第1章

机器学习概述

机器学习研究的是计算机怎样模拟人类的学习行为，以获取新的知识或技能，并重新组织已有的知识结构使之不断改善自身。机器学习就是计算机从数据中学习规律和模式，以应用在新数据上做预测的任务。伴随着机器学习的火热，人工智能在机器学习的推动下，快速发展，迅速席卷全球，为很多行业带来可信的深刻洞察以及充满希望的前景。

1.1 什么是机器学习

前几天我把自己最喜欢的电影《谍中谍 6》又看了一遍，这部电影实在是太精彩了，可以称为“最好看的谍战片”，《碟中谍》系列电影的前 5 部系列在豆瓣上评分都是 8 分以上。每当我在豆瓣上打开这部电影，豆瓣都会推荐我可能喜欢的其他电影。这些推荐的电影还真符合我的“口味”，比如推荐的电影《生死时速》、《飓风营救》也是我喜欢看的。那么豆瓣是如何知道用户的喜好，这些推荐背后的秘密是什么呢？

像豆瓣、淘宝、QQ 音乐这些推荐系统，背后的秘密武器正是机器学习。下面我们用最通俗易懂的大白话来聊聊，到底什么是机器学习？

机器学习有点像人类的思考过程，假设我们去买苹果，苹果是一种营养成分高的水果，国外谚语“每天一个苹果，医生远离我”。我们想挑又脆又甜的苹果，怎么挑呢？记得妈妈说过，苹果的表面要光滑，没有虫眼，没有干枯。所以我们有了一个简单的判断标准：只挑表面要光滑的。

如果用计算机程序来帮我们挑选苹果，则可写下这样的规则：

```
if (表面光滑)
then
```

```
    苹果是甜的
else
    苹果不甜
```

我们会用这些规则来挑选苹果。如果在我们的苹果实验中有了新的发现，比如在我们买回的苹果中有些是不好吃的，经过品尝各种不同类型的苹果，我们发现如果带着蒂的话那一定要选蒂比较绿的，枯黄的话就说明苹果已经不新鲜了。

所以我们修改了规则：

```
if(表面光滑  and  蒂比较绿  ):
    苹果是甜的
else:
    苹果不甜
```

我们会发现这个普通的计算机算法有个缺点，那就是：我们得搞清楚影响苹果甜度的所有因素的错综复杂的细节，比如又发现了自然熟透了的苹果都是黄里透红，苹果越重说明含水量越充足，等等。如果问题越来越复杂，我们就要针对所有的苹果类型建立规则，手动地制定挑选规则就变得非常困难。

那如何解决克服这个缺点呢？机器学习算法可以解决这个问题。机器学习算法是由前面的普通算法演化而来的。通过自动地从提供的数据中学习，它会让我们的程序变得更“聪明”。

我们从市场上的苹果里随机地抽取一定的样品（在机器学习里叫作训练数据），制作成下面的一张表格，上面记着每个苹果的物理属性，比如颜色、大小、产地等。（这些苹果的属性称之为特征）。还记录下这个苹果甜不甜（这叫作标签）。

我们将这个训练数据提供给一个机器学习算法，然后它就会学习出一个关于苹果的特征和它是否甜之间关系的模型。下次我们再去市场买苹果，面对新的苹果（测试数据），然后将新的苹果输入这个训练好的模型，模型会直接输出这个苹果是甜的，还是不甜的。有了这个模型，我们现在可以满怀自信地去买苹果了，根本不用考虑那些挑选苹果的细节。只需要将苹果的物理属性输入这个模型，就直接可以知道苹果是不是甜的。

更重要的是，我们可以让这个模型随着时间越变越好（增强学习），当这个模型读进更多的训练数据，它就会更加准确，并且在做了错误的预测之后进行自我修正。这还不是最棒的地方，最棒的地方在于，我们可以用同样的机器学习算法去训练不同的模型，比如我们可以使用同样的机器算法来预测橘子、西瓜的模型。这是常规计算机传统程序办不到的，这就是机器学习的专属优势。

总结一下，机器学习是用机器学习算法来建立模型，当有新的数据过来时，可以通过模型来进行预测。机器学习最基本的做法，是使用算法来解析数据、从中学习，然后对真实世界中的事件做出决策和预测。与传统的为解决特定任务、硬编码的软件程序不同，机器学习是用大量的数据来“训练”，通过各种算法从数据中学习如何完成任务。

1.2　机器学习的流程

1.2.1　数据收集

中国古代的“神农尝百草”的故事其实就是机器学习的思想，机器学习不是基于推理的“演绎法”，而是基于观测的“归纳法”。因为它用的是归纳法，所以数据是基础。其实人们很早就意识到了数据的重要性，比如在贴吧里，会看到铺天盖地的“求数据”。AI 需要见过各种类型和各种状态的实物图片才行，这就需要大量的基础数据。

初始的人工智能软件就像初生的小婴儿一样，是一张白纸什么都不会，大家要教他认识周围的事物，告诉他鼻子是鼻子，鼻子是用来呼吸的；告诉他嘴巴是嘴巴，嘴巴是用来说话和吃东西的。AI 也一样，我们需要向机器提供鼻子、嘴巴的图片，并将图片中的鼻子和嘴巴指出来，然后机器通过学习图片中鼻子和嘴巴的特征，并做好标注，机器就能够认知了。

大家应该都使用过上下班考勤的指纹打卡机，我们使用打卡机的时候，会先输入员工的指纹，再设置此指纹对应的员工的名字或者编号，待机器确认后，往后的日子只要输入指纹机器就都可以识别了。这便是最简单的人工智能原理：先输入信息，然后机器识别信息，再做输出。

机器学习的本质，一切都是量化的，而不是“抽象、模糊”的。原生的图片、文字等形式都是机器无法直接辨识的，都要先转化为数字、向量、矩阵。收集数据，再通过这些数据来训练人工智能模型。业界有一句非常著名的话：“数据决定了机器学习的上界，而模型和算法只是逼近这个上界。”由此可见，数据对于整个机器学习项目至关重要，有了优质数据的支持，人工智能才能更好地发展。

1.2.2　数据预处理

在工程实践中，我们得到的数据会存在缺失值、重复值等，在使用之前需要进行数据预处理。数据预处理没有标准的流程，通常针对不同的任务和数据集属性的不同而不同。数据预处理的主要常用流程为：去除唯一属性、处理缺失值、特征编码、特征缩放。

去除唯一属性是因为唯一属性通常是一些 id 属性，这些属性并不能刻画样本自身的分布规律，所以简单地删除这些属性即可。

对于缺失值处理的方法，除了不处理直接使用含有缺失值的特征以及缺失值补全，也可以删除含有缺失值的特征（该方法在包含缺失值的属性含有大量缺失值并且仅仅包含极少量有效值时才是有效的）。

特征编码是指特征必须是数值型才能统计计算，所以要对特征进行编码。比如性别特征['male', 'female']等，模型不能直接识别的数据，处理的目的是将不能够定量处理的变量量化。

为什么还要进行特征缩放？这是因为有些特征（属性）的值是有区间界限的，如年龄、体重。而有些特征的值是可以无限制增加的，所以特征与特征之间数值的差距会对模型产生不良影响（数量级的差异将导致量级较大的属性占据主导地位，依赖于样本距离的算法对于数据的数量级非常敏感）。如果没有对数据进行预处理的话有可能带来偏差，难以较好地反应特征之间的重要程度。通过归一化和标准化的手段将样本的属性缩放到某个指定的范围，消除样本不同属性具有不同量级时的影响。

1.2.3 特征工程

特征工程也被称为特征提取，为了提取知识和做出预测，机器学习使用数学模型来拟合数据，这些模型将特征作为输入。特征就是原始数据某个方面的数值表示，在机器学习流程中，特征是数据和模型之间的纽带。

特征工程是通过对原始数据的处理和加工，将原始数据属性通过处理转换为数据特征的过程。某种程度而言，好的数据以及特征往往是一个性能优秀模型的基础，它是机器学习流程中一个极其关键的环节，因为正确的特征可以减轻构建模型的难度，从而使机器学习流程输出更高质量的结果。机器学习从业者有一个共识，那就是建立机器学习流程的绝大部分时间都耗费在特征提取和数据预处理上。

1.2.4 模型构建和训练

当我们处理好数据之后，就可以选择合适的机器学习模型进行数据的训练了。可供选择的机器学习模型有很多，每个模型都有自己的适用场景，那么如何选择合适的模型呢？

首先我们要对处理好的数据进行分析判断，是考虑使用监督学习的模型，还是考虑使用无监督学习的模型。其次分析问题的类型是属于分类问题还是回归问题，当我们确定好问题的类型之后再去选择具体的模型。

