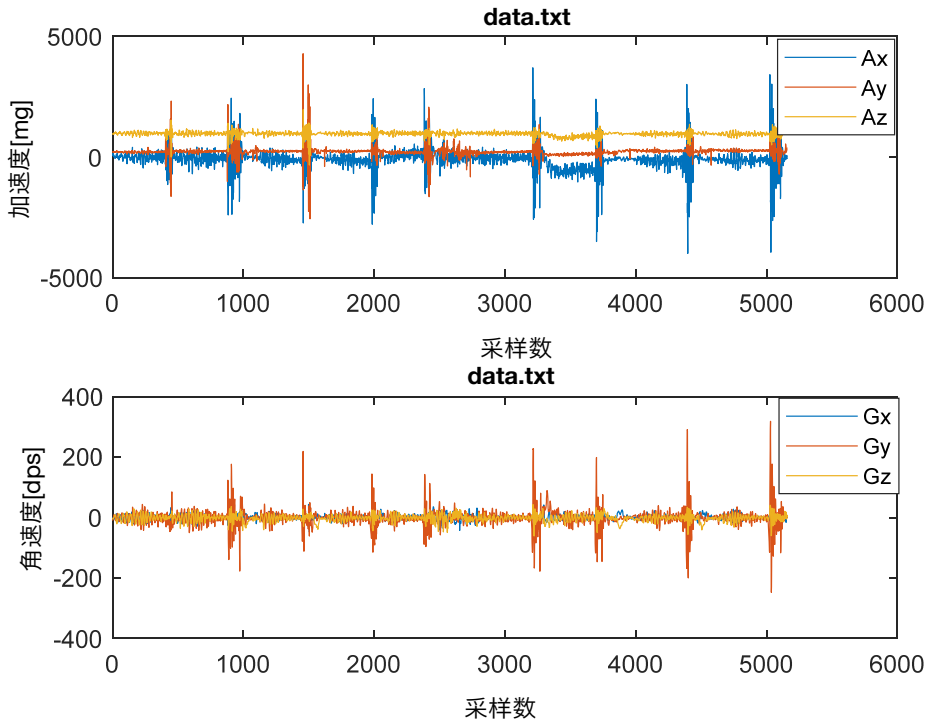
1.1 数据可视化

这是 ML 模型开发中最重要的步骤之一。数据可视化可以用于数据预处理和分析。它还可以用于理解不同类的不同特征和变体之间的关系。我们可以把该过程分为时域和频域：

1.1.2 在时域中分析信号

有多个可用的选项用于对收集的数据进行可视化。如果我们以[m*g*]为单位收集加速度计数据，以[dps]为单位收集陀螺仪数据，可以使用 MATLAB/Python/Excel/R 生成如下示例中所示的图。

图 5. 在时域中可视化数据



在时域中分析信号允许：

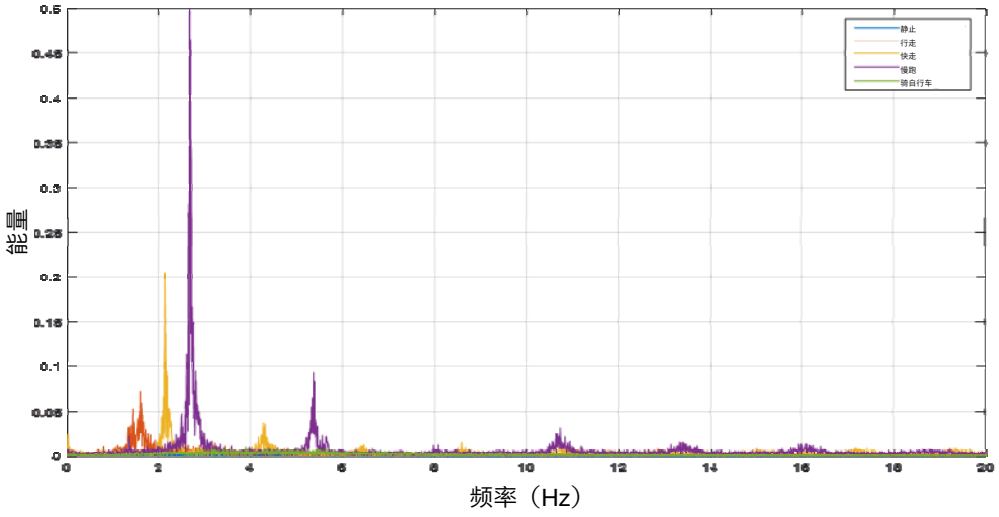
- 目测检查质量较差的数据或异常值。
- 确定不同类之间是否存在一种模式。
- 选择信息式特征（均值、峰峰值、峰值，等）。

1.1.3 在频域中分析数据

频域分析可以作为时域分析的补充。可以使用相同的工具（MATLAB/Python/Excel/R）计算并生成 FFT 或频谱图。这种图可以提供关于在处理信号以计算特征之前对信号应用滤波器的洞察力。或者它们对于评估特定记录的日志是否被正确引用比较有用。例如，考虑一个“步行”用例的数据日志。有了 FFT 图，可以验证与所处理信号相关的主频率。如果该频率太高（例如 2.5 Hz），我们可以假设日志应该被引用为“跑步”而不是“步行”。

图 6 显示在分类问题中考虑的不同人类活动（步行、快走、慢跑、骑自行车）的频谱。

图 6. 不同活动的频域分析



从图 6 中所示的图可以得到该信息。每个活动都有一个独特的主频分量，因此可以应用适当的滤波器为每个类别提取信息式信号。例如，如果我们应用一个 1.8 Hz 的低通滤波器，我们就可以将步行类别与快走类别和慢跑类别分开，后两者的特征是在更高的频点有更高的能量。

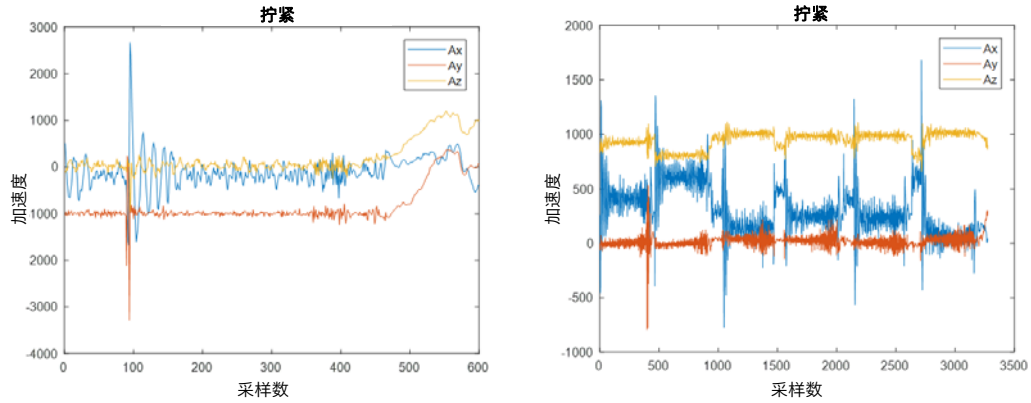
1.2 数据清洗

传感器数据标记是机器学习算法开发的起点。最佳实践是在开始 ML 训练练习之前，对收集的数据进行完整性检查。数据清洗是修复或丢弃已损坏、格式不正确或不完整数据的过程。用户可以轻松进行可视化和检查，例如，平均值、方差、最小值和最大值。如果计算的特征与相同标记类别中的日志明显不同（但它们在该应用程序中应该是相似的），则标记的数据可能不干净。