在模型的实际选择时，通常会考虑尝试不同的模型对数据进行训练，然后比较输出的结果，选择最佳的那个。此外，我们还会考虑到数据集的大小。若是数据集样本较少，训练的时间较短，通常考虑朴素贝叶斯等一些轻量级的算法，否则的话就要考虑一些重量级的算法。

选好模型后是训练模型，训练模型意味着找到最合适的权重/参数，以便最大限度地分类（在分类问题中）或者预测与实际值之间的误差最小（在回归问题中）。

在验证数据上测试你的模型，使用计算和用于训练数据的相同参数来验证验证数据。一旦你尝试了不同的模型、不同的特征和不同的精度参数，并满足自己的机器学习模型的质量要求，你的学习模型就可以用于实际数据了。

1.3　机器学习该如何学

1.3.1　AI 时代首选 Python

为何人工智能（AI）首选 Python？读完这篇文章你就知道了。Python 虽然是脚本语言，但是因为容易学，迅速成为了科学家的工具（MATLAB 也能搞科学计算，但是软件要钱，而且很贵），从而积累了大量的工具库、架构。人工智能涉及大量的数据计算，用 Python 是非常自然，且简单高效。

Python 有非常多优秀的深度学习库可用，现在大部分深度学习框架都支持 Python，不用 Python 用谁？人生苦短，就用 Python 吧。Python 现在的确已经很火了，这已经是一个不需要争论的问题了。Facebook 公司开源了 PyTorch 之后，Python 作为 AI 时代头牌语言的位置基本确立了。

你只要有一门编程语言基础，如 C 或 VB，三天之内就能掌握了 Python 的使用技能。哪怕你是第一次接触编程语言，看看 Python 在线入门课程，花一周时间也会学得差不多了。因为 Python 比其他编程语言更加简单、易学，其面向对象特性甚至比 Java、C#、.NET 更加彻底。

Python 是一种解释型语言，这意味着 Python 可以节省大量的项目开发时间，因为开发者完全不需要任何类型的编译和连接。在 Python 中，开发者可以用交互方式来使用解释器，无须耗费大量的时间和精力，这才是程序员最想要的。毕竟，时间就是金钱。

关于 Python 安装也非常简单，请一定要选择 Python3 版本，将本书附赠的 Python 软件资源包复制到硬盘上并解压缩，双击资源包中的 Python-3.6.4-md64.exe 文件，它会自动安装好 Python 3.6.4 版本，安装的时候注意要在安装界面上选择“Add Python 3.6 to PATH”复选框（添加环境变量），然后再选择中间的“Install Now”选项，如图 1-1 所示。

图 1-1

在 Python 软件安装好之后，然后双击 install.bat 文件，系统会自动安装好本书会用到的一些依赖包，如大名鼎鼎 scikit-learn 机器学习工具包（本书机器学习项目案例会用到）、Dlib 人工智能工具包（本书人脸识别项目案例会用到），等等。

IDLE 是 Python 自带的程序编辑器，打开之后会出现如图 1-2 所示的界面，这个界面叫 shell。选择 File 菜单下的“New File”命令，你就可以快乐地敲入程序代码，代码输入后选择“Run”菜单中的“Run Module F5”命令，就能看到运行结果了。

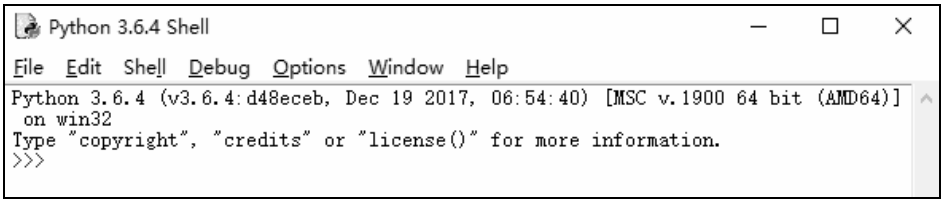

图 1-2

当你掌握 Python 基本语法之后，需要再花一点时间去学习处理数据与操作数据的方法，熟悉一下 Pandas、NumPy 和 Matplotlib 这些工具包的使用方法。Pandas 工具包可以处理数据帧，数据帧类似于 Excel 文件当中的信息表，有横行和纵列。这种数据就是所谓的结构化数据。NumPy 工具包可以基于数据进行数值运算，机器学习可以把能想象到的一切事物转化为数字，再建立出这些数字中的模型。Matplotlib 工具包可以制图，实现数据可视化。对读者来说，理解表格中的一堆数据可能很难，相信大家会更喜欢那种有线条贯穿始终的图表，实现数据可视化是交流成果的重要环节。

对于学习使用过程中，需要安装相应的工具包，推荐使用 pip 程序来安装。举例说明，比如安装 matplotlib 模块，进入到 CMD 窗口下，执行 python -m pip install matplotlib 进行自动安装，系统会自动下载安装包，如图 1-3 所示。

安装完成后，可以使用 python -m pip list 查看本机安装的所有模块，确保 Matplotlib 工具包已经安装成功，如图 1-4 所示。

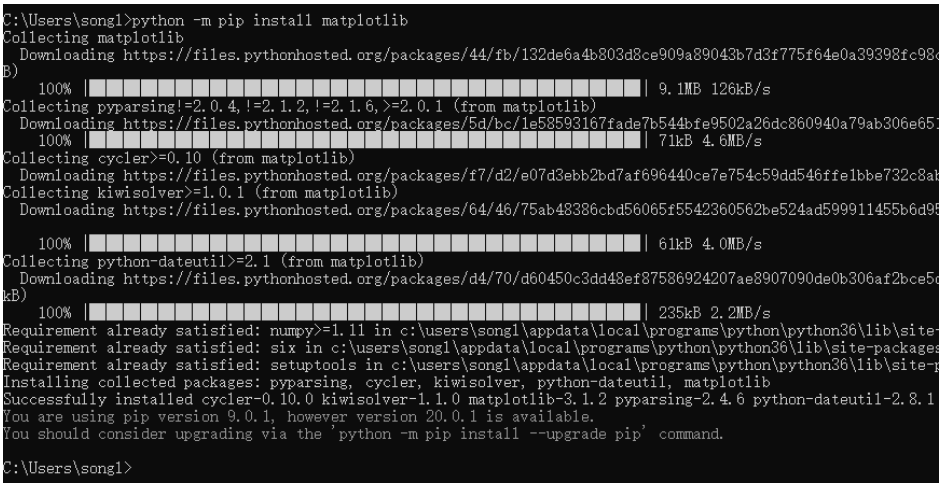

图 1-3

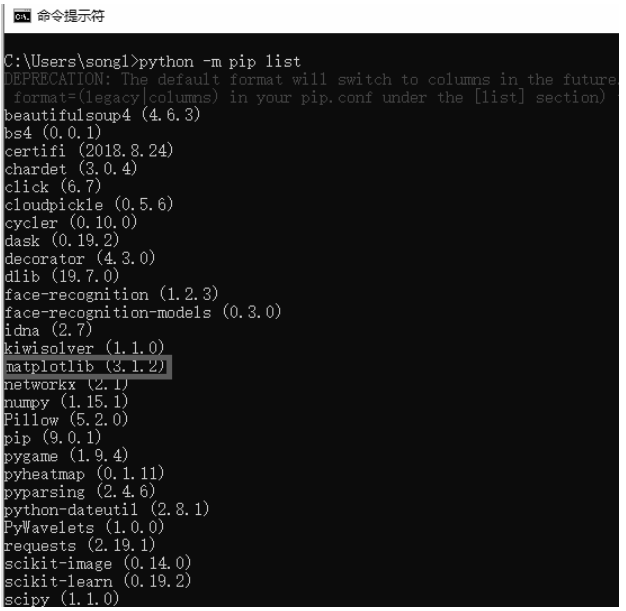

图 1-4

非常重要的机器学习工具包 scikit-learn 已经成为机器学习领域当中最知名的 Python 模块了。scikit-learn 简称 sklearn，支持包括分类、回归、降维和聚类等机器学习算法，还包含了特征提取、数据处理和模型评估三大模块。sklearn 涵盖了几乎所有主流机器学习算法的实现，它具有 BSD 许可证授权（BSD 只要求你对软件原作者的工作进行必要的认可和尊重就行了，所以这是适合商业应用的），可以将项目应用于商业开发。

sklearn 官网网址 http://scikit-learn.org/stable/index.html，里面讲解了基于 sklearn 对所有算法的实现和简单应用。在工程应用中，用 Python 手写代码来从头实现一个算法的可能性非常低，这样不仅耗时耗力，还不一定能够写出结构清晰、稳定性强的模型。更多情况下，是分析采集到的数据，根据数据特征选择适合的算法，在工具包中调用算法，调整算法的参数，获取需要的信息，从而实现算法效率和效果之间的平衡。而 sklearn 正是这样一个可以帮助我们高效实现算法应用的工具包。

总结起来，scikit-learn 工具包有以下几个优点：

（1）官方文档齐全，更新及时。

（2）针对所有的算法提供了一致的接口调用规则。

（3）涵盖主流机器学习任务的算法，包括回归算法、分类算法、聚类分析、数据降维处理等。

1.3.2　PyCharm 可视化编辑器和 Anaconda 大礼包

“工欲善其事，必先利其器”，我更喜欢安装一些功能强大的 Python 软件来辅助我编写程序。就像在学习 Java 时，正常情况选择安装 JDK，然后配置环境变量，用记事本编写程序，

再到终端编译运行即可，而我一般选择安装 JDK+MyEclipse。将 Python 和 Java 进行类比的话，在 Python 中使用 PyCharm 或 Anaconda 好比是在 Java 中使用 MyEclipse。以下分别介绍 PyCharm 和 Anaconda。

1. PyCharm 可视化编辑器介绍

PyCharm 作为一款针对 Python 的编辑器，配置简单、功能强大、使用起来省时省心，对初学者友好。PyCharm 官网提供免费的社区版与付费的专业版，如图 1-5 所示，个人学习 Python 使用免费的社区版已足够用了。安装过程照着提示一步步操作即可。

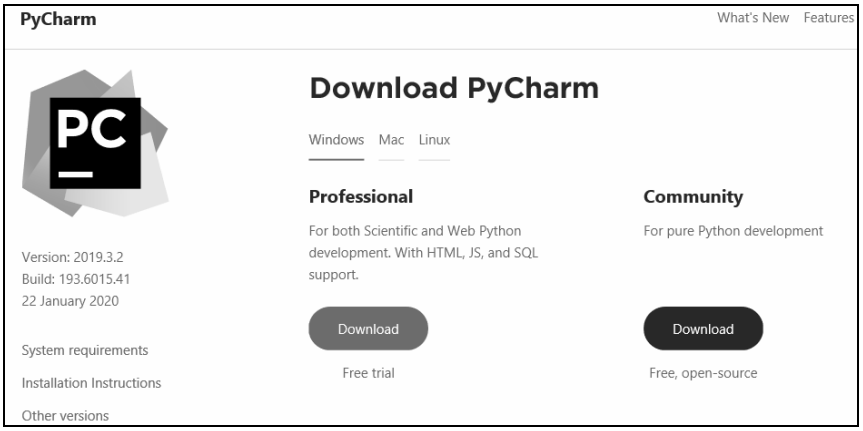

图 1-5

启动 PyCharm，选择创建新的项目“Create new Project”，选择“Existing interpreter”已经存在的 Python 解释器（之前已经安装的 Python 3.6，PyCharm 没有内置的 Python 解释器，需要我们自己下载 Python 解释器），如图 1-6 所示。

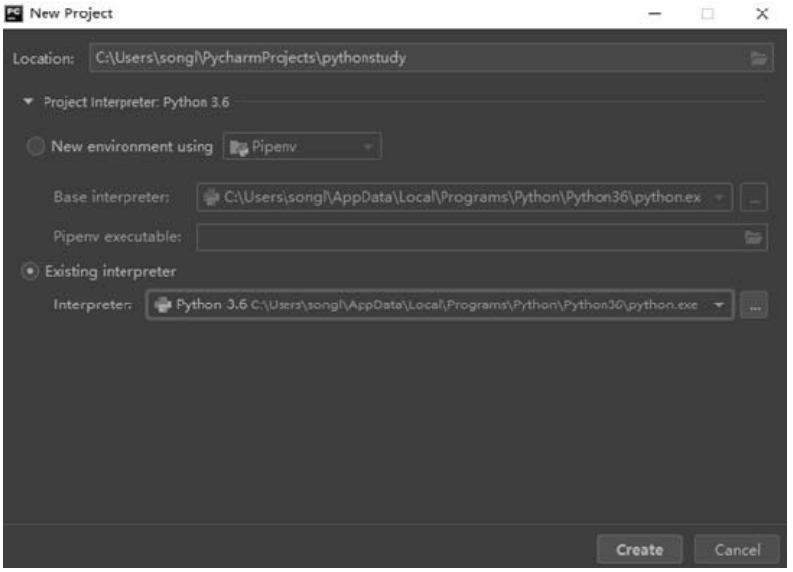

图 1-6

如图 1-7 所示，在“File”菜单下的“Settings”选项中，PyCharm 的各种配置都需要在这里配置。比如进入“Editor”→“Font”或“Color Schema”对字体以及 IDE 主题进行更改，在“Project Interpreter”中设置本地已安装的 Python 解释器。这个地方一定要注意，在选择 Python 解释器的时候，一定要选择 python.exe 这个文件，而不是 Python 的安装文件夹。

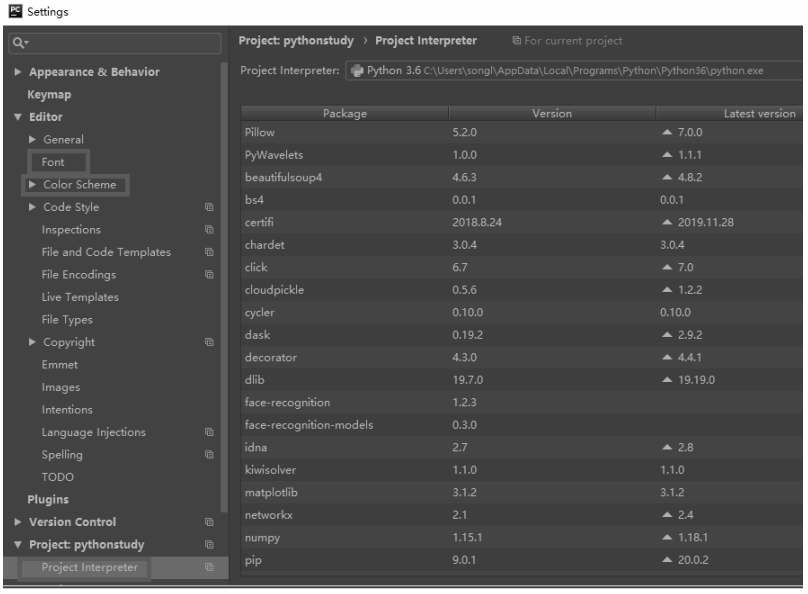

图 1-7

PyCharm 基础配置到这就完成了，接下来我们就可以开始创建 Python 脚本文件，如图 1-8 所示。

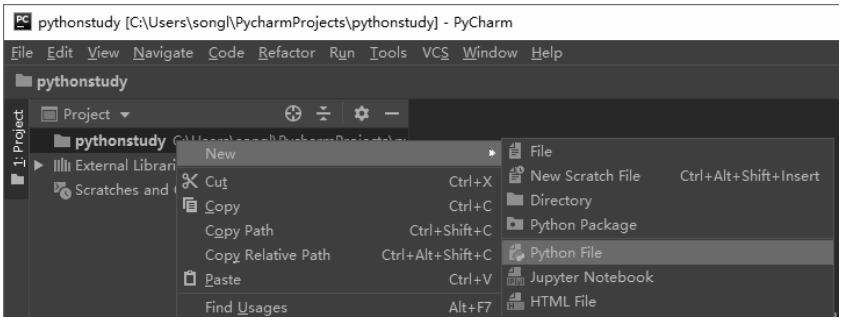

图 1-8

这里示例的 Python 脚本文件名是“example1.py”，开始敲入代码，最后在界面单击鼠标右键，选择“Run example1”，就可以看到代码的运行结果，如图 1-9 所示。

总之，PyCharm 是 JetBrains 打造的一款 Python IDE（集成开发环境）。它具备调试、语法高亮、项目管理、代码跳转、智能提示、自动完成、单元测试、版本控制等功能。可以帮助程序员节约时间，提高生产效率。

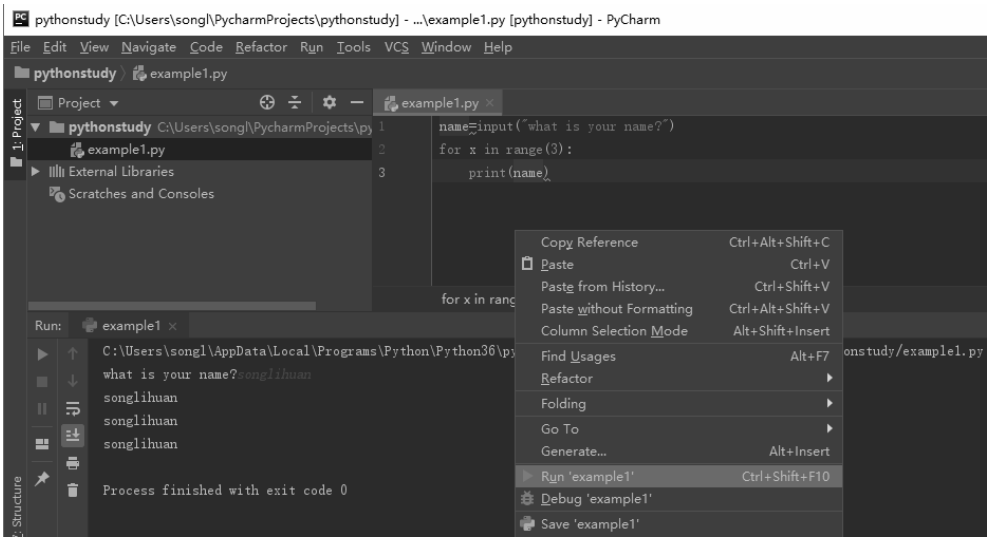

图 1-9

2. Anaconda 大礼包

Anaconda 是一个基于 Python 的数据处理和科学计算平台，相当于一个“全家桶”。它已经内置了许多非常有用的第三方库。安装好 Anaconda，就相当于把 Python 和一些如 NumPy、Pandas、Scrip、Matplotlib 等常用的库自动安装到位，使得安装比常规 Python 安装要容易。如果选择安装 Python 的话，那么还需要 pip install 逐个安装各种库，安装起来比较痛苦。