在大多数项目中，当我们记录数据时，我们知道并收集正确标记数据。通常一次日志记录的开始和停止只包含一个类。但是，由于各种原因，记录的单一日志文件中可能包含不同的类。在这种情况下，在日志文件被认为已做好 ML 训练准备之前，应该对数据进行删节，使每个文件只有一个类。

用户经常犯的一个错误是在同一传感器数据日志中包含多个类。在监督式学习中，数据清洗真的很重要，因为任何不属于相应类的数据都会影响已训练模型的性能。例如，下图（图 8 图 7）显示了为“拧紧”类收集数据时，来自固定在钻床上的加速度计的数据。这里的目标是收集类别的标记数据：空闲、拧紧、松开和钻孔。

图 7.　钻床数据采集实例



左手侧的图显示，当钻床运动停止时，日志收集没有停止。当没有钻床运动时，在图的末尾会有冗余数据。右手侧的图显示日志包含拧紧多个螺丝的数据，不同的会话记录在同一文件中。钻床的方位不断变化，Ax, Az 每隔[500,1000]秒间隔发生变更。所有这些问题会实时影响生成的决策树和 MLC 结果的性能。几个建议：

- 每个文件中标记数据的单位应该是一致的。我们以[m*g*]作为加速度的单位，以[dps]作为角速度的单位。
- 每个数据收集会话都必须根据适当的时间段进行删节。
- 基于类的单独数据，使用任何可用的工具，如 Python、MATLAB、R
- 删除不属于任何类的数据，或将其用于“other（其他）/idle（空闲）”类。
- 根据需要进行适当缩放。

2. 窗口大小和特征选择

完成对所收集数据的预处理后，下一步就是确定窗口长度，开始特征选择过程。

2.1 窗口大小

窗口长度紧密依赖于应用。我们应确保窗口能够捕获所有信息，以便决策树区分不同的类。由于对每个窗口的数据都进行了预测，因此较短的窗口长度让响应更快。然而，如果窗口长度太小，无法捕获足够的信息，也会导致精度下降。用户应在完成数据分析和可视化后谨慎选择窗口大小。如果运动是在某频率上重复的，则理想的窗口应该足够长，确保捕捉整个运动模式。此外，还请考虑变化最慢的信号。例如，如果我们要区分 A 类（周期为 0.9 Hz）和 B 类（周期为 1.9 Hz），我们应该根据 A 类来确定窗口大小，并将窗口大小设为（1/0.9 Hz）≈1.1 s 或更长。

2.2 特征选择

2.2.1 观察

我们可以通过观察运动来选择信息式和相关特征。考虑这样一个例子，我们将检测头部姿势，例如当传感器安装在耳机上，Y 轴指向地面（重力方向），X 轴与用户行进方向对齐时的“点头”姿势。在这里，我们可以选择 X 轴和 Y 轴加速度数据的方差、平均值、最小值和最大值，因为用户点头时，X 轴和 Y 轴的这些特征会发生变化。

再举一个无具体方向的特征的例子。如果我们在钻床上安装传感器，我们不希望因为钻床必须使用的方向而限制钻床的使用。在这种情况下，我们不应该应用依赖于方向的特征，例如，x、y、z、平均值、最小值、最大值。我们可以选择方差、过零、峰值检测、能量范数（x、y、z 轴的大小）。这些特征与钻床的方向无关，这些特征最适用于振动检测。

2.2.2 利用模型训练

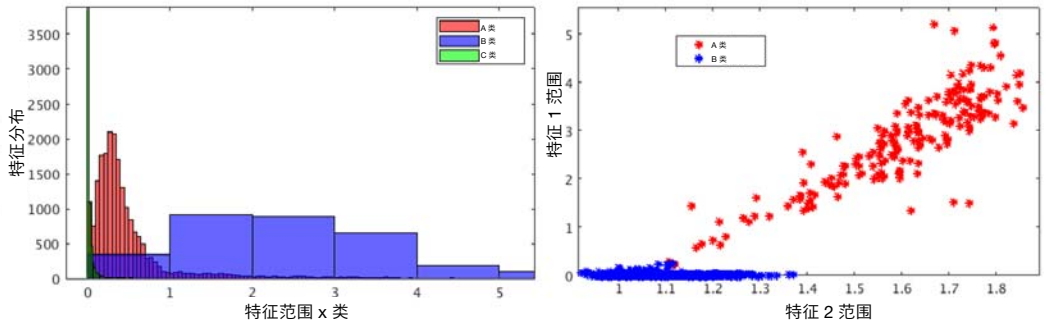
获得第一个可能的候选特征池的一种方法是选择所有可用的特征并根据所选的分类器训练模型。在我们的例子中，可用特征在 MLC 中预先定义，分类器是一个需要训练的决策树。在训练模型时，决策树学习算法将选择能够更好地区分类别的特征。我们还建议选择直观的特征来解决该问题。然后，这些特征可能会被选择为机器学习模型下一步开发中唯一使用的特征。

该方法也有一些缺点，比如可能由于使用了太多的特征或与特定类的检测实际上不相关的特征而导致模型过度拟合。因此，始终建议也要依赖其他方法进行特征选择，并尽可能避免最初就使用所有特征。一个更好的方法是首先确定一组潜在的优良特征，然后训练模型，以了解它使用哪些特征。

2.2.3 特征可视化

可视化有助于选择和删除特征，以优化模型的性能。单一特征的信息贡献量评估方法：衡量分离了多少不同的类。可以使用图 8 中所示的 1D/2D 表示法以可视方式进行分析。

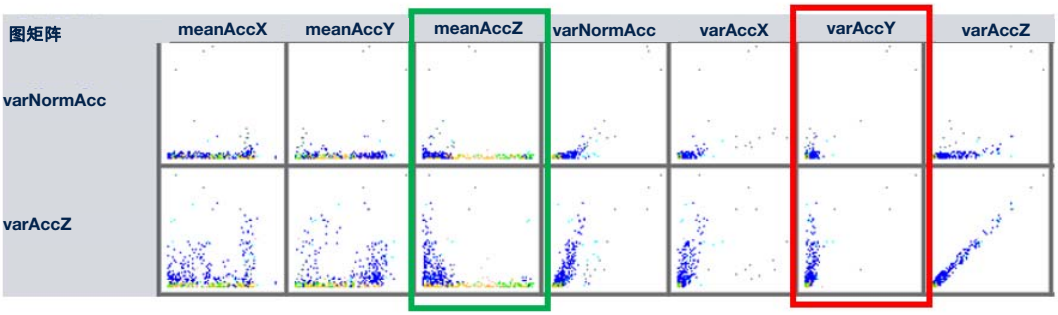
图 8. 三个不同类别的单一特征计算值的柱状图。对这些特征进行了合理分离，这意味着所选特征对于区分不同的类有一定帮助特征和类可视化。