首先访问 Anaconda 官网下载对应的软件，根据自己的计算机配置选择不同的操作系统，一定要选择 Python 3 版本，下载后安装，如图 1-10 所示，非常简单。

图 1-10

安装完成后，如果是 Windows 系统，可以在“开始”菜单看到如图 1-11 所示的安装

结果。

图 1-11

可以选择“Anaconda Prompt”，打开一个命令行窗口，你可以输入“conda list”命令，可以查看目前已经安装的各种工具包库函数，如图 1-12 所示。如果需要额外安装一些其他的工具包，则可以使用“pip install”命令。如果在安装过程中报错，可以先尝试下载安装包，然后再进行安装。

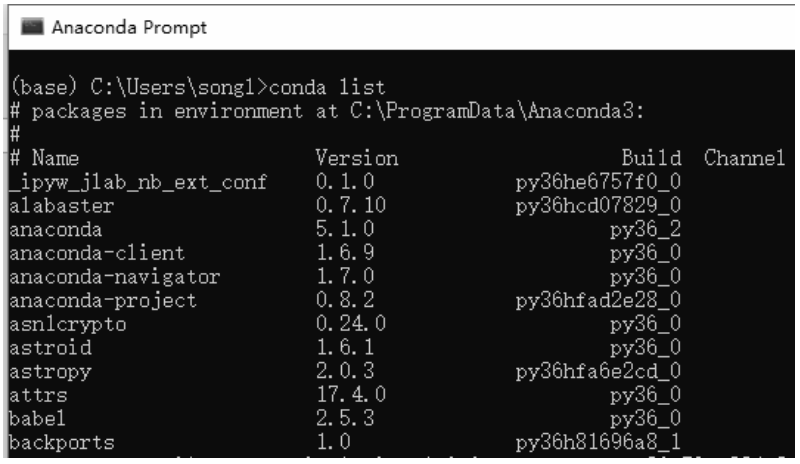

图 1-12

还有一个好用的工具是 Jupyter Notebook，Jupyter Notebook 相当于在浏览器中完成编程任务，不仅可以写代码，做笔记，而且还可以得到每一步的执行结果。它是一个在浏览器中使用的交互式的笔记本，可以将代码、文字完美地结合起来。

启动 Jupyter Notebook，它将在你的默认网页浏览器中打开一个新的标签，如图 1-13 所示。创建一份新的 Notebook，选择“New”下面的“Python 3”选项，即可进入 Python 的操作和执行窗口，在其中可以运行 Python 代码，这个过程其实很简单，因为 Python 代码都是在 cell 单元格中编写的。在 cell 单元格中编写好 Python 代码，然后单击运行，就可以直接在下面看到结果。仔细观察一下，我们可以发现，第一个 cell 前面有“In [1]:”提示符，第二个 cell 前面有“In[2]:”提示符，同时也有“Out[2]:”输出符（见图 1-14），这是因为如果没有 print 语句

的话，Notebook 会将当前 cell 的最后一条语句的结果以“Out[?]:”的方式输出。

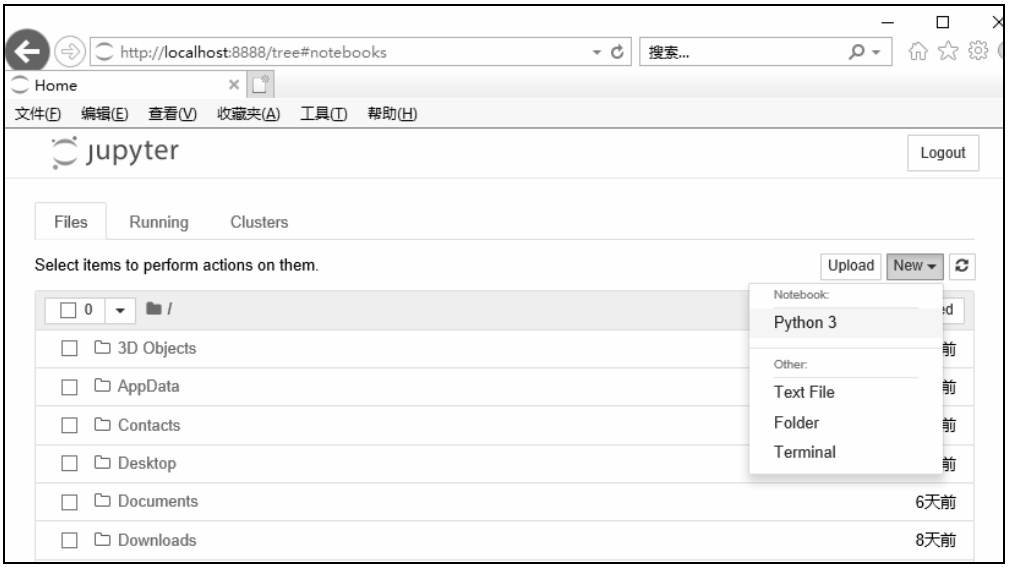

图 1-13

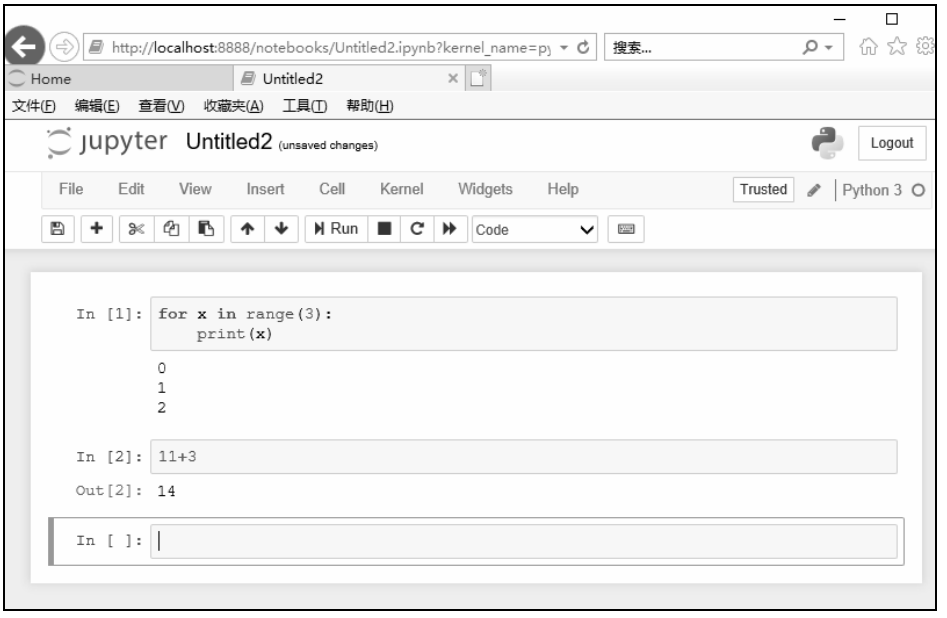

图 1-14

总的来说，Jupyter Notebook 是一个在线编辑器，可以在网页上编辑程序，在编辑的过程中，每次编辑一行代码就可以运行一行代码，运行的结果也就直接显示在代码的下方，方便查看。当所有的程序编写和运行完毕之后，还可以直接把编辑和运行之后的所有信息保存在文件中。这些工具软件的使用不需要死记硬背，在使用过程中需要什么就去查，多用用就能记住了。

1.3.3　掌握算法原理与掌握机器学习软件库同等重要

很多人会认为工具包已经很成熟了，是不是只要将相应的数据输入工具、框架中，用几行代码指定模型的类型和参数，就能自动计算出结果。既然如此，何必再去学习其中的原理，一步步推导呢？