如果可能的话，绘制 2 个特征的笛卡尔图可能有用，如上图（右侧），其中显示了与 2 个类别分类问题有关的 2D 图和 2 个不同特征的选择。在这种情况下，很明显进行了严格分离，因此，所选两个特征组合后表示的信息是可见的。这种图还有助于评价不同特征和类别之间的关系。左边的图是每个类的单一特征的直方图。每个类的特征值分布截然不同，即使使用一个简单的阈值，我们也可以看到类是可以分离的。

ML 工具（比如 WEKA）也支持同样的方法。下面显示的两个图代表了不同特征和类之间的关系。特别是，我们可以看到，绿色块中的特征组合具有可分离性，而红色块中的特征组合则没有可分离性。

图 9. WEKA 中的特征可视化



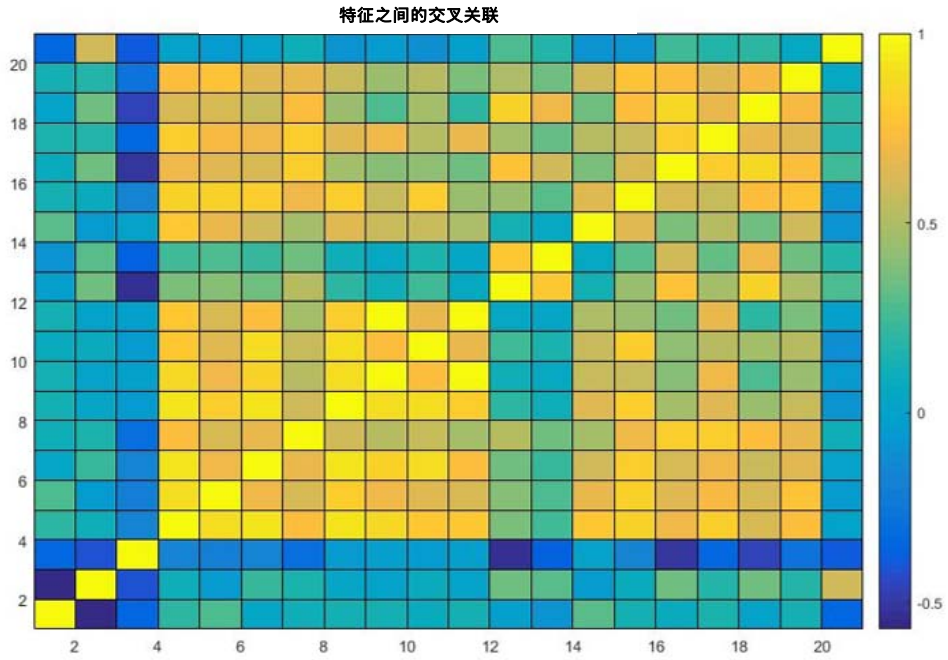
在我们将该特征纳入最终计算之前，需要考虑以下事项：

- 如果该特征与另一个特征有线性关系或强相关，则该信息已经被另一个特征捕获，因此该特征可以被丢弃。如果难以实现可视化，可以计算协方差矩阵以识别最相关的特征，

如图 10 中的一个例子所示。在该图中，我们可以看到有许多非对角线元素，它们之间有很强的相关性，因此它们可以被丢弃。

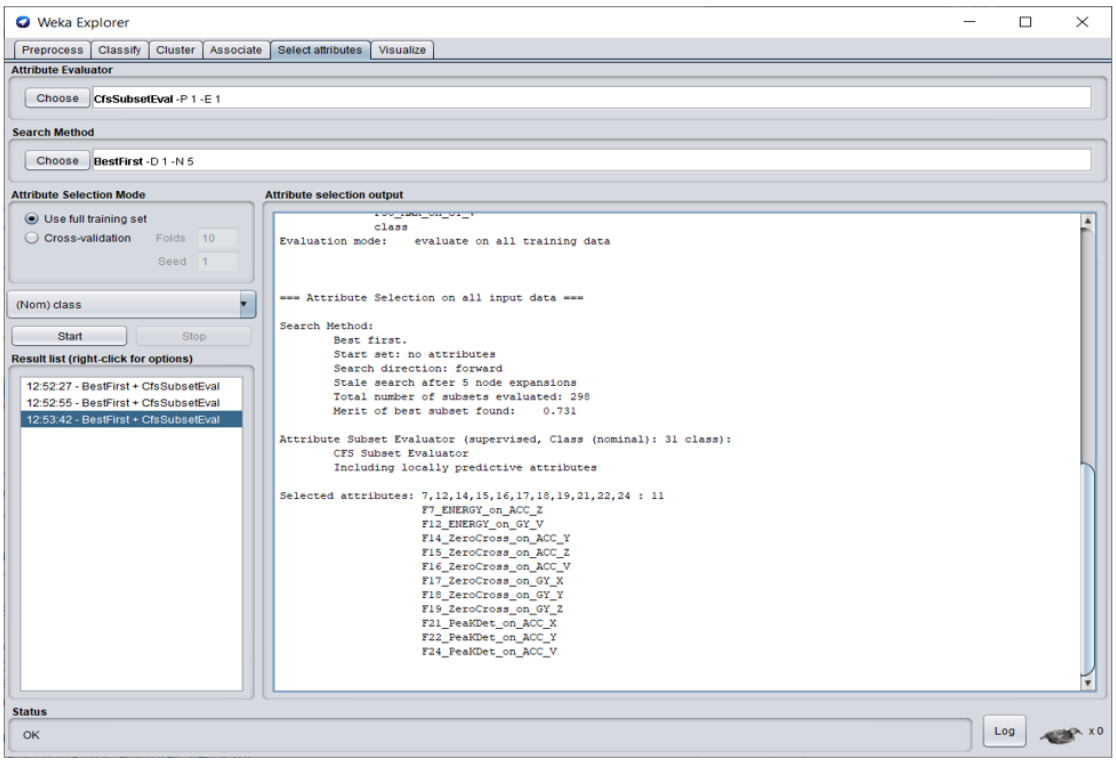
- 可变性：如果特征在不同的类之间没有显示出太大的差异，那么该特征可能无助于分离类。

图 10. 特征之间的交叉关联



最后，WEKA 提供一些内置工具，用于识别信息式特征的最佳子集。在图 11 中，一个示例显示了在 Weka 中获取特征排序的过程。在所示的示例中，ARFF 文件有 30 个特征，通过使用特征排序逻辑，Weka 仅为当前数据集选择 11 个重要特征。

图 11.　Weka 中的特征选择方法

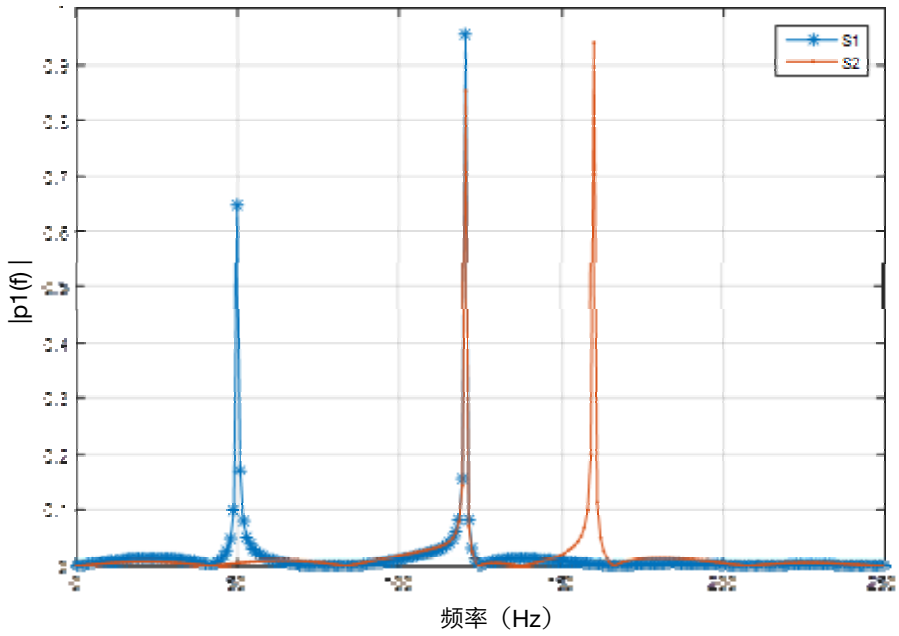


2.3　滤波器选择

基本特征如均值、方差和能量包含直接信息，但可能无法分离具有不同主频率的类。对信号应用滤波器可以从信号中隔离特定信息，还可以成功地分离类。我们可以对原始信号应用专用滤波器，以选择特定的频率范围。对于人类活动检测，我们可以使用适当的滤波器，精确地区分用户是步行（通常为 1 - 2 Hz）还是慢跑（2.5 - 4 Hz）。滤波器选择应通过对信号进行傅立叶分析和观察信号的功率谱密度来实现。通过傅立叶分析可以了解每个类中哪个频率范围处于主导地位。

该过程包括选择合适的滤波器（低通、高通、带通）并确定截止频率。选择滤波器和感兴趣的频率是通过在类之间的非重叠区域中发现大量能量来实现的。下面的图 12 说明 S1 类和 S2 类的重叠区域和感兴趣的频率范围。

图 12. 两个不同类的频率分析



如果我们计算总能量（所有频率（0 - 250 Hz）的总功率），它将不会提供特定频率（例如 50 Hz）的能量信息。因此，如果不同类的总能量相似，但在不同的频率有不同的功率，则可能不足以分离两个类。在计算能量之前，用户必须应用适当的滤波器将信号从特定的频率范围中去除。应用滤波器将允许信号通过低于（在低通滤波器中）截止频率的阈值。如果我们以图 12 中的示例为例，应用 60 Hz 的低通滤波器，信号 S2 将被完全滤除，部分低于 55 H 的 S1 信号将通过。现在如果我们计算滤波后两个信号的能量，S1 的能量高于比 S2，量因为 S2 信号被完全过滤掉了。上述案例中推荐的滤波器可以是 50 Hz 和 150 Hz 的带通滤波器，或者 55-100 Hz 和 125-150 Hz 的低通滤波器。

2.4 训练-测试拆分

一旦我们决定了窗口长度，下一步就是决定如何将数据拆分为训练集和测试/验证集。对于每一个机器学习模型，将数据划分为测试、训练和验证（如果需要）是很重要的。在选择模型、检查最终模型的质量（欠拟合、过拟合）时，需要对数据进行拆分。训练数据用于训练模型，测试数据用于评估已训练模型的质量/准确性，如果我们可以从多个模型中选择合适的模型，则使用验证数据。测试和训练数据的准确性可帮助判断模型是欠拟合、过拟合还是非常平衡。例如，如果训练数据的准确性很高（>80%），而测试数据的准确性较低（<70%），则模型很可能过拟合，可能需要进行一些剪枝或添加更多不同的数据。另一方

面，如果训练数据和测试数据的准确性较低（< 60%），则模型大概率欠拟合，可能需要添加更多的特征或减少剪枝。更多详细信息请参见第 4 节。

通常情况下，如果我们评估单一模型，训练样本和测试样本的比例大约是 80:20 或 70:30。如果我们有兴趣评估多个模型（来自不同的训练算法 J48、CART 或不同的特征集），那么我们应该以 60:20:20 的比例将数据拆分为训练、验证和测试三部分。验证数据应仅用于选择最佳模型，测试数据应用于评估所选模型的最终准确性。数据划分比率的选择是特定于问题的，可以根据应用的需求进行更改。

如下所示，Python 代码将整个数据分为训练数据和测试数据。

```
from sklearn.model_selection import train_test_split
# split train : test = 70 : 30
X_train, X_test, y_train, y_test =
  train_test_split(wholeData, label, test_size=0.3)

# split train : test : validate = 70: 15 : 15
X_test, X_validate, y_test, y_validate =
    train_test_split(X_test, y_test, test_size=0.5)
```

上面列出的 Python 代码将首先以 70:30 的比例将整个数据集拆分为训练和测试两部分。然后它将测试数据集分成两半，其中一半作为验证数据集。训练、测试和验证数据之间的最终比例可以是 70%、15%和 15%。

有一个简单的规则用于决定是否需要拆分验证数据集。如果有一个模型，则将数据拆分为训练和测试两部分（30%-70%或 20%-80%），并在测试数据集上对用训练数据训练的模型进行评估。如果有多个模型，则将数据拆分为测试、验证和训练三个数据集（15%-15%-70%或 20%-20%-60%）。我们可以对训练和验证进行交叉验证。然后，我们通过验证选择模型，并在测试数据集上评估最终模型。

3 决策树

决策树是由节点和叶子（没有子树的节点）组成的二叉树结构。一个决策节点有两个分支。叶子节点产生分类输出或决策。在每个决策节点上，根据一个输入特征的阈值对数据进行拆分。通常，小于阈值样本会选择左节点，否则选择右节点。在遍历决策节点时，对数据进行越来越多的拆分，该过程在具有决策的某个叶子处停止。每个叶子都与一个类标签关联，从根到叶子的路径产生分类规则。

3.1 不同类型的决策树

有不同的算法可生成决策树分类器。

不同算法在概念上的区别不仅在于在节点进行划分的方式，还在于特征重要性、剪枝准则和停止准则方面的不同。决策树训练的基本步骤是：

a. 拆分（节点）：拆分准则如 Gini、信息增益或熵允许使用某个特征最大化信息（最大化分离类）。该过程包括选择最佳特征和相应的阈值。图 14 中的两个示例说明了不同类的特性分布情况。如果我们选择此特征和合适的阈值，我们可以看到右侧的示例不会像左侧示例那样轻松将类分离。这些准则（如 Gini、熵、信息增益）反映了在给定的阈值下，类别之间的分离。