我的意见是机器学习的原理和数学推导一定要学！此处举个直观的例子：工具就像是武器，学会使用某种工具，只是学会了这种武器最基本的招式和套路。而理论学习即策略学习，决定了在未来真实对战中，遇到对手攻击时，你选取哪些招式套路，如何组合起来去迎敌。

这里有两点建议：

- 熟悉算法原理比实现算法重要：机器学习的本质是数学和统计学的结合，搞懂这些才是王道。我们知道一个模型可能有好多参数，不同参数的设置带来的模型效果可能截然不同，我们只有弄懂了理论原理，才能更好地指导我们选择合适的参数。如果你只是实现了算法，却没有让算法发挥最大的效果，那又有什么意义呢？学习的过程是有些枯燥，本书强调机器学习极简入门，把枯燥的算法用通俗的白话演绎出来，同时又渗透高等数学知识，保持你对数据科学的兴趣和动力。
- 熟悉掌握一个机器学习软件库：成熟的软件库一般包含了绝大多数的机器学习算法，并且在实现上使用了许多的数值计算优化技巧。为了实现自己的需求，我们不必从头实现算法，即使你实现了，你的计算效率也一定没有软件库实现的高，所以我们要熟练使用一个软件库。

1.3.4　机器学习与深度学习的区别

机器学习是人工智能的一个重要的子集，深度学习又是机器学习的一个重要的子集，如图 1-15 所示。机器学习与深度学习都是需要大量数据来“喂”的，同时深度学习还需要更高的运算能力支撑，如 GPU。

图 1-15

深度学习（Deep Learning，简称 DL）属于机器学习的子类。它的灵感来源于人类大脑的工作方式，是用于建立、模拟人脑进行分析学习的神经网络，也是模仿人脑的机制来解释数据的一种机器学习技术。它的基本特点是试图模仿大脑的神经元之间传递、处理信息的模式。最显著的应用是计算机视觉和自然语言处理（NLP）领域。显然，“深度学习”与机器学习中的“神经网络”是强相关的，我们可以将“深度学习”称之为“改良版的神经网络”算法，可理解为包含多个隐藏层的神经网络结构。为了提高深层神经网络的训练效果，人们对神经元的连接方法以及激活函数等方面做出了调整。其目的在于建立、模拟人脑进行分析学习的神经网络，模仿人脑的机制来解释数据，如文本、图像、声音。

神经网络的计算量非常大，事实上在很长时间里由于基础设施技术的限制进展并不大。而 GPU 的出现让人们看到了曙光，也造就了深度学习的蓬勃发展，“深度学习”才一下子火热起来。

用机器学习的技术，开发算法来感知图像，例如识别图像是不是一个停止标志牌，但是它太容易受环境条件的干扰。如果遇到阴雨天，标志牌变得不是那么清晰可见，算法就难以成功了，但是随着时间的推移，学习算法的发展改变了一切。

人工神经网络（Artificial Neural Network）是早期机器学习中的一个重要的算法，历经数十年风风雨雨。神经网络的原理是受我们大脑的生理结构——互相交叉相连的神经元启发。例如：我们可以把一幅图像切分成图像块，输入到神经网络的第一层。在第一层的每一个神经元都把数据传递到第二层。第二层的神经元也是完成类似的工作，把数据传递到第三层，以此类推，直到最后一层，然后生成结果。每一个神经元都为它的输入分配权重，这个权重的正确与否与其执行的任务直接相关。最终的输出由这些权重加总来决定。

我们仍以识别停止（Stop）标志牌为例，用神经元进行“检查”，神经网络的任务就是给出结论，它到底是不是一个停止标志牌。神经网络会根据所有权重，给出一个经过深思熟虑的猜测。因为它还是很容易出错的。它最需要的就是几千、上万甚至几百万张图像来训练，直到神经元的输入的权值都被调制得十分精确，无论是否有雾，晴天还是雨天，每次都能得到正确的结果。只有这个时候，我们才可以说神经网络成功地自学习到一个停止标志。

深度学习（Deep Learning）加入了“深度”（Deep）。这里的“深度”就是说神经网络中众多的层。层数非常多，神经元也非常多，然后给系统输入海量的数据来训练。现在，经过深度学习训练的图像识别，在一些场景中甚至可以比人做得更好：从识别猫，到识别血液中癌症的早期成分，到识别核磁共振成像中的肿瘤。Google 的 AlphaGo 先是学会了如何下围棋，然后与自己下棋进行训练。它训练自己神经网络的方法，就是不断地与自己下棋，反复地下，永不停歇。

机器学习和深度学习都是 AI 的具体技术实现，但两者区别明显。机器学习更是一种通用型的技术，包括决策树、贝叶斯、支持向量机等算法，也包括神经网络算法。而深度学习深耕神经网络，是深度神经网络算法技术，包括深度卷积网络、深度循环网络等。

机器学习能够适应各种数据量，特别是数据量较小的场景。如果数据量迅速增加，那么深度学习的效果将更加突出，这是因为深度学习算法需要大量的数据才能完美理解。另外，深

度学习算法需要高端 GPU 在大量数据的合理时间内进行训练。这些 GPU 非常昂贵，但是如果没有它们训练深层网络来实现高性能，这在实践上就不可行。高性能的 GPU 才能够实现快速计算，在建模上才能花更少时间来分析所有的图像。相对而言，传统的机器学习算法只需要一个 CPU 就可以训练得很好，而不需要最好的硬件。

1.4　机器学习分类

根据训练数据是否有标注，机器学习问题大致划分为监督学习（Supervised Learning）和无监督学习（Unsupervised Learning）两大类。

1.4.1　监督学习

在社会中，我们在很小的时候就被大人教授这是鸟啊，那是猪啊，这个是西瓜，那个南瓜，这个可以吃，那个不能吃啊之类的，我们眼里见到的这些景物和食物就是机器学习中的输入，大人们告诉我们的结果就是输出，久而久之，当我们见得多了，大人们说得多了，我们脑中就会形成一个抽象的模型，下次在没有大人提醒的时候看见别墅或者洋楼，我们也能辨别出来这是房子，不能吃，房子本身不能飞等信息。上学的时候，老师教认字、数学公式、英语单词等等，我们在下次碰到的时候，也能区分开并识别它们。这就是监督学习，它在我们生活中无处不在。每个输入样本都有标注，这些标注就像老师的标准答案一样“监督”着学习的过程。

监督学习又大致分成两类：分类（Classification）和回归（Regression）：

- 分类问题：标注是离散值，比如用户“点击”和“不点击”。如果标注只有两个值，则称为二分类，如果标注有多个值，则称为多分类。
- 回归问题：标注是连续值，比如如果问题是预测北京市房屋的价格，价格作为标注就是一个连续值，属于回归问题。

假如你想预测一下现在的房价，这是一个数据集，横轴是房子的大小，纵轴是房价。例如你有一套 100 平方米的房子，你想知道能卖多少钱，那么机器学习算法怎么帮助你呢？它会根据数据集拟合出一个函数，让函数尽可能匹配到所有的数据。当你输入房子的大小，它就会返回给你一个目前市场上比较合理的价格。这是一个监督学习的例子，是一种回归（Regression）问题，意指要预测一个连续值的输出。例如上面的房价，给定房价的数据集，对于里面的每套房子大小数据，算法都知道其对应的正确房价。

再看一个监督学习的例子，这是一个胸部肿瘤的数据集，横轴表示肿瘤的大小，纵轴表示肿瘤是否为良性的。假如有人非常不幸，胸部长了肿瘤，对应的机器学习算法就是，根据肿瘤的尺寸，估算出一个概率，即肿瘤为良性肿瘤的概率或者恶性肿瘤的概率。当然这是一个分类（Classification）问题。分类就是要预测一个离散值的输出，这里是 0/1，也就是良性/恶性。

总结一下，在有监督的学习中，我们得到一个训练数据集，监督学习的训练集要求包括

输入输出，也可以说是特征和目标，训练集中的目标是由人标注的。数据集中的每个样本有相应的“正确答案”，即认为输入和输出之间有一个关系。我们从给定的训练数据集中学习出一个函数（模型参数），当新的数据到来时，可以根据这个函数预测结果。根据这些样本做出预测分成两类：回归问题和分类问题。回归问题举例，例如：预测房价，根据样本集拟合出一条连续曲线。分类问题举例，例如：根据肿瘤特征判断良性还是恶性，得到的是结果是“良性”或者“恶性”。

1.4.2 无监督学习

我们事先没有任何训练数据样本，需要直接对数据进行建模。比如去参观一个画展，我们对艺术一无所知，但是欣赏完很多幅作品之后，我们面对一幅新的作品之后，至少可以知道这幅作品是什么流派的吧，比如更抽象一些还是更写实一点，虽然不能很清楚地了解这幅画的含义，但是至少我们可以把它分为哪一类。