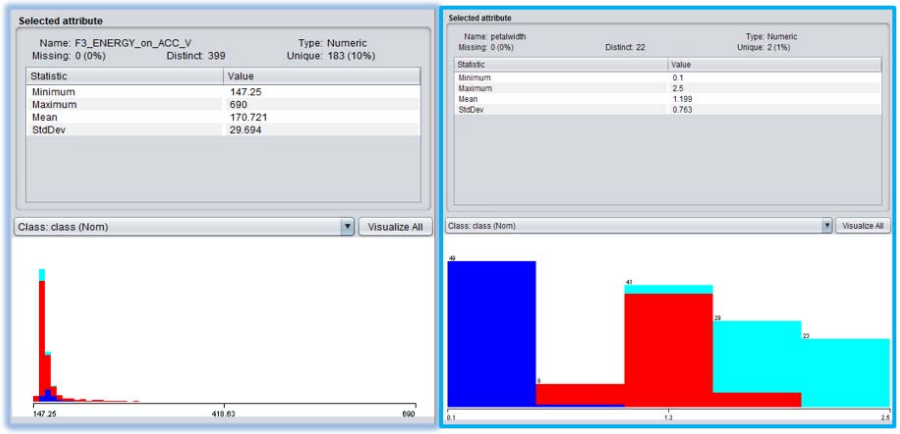
b. 剪枝：决策树的剪枝过程包括删除最终决策树中的某些节点，以减少尺寸和避免过拟合。我们将在 4.2.4 节介绍一个剪枝方法的例子。

c. 停止准则：决策树可以深入增长以达到 100%的准确率，但这样会导致过拟合问题。每个算法使用不同的准则阻止节点进一步拆分。准则可以是每个节点中的最小样本数或用于进一步拆分的最小增益。下表总结了不同类型决策树算法的总体差异。

图 13. 不同决策树算法

算法	拆分准则	剪枝准则	其他功能
CART	• Gini • 二分规则	交叉验证 后剪枝	1. 回归/分类 2. 名义/数值属性 3. 缺失值 4. 斜拆分
ID3	信息增益	预剪枝	1. 分类 2. 名义属性
C4.5	• 信息增益 • 信息增益率	基于统计的后剪枝	1. 分类 2. 名义/数值属性 3. 缺失值 4. 规则发生器

图 14. 不同类的特征分布

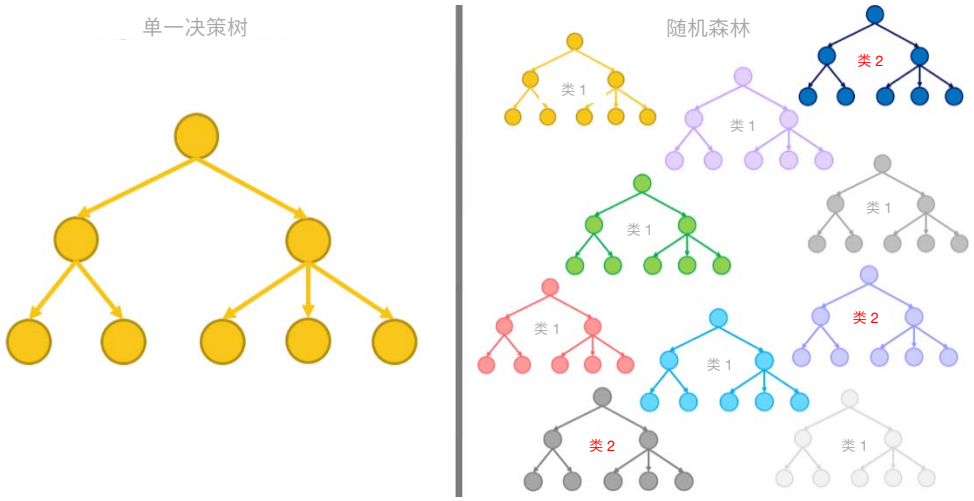


每一种决策树训练算法都有特定的优势，但 C4.5 是上述算法中最新的一种，它对数据集中的异常值具有很强的抗干扰性，可以生成均衡的决策树。

3.2 决策树 vs. 随机森林

随机森林是另一种分类方法。多决策树可以产生比单一决策树更准确的预测，因为它是使用不同准则生成的。换句话说，随机森林构建 K 棵（预选的，树的最大数量）略有不同的已训练决策树并将它们合并，以获得比单一决策树更准确和稳定的预测。下图说明了这一概念。

图 15. 单一决策树 vs. 随机森林



我们通常使用数据集的 70%作为训练数据，剩下的（30%）作为测试数据。随机森林的一种实现方法被描述为：对于随机森林中所用的每一棵决策树，我们使用增强或直接随机抽样选择数据集和不同的特征集以得到不同的决策树，而最终输出将会是不同决策树输出的平均值。因此，随机森林不存在过拟合的问题，可以提供更合适、准确、稳定的预测。有了 MLC，我们可以复制类似的行为，因为它允许同时运行 8 棵决策树对相同的问题进行分类。然后，我们可以将结果在一个微控制器上合并，该微控制器可以处理这些决策树的输出。

3.3 构建和可视化决策树

在 Python 中，我们可以在训练完成后将决策树可视化。下面是可以用于实现该过程的代码，步骤包括：

- 将数据拆分为训练数据和测试数据
- 构建决策树
- 决策树可视化

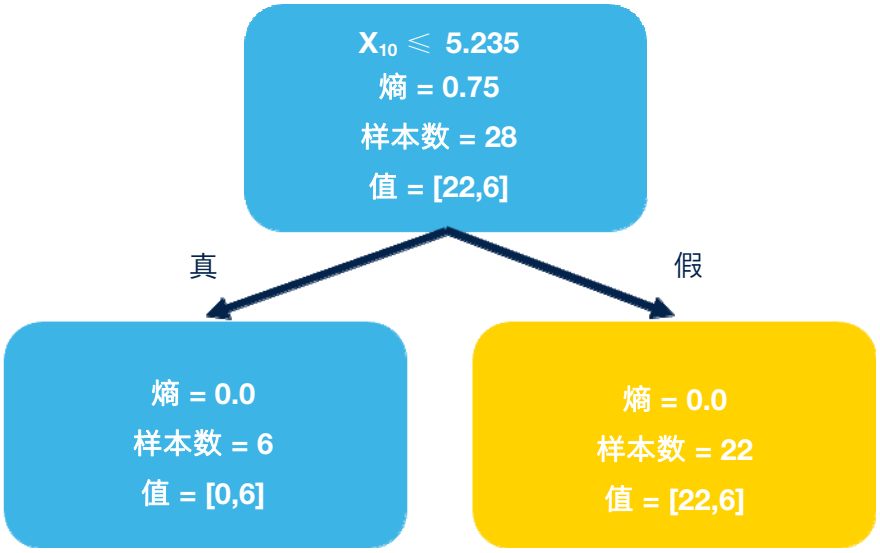
```
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.externals.six import StringIO
from sklearn.tree import export_graphviz

X_train, X_test, y_train, y_test = train_test_split(
    segm, labels, test_size = 0.3, random_state = 101)

clf_entropy = DecisionTreeClassifier(criterion =
    "entropy", random_state = 100)
dot_data = StringIO()
export_graphviz(clf_entropy, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True)
graph = pydotplus.graph_from_dot_data(
    dot_data.getvalue())
```

这是一个非常简单的示例，只有一个节点和两片叶子。上面的代码将生成图形化的决策树，如图 16 中所示。

图 16. Python 中的决策树图示例



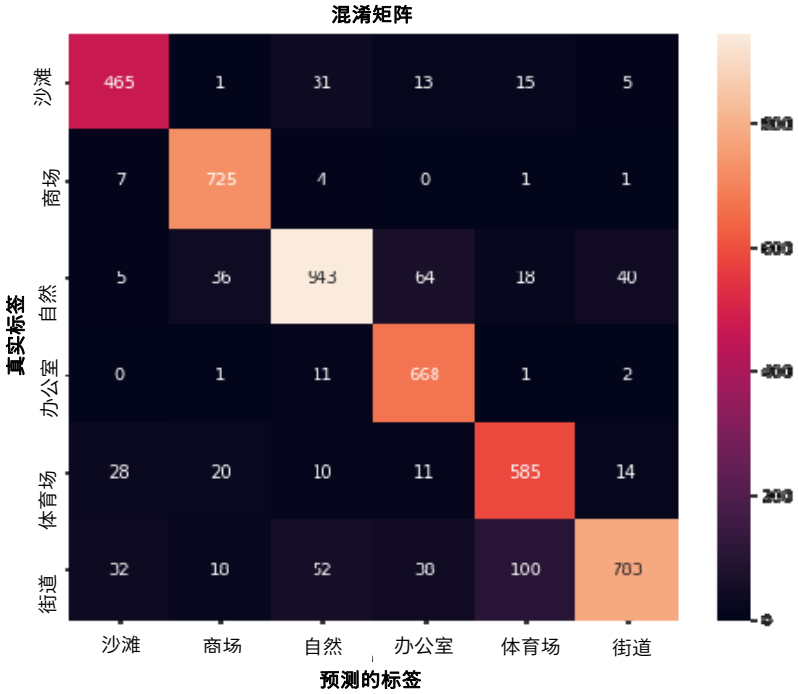
节点中的第一行表示用于拆分数据的某个特征的条件。下一行是关于熵的，表示响应（类）的变化。如果只有一个类，则熵字段为零。样本字段表示到达给定节点或叶子的样本总数。值字段包含关于单个类中样本的信息。我们应该注意，每次拆分后的熵都减少了，减少得越多，节点的重要性就越高。

如果决策树比较小，可视化有助于理解节点中的规则/条件。在一些情况下，它可以让我们了解规则设置是否如预期。

3.4 建立预测和混淆矩阵

混淆矩阵用于总结和显示决策树的分类性能。该矩阵提供关于多个类之间正确分类和混淆的概述。由于可以有不同的定义来表示分类算法的准确性，使用混淆矩阵是避免对“准确性”产生误解的最直接方法。让我们以下图为例进行说明。

图 17. 用于音频空间环境分类的混淆矩阵



我们可以将行设置为真正的类，将列设置为预测的类。对于给定的类，分类算法的输出输入到相应行的每个单元格中。这将产生决策树算法在地面真值和预测结果之间的映射。这里，我们可以看到算法混淆了“街道”类和“体育场”类。从混淆矩阵最后一行的第五个单元格可以看出，算法错误地将“街道”类的 100 个样本预测到“体育场”类中。下面的 Python 代码让我们可以构建如图 17 中所示的混淆矩阵。

```
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.externals.six import StringIO
from sklearn.tree import export_graphviz
from sklearn.metrics import confusion_matrix

def confusion_matrix(y_test, y_pred, class_list):
    confusion_matrix = metrics.confusion_matrix(y_test,
               y_pred)
    fig_confuse = plt.figure(figsize=(4, 3.5))
          sns.heatmap(confusion_matrix, xticklabels=
          sorted(class_list), yticklabels = sorted(class_li
          st), annot=True, fmt="d");
    plt.title("Confusion matrix")
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show();

class_list = ["beach", "mall", "nature", "office",
                  "stadiu  m", "street"]
y_pred_entropy = clf_entropy.predict(X_test)
confusion_matrix(y_test, y_pred_entropy, class_list)
```

4. 模型评估

我们如何知道我们训练的决策树模型是否足够优秀？有多种方法可以度量模型的性能。首先，我们必须用测试数据集实现混淆矩阵。为了选择稳定的模型，可以实现 K-fold（交叉验证）。下一节将解释如何确定模型的质量以及如何修复模型中的任何问题。

交叉验证也被称为抽样外技术。我们通常将数据随机拆分为测试数据和训练数据，对模型进行训练和测试。假设训练数据已经捕获了不同数据，并且足够用于模型训练练习。使用交叉验证有 3 个具体原因：

- 首先，对于一些特定问题，随机抽样方法选择的所有数据有可能是测试数据的一部分，而不是训练数据的一部分。在这种情况下，由于模型没有针对特定用例进行训练，所以模型性能不佳。考虑这样一种情况：数据季节性变化，随机抽样无法拆分数据并使其成为训练和测试数据的一部分。
- 其次，由于我们执行随机拆分，模型的准确性在每次随机拆分后都会不断变化。
- 第三，用于验证的数据仅用于验证和选择模型；一旦选定了模式，我们就无法通过该数据来提高模型的准确性。

可用的交叉验证技术有很多种。每一种都有特定的优点和缺点。标准的交叉验证技术有：

- 留一法
- 留 P 验证

- 重复随机子抽样
- K-fold
- 留出法

最常用的方法是 K-fold 交叉验证（特别是 10-fold）。如下图图 18 所示，在 K-fold 交叉验证中，训练数据需要进行 K-fold 拆分，训练次数为 k 次。

图 18. 5-折交叉验证

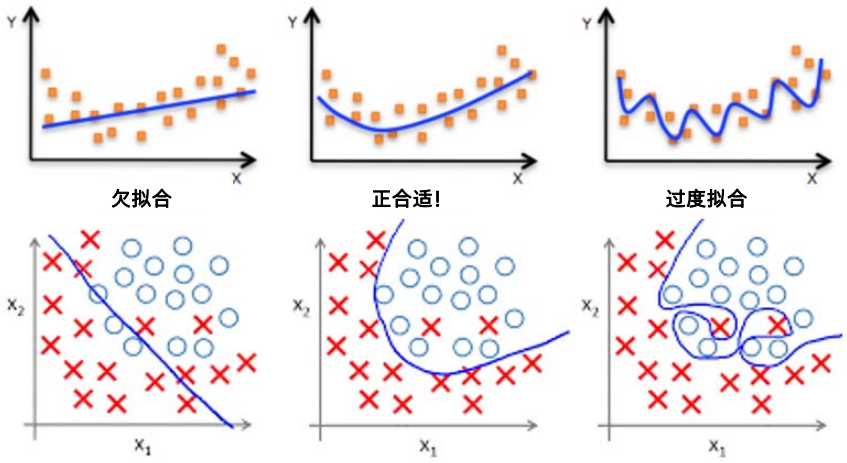
其中一个子集用作验证集，其他 k-1 个子集用作训练集。通过这种方式，每个子集将被 k-1 次用于训练，还有 1 次被用于验证。这种方法的好处是，误差估计是对所有 k 次试验进行平均，可获得模型的总有效性。在 K-fold 验证的最后，将利用所有训练数据准备最终模型。