在无监督学习中，我们没有属性或者标注这个概念了。也就是所有的数据都是一样的，没有什么区别。所以在无监督学习中，我们只有一个数据集，没人告诉我们应该怎么做，我们也不知道每个数据点是什么意思。它只告诉你，这里是一个数据集，你能在其中找到某种结构或者规律吗？

基于给出的数据集，无监督学习算法可以给出不同的聚类，这就是所谓的聚类算法。举个例子，给定一组不同的个体，对于每一个不同的个体，检测它们是否拥有某个特定的基因，然后我们运行一个聚类算法，把不同的个体归入不同的类，这就是无监督学习。因为没有提前告诉我们的算法，这种基因类型具体属于哪一类的人，我们只是告诉算法这里有一堆数据，我也不知道这些数据是什么，但是你要帮我自动找到这些数据中的类型。

无监督学习使我们能够解决那些事先不知道结果的问题。训练样本没有标注，无监督学习解决的典型问题是聚类（Clustering）问题。虽然我们并不知道变量的影响，但是我们可以从数据中提取结构。我们可以根据数据中变量之间的关系对数据进行聚类，从而得出这种结构。比如对一个网站的用户进行聚类，看看这个网站用户的大致构成，分析一下每类用户群的特点是什么。

总结一下，无监督学习就是：输入数据没有被标记，也没有确定的结果。样本数据类别未知，需要根据样本间的相似性对样本集进行分类（即聚类，Clustering），试图使类内差距最小化，类间差距最大化。在实际应用中，不少情况下无法预先知道样本的标签，也就是说没有训练样本对应的类别，因而只能从原先没有标注的样本集开始学习分类。

1.4.3 强化学习

这里先解释一下强化学习这个名字。为什么叫强化学习呢？因为这个过程是不断重复、不断强化认知，英文 Reinforcement Learning 中的 Reinforcement 更准确的中文翻译就是“强化”。

这里最关键的因素是三个：智能体（Agent）、状态（State）和奖励（Reward）。对于

Agent 来说，自身具备的选择决策的能力，叫作策略。这个策略意思就是，观测到了环境现在处于什么状态，而选择做出什么动作来。现在智能体处于一个很暗的环境之中，它并不知道这个环境里面到底是什么，我们希望它能够通过和环境打交道来适应这个环境，学会做什么动作才是最好的。这个智能体不是做一次决策就完成了学习的过程。实际上，它要做的是一个序列的决策。我们怎么评判智能体策略的好坏呢？评判的形式就是，它能拿到的奖励会有多大。每一步都可能有奖励，所以评判的形式是把总的奖励加起来看看它有多大。

为了更好地认识强化学习，我们通过一个现实世界中与其类似的场景进行理解。如图 1-16 所示，使用强化学习来训练狗的一般过程。

在这种情况下，强化学习的目标是训练狗（Agent 代理）在一个环境（Environment）中完成一项任务，这里的“环境”包括狗所处的物理环境和训练者。首先，驯兽师发出一条命令或指示，狗会观察（Observation），然后狗会做出反应。如果动作接近期望的行为，训练者可能会提供奖励（Reward），如食物或玩具，否则，将不提供任何奖励或提供负面奖励。在训练开始的时候，狗可能会做出更多的随机动作（Action），比如当命令是“坐下”时，它会翻身，因为它试图将特定的观察与动作和奖励联系起来。观察和动作之间的这种关联或映射称为策略（Policy）。

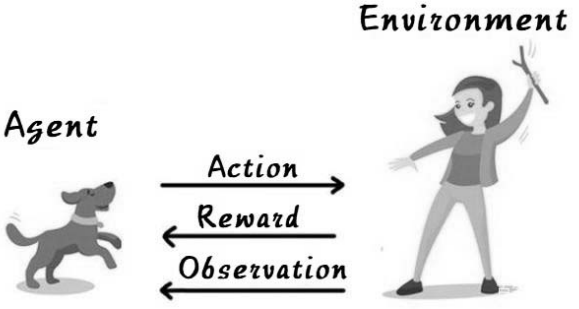

图 1-16

从狗的角度来看，最理想的情况是它能对每一个提示做出正确的反应，这样它就能得到尽可能多的奖励。因此，强化学习训练的全部意义在于“调整”狗的策略，使它学习期望的行为，从而获得最大的回报。训练完成后，狗应该能够观察主人并采取适当的行动，例如，当命令它“坐下”时，它应该使用它“发展”出的内部策略来“坐下”。

强化学习同样没有标签（Label），但是拥有回报函数（即奖励）来判断你是否更加接近目标。例如让学生搜寻某个正确答案，学生靠近正确答案，就进行奖励——比如给一个棒棒糖，如果更加偏离答案，就被扣一分，久而久之，学生会越来越靠近正确答案。

1.5　过拟合和欠拟合

无论在机器学习还是深度学习建模当中都可能会遇到两种最常见结果，一种叫过拟合

（Over-fitting）；另一种叫欠拟合（Under-fitting）。

1.5.1 过拟合

所谓过拟合（Over-fitting）其实就是所建的机器学习模型或者是深度学习模型在训练样本中表现得过于优越，导致在验证数据集以及测试数据集中表现不佳。过拟合就是学到了很多没必要的特征，比如长得像猫的狗，或者长得像狗的猫，其实这只是特例，但神经网络为了更好地降低损失（Loss），就只能被迫学习这些特征用来区分猫和狗。但是学习得太过了。举个例子：一个男人穿着蓝色的衣服，神经网络可能把是否穿蓝色衣服作为区分男人女人的特征，这就是过拟合。遇到了新样本，这些错误的特征就没有什么用了。所以过拟合就是表现为训练的时候效果很好（因为神经网络已经学到了很多有用没用的特征），但是在测试样本上的效果就很差（有的特征完全没用）。一般来说，如果训练数据集过小，特别是比模型参数数量更小时，过拟合更容易发生。

降低“过拟合”的方法如下：

（1）增加训练数据。举个例子，投硬币问题，如果你碰巧投了 10 次都是正面，那么你根据这个数据学习，是无法揭示真实规律的，根据统计学的大数定律（通俗地说，这个定理就是，在试验不变的条件下，重复试验多次，随机事件的频率近似于它的概率），当样本多了，这个真实规律是必然出现的。使用更多的训练数据是解决过拟合的最有效手段，因为更多的数据能让模型学习到更多的有效特征，减小噪声的影响。当然，直接增加实验数据一般是困难的，但是可以通过一定的规则来扩充训练数据。比如，在图像分类的问题上，可以通过图像的平移、旋转、缩放等方法来扩充数据；更进一步，可以使用生成式对抗网络（GAN）来合成大量的新数据。

（2）降低模型复杂度。在数据较少时，模型过于复杂是产生过拟合的主要因素，适当降低模型复杂度可以避免模型拟合过多的采样噪声。例如，在神经网络模型中减少网络层数、神经元个数等；在决策树模型中降低树的深度、进行剪枝等。

（3）正则化的方法。给模型的参数加上一定的正则约束。比如将权值的大小加入到损失函数中（损失函数用来评价模型的预测值和真实值不一样的程度。通常情况下，损失函数越好，模型的性能越好）。

1.5.2 欠拟合

所谓欠拟合（Under-fitting）是什么意思呢？相对过拟合，欠拟合还是比较容易理解，可能训练样本被提取的特征比较少，导致训练出来的模型不能很好地匹配，甚至样本本身都无法高效地识别。训练的模型在训练集上表现很差，在验证集上表现也很差，其本质的原因是训练的模型太简单，最通用的特征模型都没有学习到。

降低“欠拟合”的方法：

（1）添加新特征。当特征不足或者现有特征与样本特征标签的相关性不强时，模型容易出现欠拟合。

（2）增加模型复杂度。简单的模型学习能力较差，通过增加模型的复杂度可以使模型拥有更强的拟合能力。例如在线性模型中添加高次项，在网络模型中增加网络层数或神经元个数。

1.6　衡量机器学习模型的指标

对于模型性能的好坏，我们并不知道这个模型很可能就是一个差的模型（对测试集不能很好地预测或分类）。那么如何知道这个模型是好是坏呢？我们必须有个评判的标准。为了进一步了解模型的能力，我们需要用某个指标来衡量，这就是性能度量的意义。有了一个指标，我们就可以对比不同模型了，从而知道哪个模型相对好，哪个模型相对差，并通过这个指标来进一步调参以逐步优化我们的模型。

1.6.1　正确率、精确率和召回率

假设你有一台自动分类装置，可以自动检测、分类目标。为论述方便，我们就假设它是用来预测某种疾病的机器。这台机器需要用某种疾病的数据作为输入，输出只可能为两条信息之一：有病或者没有病。虽然机器的输出只有两种，但是其内部对疾病的概率估计是一个实数，比如说 p。机器上还有一个旋钮用来控制灵敏度阈值 a。因此预报过程是这样子：首先用数据计算出 p，然后比较 p 和 a 的大小，p>a 就输出有得病（检测结果为阳性），p<a 就输出没有得病（检测结果为阴性）。