当我们进行 K-fold 交叉验证时，其验证通过统计方法总结各种误差，如均方误差、均方根误差或绝对中位差。在选择合适的模型以确定模型的质量（欠拟合、过拟合等）时，这些参数是有用的。在下一小节中，我们将更详细地探讨模型的质量。

4.1 欠拟合

如果机器学习模型不够复杂或没有足够的信息（特征）来捕捉数据集中包含的关系和信息，就会发生欠拟合。欠拟合、拟合良好（稳健）和过拟合的概念在图 19 中说明。

图 19. 欠拟合、拟合良好（稳健）和过拟合的概念



如何判断模型是否欠拟合？模型欠拟合时，它无法捕获训练数据中包含的数据和模式。换句话说，如果已训练模型在训练数据和测试数据方面都表现不佳，可能是由于欠拟合。在该场景中，所选的特征可能不足以为模型捕获适合数据的信息。

欠拟合很容易处理，标准解决方法是添加额外的特征并重新训练模型。另一种选择是减少类的数量。在某些情况下，如果两个类之间的数据看起来相似，则模型可能不能恰当地对其进行分类。添加不同的数据可能也有助于解决欠拟合问题，因为模型可能会变得更复杂，并学习在之前训练中错过的模式。

4.2 过度拟合

与欠拟合不同，过拟合是一种建模误差，当模型在已训练的数据点上表现良好，但在不可见的数据上表现不佳时就会出现过拟合。它在训练数据集上具有较高的准确率，但在测试或其他新采集的数据集上准确率较低。过拟合模型通常构建一个过于复杂的模型来捕获训练中所用数据中的特征。当模型拥有高度灵活性和自由度来拟合尽可能多的点（将噪声解释为实际模式）时，就会出现这种情况。K-fold 验证或任意交叉验证通常都有助于识别过拟合。如果我们在训练数据集上运行模型时，其预测准确率为 95%或更高，但在测试数据上运行，其准确率为 50-70%，则这是识别过拟合问题的一种好方法。K-fold 验证中的 RMSE（均方根误差）是验证数据进行不同折验证时的累积误差。如果最终模型的准确率不错，但 RMSE 很高（>0.2），则是过拟合的明显迹象。

以下方法可帮助解决过拟合问题。

4.2.1 收集更多日志

通常情况下，这是一种比较简单的过拟合问题解决方案。新日志应该在不同的情况下收集，以确保更加多元化。收集的数据应该捕获模型在实际应用中可能遇到的各种用例。

4.2.2 数据扩增

有时，我们没有多个用户来收集日志，或者不能轻松拥有多元化的环境/条件。我们可以考虑添加噪声或以不同角度旋转数据。以一个头部姿势检测项目为例。该项目的目标是检测以下类[“点头”、“摇头”、“静止”、“行走”、“摆头”]，传感器附设在耳机上。我们期望 Y 轴指向地面（与重力方向相同），X 轴与用户行进方向一致。

然而，如果测试数据仅有数量有限的用户，并且有限的方向设定，则模型可能会遇到两种问题。首先，基于该数据进行训练的决策树将针对收集日志的人进行微调，并在其他不可见的用户上表现欠佳。其次，该模型针对特定方向进行调整，性能可能会受到用户对耳机方向的偏好影响。要克服这些问题，我们可以：

- 如前所述，注入噪声以处理数据中的任何细微变化。
- 以可能的不同角度（在使用期间）旋转现有数据，并将旋转后的数据与训练数据集连接起来。已训练模型将涵盖用户的大多数首选方向，解决方案将足够稳健，以抵消传感器方向的影响。

4.2.3 减少特征数量

在训练过程中选择大量特征可能允许以更大的灵活性和自由度将模型拟合到所有训练数据，但这样可能导致过拟合。去除不是很重要的特征有助于降低模型的复杂度并解决过拟合问题。正如我们在特征可视化一节所提到的，绘图和可视化特征有助于理解特征和类之间的关系。如果该特征没有意义，用户可以将其从模型中删除以减少复杂性。

4.2.4 剪枝

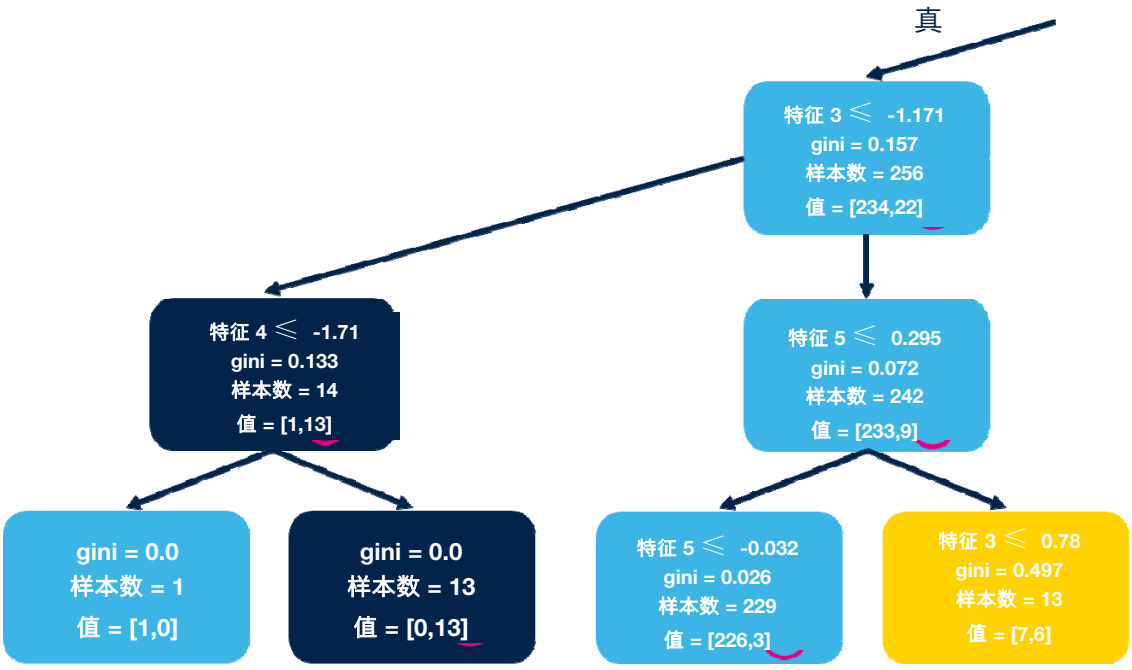
剪枝意味着通过删除决策树中不重要的部分来简化决策树。这可以对应没有对太多数据点进行分类的节点。您可以把这看作是模型生长过程中的早期停止点。我们在该点停止任何节点的分支。不同类型的决策树有不同的剪枝准则，如图 13 中所示。有两种类型的剪枝：

- 预剪枝：预剪枝在决策树继续对子集进行分类之前停止树的生长。
- 后剪枝：构建决策树之后，由逻辑决定是否修剪某个节点或子树。

大多数时候首选后剪枝，因为很难知道停止决策树生长的确切时间。在后剪枝过程中，逻辑首先生长完整的决策树，然后使用准则（如信息增益，或类的不平衡）开始删除决策树的非关键部分。我们在这里提供一个实例，以便用户理解后剪枝。如果决策树中有某些节点只对某一特定类的少数样本进行分类，而其余大量样本属于其他类，则将该节点替换为对所有属于“其他”类的样本进行分类的叶子。生成的决策树的性能不会受到此操作的不利影响。

剪枝减少了叶子（属于决策树的一部分），可能导致过拟合。这就是剪枝有助于减少过拟合并使解决方案变得稳健的原因。让我们以下面图 20 中所示的节点为例。

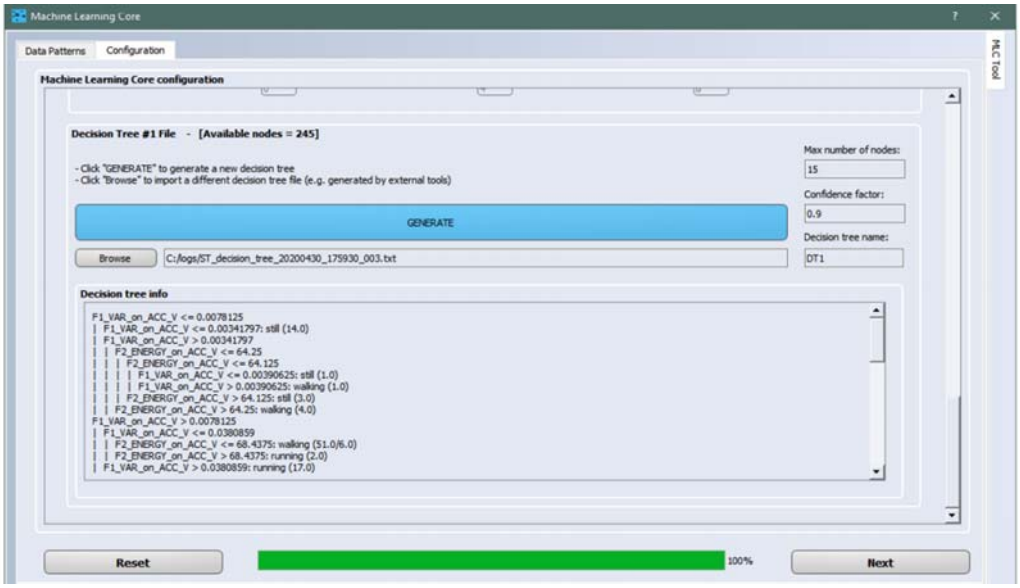
图 20. 剪枝示例



上述决策树的左侧节点（特征 4⩽-1.71）仅被分类为异常样本，其余 13 个样本均属于另一类。在这种情况下，最好将该节点作为叶子。

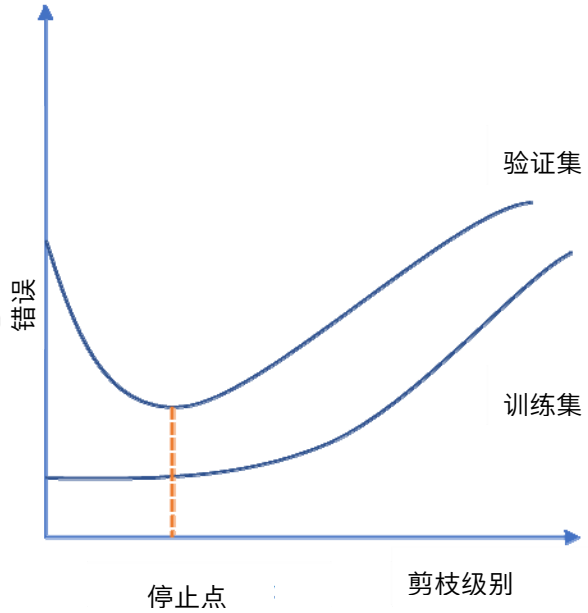
Unico-GUI 中的决策树算法具有剪枝选项。我们可以设置有限数量的节点，可信因子会进行剪枝，直到我们满足条件。

图 21. Unico-GUI 中的 MLC 默认决策树算法



确定最佳剪枝的方法是监测验证和训练的准确性。如果我们根据剪枝级别绘制验证和训练数据集的准确性/误差图，往往会得到如下图 22 中所示的图。

图 22. 剪枝的停止点



如果没有剪枝，训练数据上的模型误差将最低，但过拟合会导致验证集上的误差很高。随着我们提高剪枝级别，训练集中的误差会增加，但是验证集上的误差会开始减少，因为一些节点对噪声过拟合。随着我们进一步增加剪枝，验证集和训练集上的错误都会增加，因为剪枝将删除正确分类的节点。就在改点停止剪枝过程。

4.3 元分类器

元分类器为机器学习模型提供一种额外方法，以提高准确性，特别是已计算模型的输出稳定性。元分类器可应用于分类算法（不受决策树形式的限制），充当决策树输出上的平滑滤波器（如最大投票权）。各自的 MLC 应用笔记中有对元分类器的详细说明。

下面的例子说明了在类之间转换不频繁的用例中使用元分类器的好处。考虑对两个类进行活动检测的案例：当传感器安装在智能手表中时，有“清醒”和“睡着”两个类。当用户实际上“睡着”时，轻微的手腕运动就会触发分类输出到“清醒”。元分类器可以平滑分类算法的输出，并从输出中去除虚假的“清醒”分类。

辅助资料

产品型号	市场	系列	应用笔记
LSM6DSOX	消费电子	iNEMO 惯性模块（IMU）	AN5259
LSM6DSRX	消费电子	iNEMO 惯性模块（IMU）	AN5393
ISM330DHCX	工业	iNEMO 惯性模块（IMU）	AN5392
IIS2ICLX	工业	加速度计	AN5536

机器学习资源

面向 MLC 的 GitHub 项目	**https://github.com/STMicroelectronics/STMems_Machine_Learning_Core**
意法半导体 MLC 生态系统	**www.st.com/mems-sensors-ml**
面向 ML 的意法半导体 MEMS	**community.st.com/s/group/CollaborationGroup**
MEMS 和传感器问答	**Mems 和传感器问答**

版本历史

日期	版本	变更
2020 年 11 月 03 日	1	初始版本
2021 年 2 月 08 日	2	改进了整个文档，添加了图 4，更新了图 16、图 20，添加了“机器学习资源”