如何评价这台机器的疾病预测性能呢？这里就要注意了，并不是每一次都能准确预报的机器就是好机器，因为它可以次次都预报有疾病（把 a 调得很低），自然不会漏掉，但是在绝大多数时候它都只是让大家虚惊一场，称为虚警；相反，从不产生虚警的机器也不一定就是好机器，因为它可以天天都预报没有得病（把 a 调得很高）——在绝大多数时间里这种预测显然是正确的，但也必然漏掉真正的病症，称为漏报。一台预测能力强的机器，应该同时具有低虚警和低漏报。精确率高意味着虚警少，能保证机器检测为阳性时，事件真正发生的概率高，但不能保证机器检测为阴性时，事件不发生。相反，召回率高意味着漏报少，能保证机器检测为阴性时，事件不发生的概率高，但不能保证机器检测结果为阳性时，事件就一定发生。

先介绍几个常见的模型评价术语，现在假设我们的分类目标只有两类，正例（Positive）和负例（Negative）分别是：

- True Positives（TP）：被正确地划分为正例的个数，即实际为正例且被分类器划分为正例的实例数（样本数）。
- False Positives（FP）：被错误地划分为正例的个数，即实际为负例但被分类器划分为正例的实例数。

- False Negatives（FN）：被错误地划分为负例的个数，即实际为正例但被分类器划分为负例的实例数。
- True Negatives（TN）：被正确地划分为负例的个数，即实际为负例且被分类器划分为负例的实例数。

用大白话说，真正例 TP 是指模型将正类别样本正确地预测为正类别，同样真负例 TN 是指模型将负类别样本正确地预测为负类别；假正例 FP 是指模型将负类别样本错误地预测为正类别，而假负例 FN 是指模型将正类别样本错误地预测为负类别。这四个术语的混淆矩阵，如图 1-17 所示。

正确率是我们最常见的评价指标，正确率（Accuracy）=（TP+TN）/（P+N），这个很容易理解，就是被分对的样本数除以所有的样本数。通常来说，正确率越高，分类器越好。

错误率则与正确率相反，描述被分类器错分的比例，错误率（Error Rate） =（FP+FN）/（P+N）。对某一个实例来说，分对与分错是互斥事件。

	预测类别			
		Yes	No	总计
实际类别	Yes	TP	FN	P（实际为Yes）
	No	FP	TN	N（实际为No）
	总计	P′（被分为Yes）	N′（被分为No）	P+N

图 1-17

灵敏度（Sensitive) = TP/P，表示的是所有正例中被分对的比例，它衡量了分类器对正例的识别能力。

特效度（Specificity）= TN/N，表示的是所有负例中被分对的比例，它衡量了分类器对负例的识别能力。

精确率也叫精度（Precision），是针对我们预测结果而言的，表示被分为正例的示例中实际为正例的比例，那么预测为正就有两种可能了，一种就是把正类预测为正类（TP），另一种就是把负类预测为正类（FP），精度 Precision = TP/ (TP+FP)。

召回率（Recall）是覆盖面的度量，度量有多个正例被分为正例。召回率也是针对我们原来的样本而言的，它表示的是样本中的正例有多少被预测正确了。那也有两种可能，一种是把原来的正类预测成正类（TP），另一种就是把原来的正类预测为负类（FN），召回率 Recall = TP/ (TP+FN) = TP/P = 灵敏度 Sensitive，可以看到召回率与灵敏度是一样的。

举一个例子，假设儿童有 5 万例，识别成儿童的有 4 万例，识别成 Other（其他）的有 1 例；Other 有 10 万例，识别成 Other 的有 8 万例，识别成儿童的有 2 万例。由于我们现在分析的是儿童的准确率和召回率，所以儿童是正类，Other 是负类。下面分析一下儿童的准确率和召回率：

- 检索到儿童，是儿童的数据并识别为儿童（即正类识别为正类），TP=4 万。

- 把 Other 识别成儿童（即负类识别为正类），FP=2 万。
- 未检索到儿童，是儿童却识别为 Other（即正类识别为负类），FN=1 万。
- 把 Other 识别为 Other（负类识别为负类，TN=8 万）。

精确率可以解释为，在所有判别为儿童的数据中是儿童的比例，精确度 = TP /（TP + FP）= 66.67%；召回率可解释为，在所有儿童相关的数据中，判别为儿童的比例，召回率 = TP/（TP + FN）= 80%。

一般来说，我们不可能同时提高所有上面介绍的指标，比如人们希望精确率和召回率都高，最好都是 100%，这样代表识别出来的样本确实都是正类别，且所有正类别都被识别出来了，但是现实中这两个指标往往是此消彼长的关系。提高精确率通常会降低召回率值，即使不从数学上去证明一下，从感觉上也能理解，因为这两个指标的目标刚好相反，一个要求尽可能精确，那么要抛弃掉难以决定的、把握不大的样本；而另一个指标要求尽可能识别出所有的正类别，那么就可能带入把握不大的样本。可以想象一下识别垃圾邮件，精确与识别全是难以两全的。

根据实际应用场景，比如医院检测疾病的仪器是宁愿多虚警也要少漏报，因为没病的病人如果错误地检测出有病（虚警），可以通过后续更加仔细地检查进一步排除。比如网贷违约率，相对好用户，我们更关心坏用户，不能错放过任何一个坏用户。因为如果我们过多地将坏用户当成好用户，这样后续可能发生的违约金额会远超过好用户偿还的借贷利息金额，结果得不偿失。召回率越高，代表实际坏用户被预测出来的概率越高，它的含义类似“宁可错杀一千，绝不放过一个”。再比如地震预测，没有谁能准确预测地震的发生，但我们能容忍一定程度的误报，虽然谎报了几次地震，但真的地震来临时，我们没有错过，这样才是我们想要的。

1.6.2　F1 score 和 ROC 曲线

由于存在精确率和召回率两个指标，对于多个模型来说，这两个指标差不多的情况下难以判断哪个更好，于是出现了 F1 值。假如两个模型识别样本的精确率与召回率分别是：0.6 0.6 与 0.5 0.7，那么哪个更好呢？于是数学家又定义了一个指标去计算，名叫：F score，常见的是 F1 score。

F1 score 是精确值和召回率的调和均值，它的公式如图 1-18 所示。

$$F_1 = \frac{2}{\frac{1}{recall} + \frac{1}{precision}} = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

对于极端最好的例子：精确率=1.0 召回率=1.0时, $F_1 = 2 \cdot \frac{1.0 \cdot 1.0}{1.0 + 1.0} = 1.0$

图 1-18

对于上面的两个例子，F1 score 分别是：

- precision=0.6 recall=0.6 时，F1 score=0.60。

- precision=0.5 recall=0.7 时，F1 score=0.58。

显然 precision=0.6 recall=0.6 的模型效果相对稍微好一点。

ROC 曲线

ROC 曲线起源于第二次世界大战时期雷达兵对雷达的信号判断。当时每一个雷达兵的任务就是去解析雷达的信号，但是当时的雷达技术还没有那么先进，存在很多噪声（比如一只大鸟飞过），所以每当有信号出现在雷达屏幕上，雷达兵就需要对其进行破译。有的雷达兵比较谨慎，凡是有信号过来，他都会倾向于解析成是敌军轰炸机，有的雷达兵又比较神经大条，会倾向于解析成是飞鸟。这个时候，雷达兵的上司就很头大了，他急需一套评估指标来帮助他汇总每一个雷达兵的预测信息，以及来评估这台雷达的可靠性。于是，最早的 ROC 曲线分析方法就诞生了，用来作为评估雷达可靠性的指标，在那之后，ROC 曲线就被广泛运用于医学以及机器学习领域。

ROC 的全称是 Receiver Operating Characteristic Curve，中文名字叫“受试者工作特征曲线”，顾名思义，其主要的分析方法就是画这条特征曲线，图样示例如图 1-19 所示。可以看到该曲线的横坐标为假阳性率（False Positive Rate，FPR），负例分错的概率 = FP/ (FP+TN)，俗称假警报率。纵坐标为真阳性率（True Positive Rate，TPR），正例分对的概率 = TP/ (TP+FN)，其实就是查全率（召回率 Recall），预测对的正例数占真正的正例数的比率。

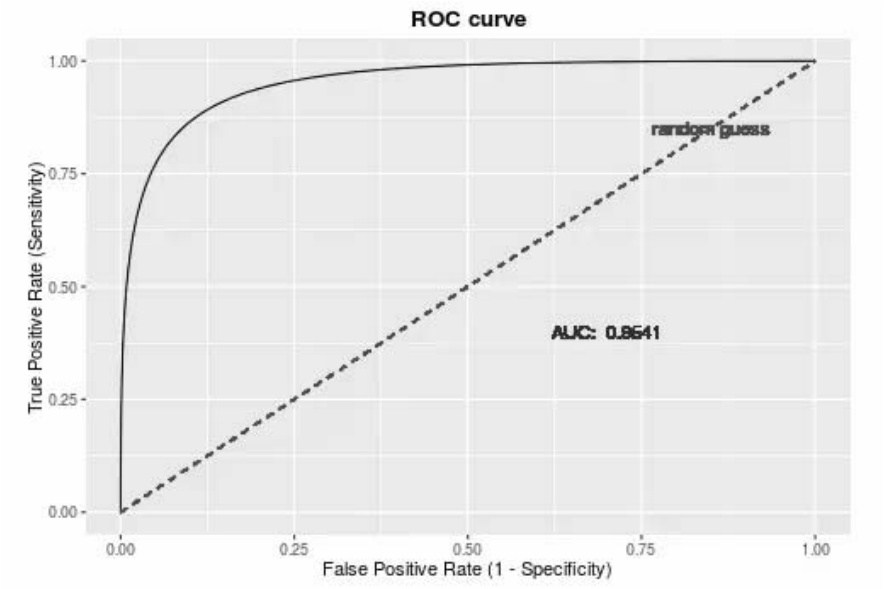

图 1-19

如果是随机分类（比如投掷硬币），没有进行任何学习器，FPR=TPR，即正例分对和负例分错概率相同，预测出来的正例负例和正例负例本身的分布是一致的，所以是一条 45° 的直线。因此，ROC 曲线越向上远离这条 45° 直线，说明用了这个学习器在很小的代价（负例分错为正例，横轴）下达到了相对较大的查全率（TPR）。

举一个简单的例子方便大家的理解，还是刚才雷达的例子。假设现在有 10 个雷达信号警报，其中 8 个是真的轰炸机（P 是真实正样本的个数）来了，2 个是老鹰（N 是真实负样本的个数）飞过，经过某分析员解析雷达的信号，判断出 9 个信号是轰炸机，剩下 1 个是老鹰，其中被判定为轰炸机的信号中，有 1 个其实是老鹰的信号（FP=1，FP 是 N 个负样本中被分类器预测为正样本的个数），而剩下 8 个确实是轰炸机信号（TP=8，TP 是 P 个正样本中被分类器预测为正样本的个数）。因此可以计算出 FPR 为 0.5，TPR 为 1，而（0.5,1）就对应 ROC 曲线上一点。对于敏锐的雷达系统来说，我们肯定希望它能把所有的敌方轰炸机来袭都感知到并预测出来，即 TPR 越高越好，但我们又不希望把飞过的老鹰也当成轰炸机来预警，即 FRP 越低越好。这两个坐标值其实蕴含了相互制约的一个概念。

当绘制完成曲线后，就会对模型有一个定性的分析。ROC 曲线的应用场景有很多，根据上述的定义，其最直观的应用就是能反映模型在选取不同阈值的时候其敏感性（Sensitivity，FPR）和其精确性（Specificity，TPR）的趋势走向。ROC 曲线还有一个巨大的优势就是，当正负样本的分布发生变化时，其形状能够基本保持不变。

第2章

机器学习中的数据预处理

数据预处理是进行数据分析的第一步，如何获取干净的数据是分析效果的前提。如果你想要你的努力获得效果（模型获得更好的预测结果），就必须对数据做预处理。

2.1 数据预处理的重要性和原则

机器学习人工智能的爱好者，往往在获得数据后，就开始疯狂地想套用一个算法模型，迫不及待地把数据往里面“喂”。当你信心满满地开始运行后，你会看到下面显示一行一行的红色字体，大体意思是这里数字无效，这时候心态就崩溃了。数据科学家在他们的工作中有50%到 80%的时间花费在收集和准备不规则数据的这种更为平凡的任务中，然后才能探索有用的价值。

在机器学习中数据是王道，较好的数据经过不同的模型训练后，其预测结果差距不是太大。在真实数据中，我们拿到的数据可能包含了大量的缺失值，可能包含大量的噪音，也可能因为人工录入错误（如医生的诊断记录）导致有异常点存在，对我们挖掘出有效信息造成了一定的困扰，所以我们需要通过一些方法，尽量提高数据的质量。在机器学习中，数据的质量关乎机器学习任务的成败、直接影响着预测的结果。

那么对于数据的预处理，有如下常用的处理原则和方法。

（1）针对数据缺失的问题，我们虽然可以将存在缺失的行直接删除，但这不是一个好办法，还很容易引发问题。因此需要一个更好的解决方案。最常用的方法是，用其所在列的均值来填充缺失。

（2）不属于同一量纲，即数据的规格不一样，不能够放在一起比较。

（3）对于某些定量数据，其包含的有效信息为区间划分，例如学习成绩，假如只关心“及

格”或“不及格”，那么需要将定量的考分，转换成“1”和“0”表示及格和不及格。二值化可以解决这一问题。

（4）大部分机器学习算法要求输入的数据必须是数字，不能是字符串。需要将描述变量转化为数字型变量，因为大部分算法无法直接处理描述变量。

（5）某些算法对数据归一化敏感，标准化可大大提高模型的精度。标准化即将样本缩放到指定的范围，标准化可消除样本间不同量级带来的影响（大数量级的特征占据主导地位；量级的差异将导致迭代收敛速度减慢;所有依赖于样本距离的算法对于数据的量级都非常敏感）。

（6）在数据集中，样本往往会有很多特征，并不是所有特征都有用，只有一些关键的特征对预测结果起决定性作用。

2.2　数据预处理方法介绍

当我们拿到一批原始的数据，步骤如下：

（1）首先要明确有多少特征，哪些是连续的特征，哪些是类别的特征？
（2）检查有没有缺失值，对确实的特征选择恰当方式进行弥补，使数据完整。
（3）对连续的数值型特征进行标准化，使得均值为 0，方差为 1。
（4）对类别型的特征进行独热编码（One-Hot Encoding）。
（5）将需要转换成类别型数据的连续型数据进行二值化。
（6）为防止过拟合或者其他原因，选择是否要将数据进行正则化。

数据预处理的工具有许多，主要有两种：Pandas 数据预处理和 scikit-learn 中的 sklearn.preprocessing 数据预处理。本章主要使用 sklearn.preprocessing 包进行数据预处理。

2.2.1　数据预处理案例——标准化、归一化、二值化

在机器学习算法实践中，我们往往有着这些需求：将不同规格的数据转换到同一规格中用，或将不同分布的数据转换到某个特定分布中，这种需求统称为将数据“无量纲化”。标准化则是将数据按照比例缩放，使之放到一个特定区间中。标准化后数据的均值＝0，标准差＝1，因而标准化的数据可正可负，只不过归一化是将数据映射到了[0,1]这个区间中。

把数据缩放到给定的范围内，通常在 0 和 1 之间，或者使用每个特征的最大绝对值按比例缩放到单位大小。在大多数机器学习算法中，会选择 StandardScaler 来进行特征缩放，因为 MinMaxScaler 对异常值非常敏感。在 PCA、聚类、逻辑回归、支持向量机、神经网络这些算法中，StandardScaler 往往是最好的选择。MinMaxScaler 在不涉及距离度量、梯度、协方差计算以及数据需要被压缩到特定区间的情况下使用广泛，比如数字图像处理中量化像素强度时，都会使用 MinMaxScaler 将数据压缩于[0,1]区间之中。

程序代码如下：

```
import numpy as np
data = np.array([[3,-1.7,3.5,-6],
                 [0,4,-0.3,2.5],
                 [1,3.5,-1.8,-4.5]])
print('原始数据：')
print(data)
from sklearn.preprocessing import StandardScaler
data_standardscaler=StandardScaler().fit_transform(data)
print('原始数据使用 StandardScaler 进行数据标准化处理后:')
print(data_standardscaler)
from sklearn.preprocessing import MinMaxScaler
data_minmaxscaler=MinMaxScaler(feature_range=(0,1)).fit_transform(data)
print('原始数据使用 MinMaxScaler 进行归一化处理（范围缩放到[0-1]）后:')
print(data_minmaxscaler)
from sklearn.preprocessing import Binarizer
data_binarizer=Binarizer().fit_transform(data)
print('原始数据使用 binarizer 进行二值化处理后:')
print(data_binarizer)
```

运行上面这段代码，结果如图 2-1 所示。

```
原始数据：
[[ 3.   -1.7  3.5 -6. ]
 [ 0.    4.   -0.3  2.5]
 [ 1.    3.5 -1.8 -4.5]]
原始数据使用StandardScaler进行数据标准化处理后:
[[ 1.33630621 -1.4097709   1.35987612 -0.89984254]
 [-1.06904497  0.80188804 -0.34370495  1.39475594]
 [-0.26726124  0.60788287 -1.01617117 -0.4949134 ]]
原始数据使用MinMaxScaler进行归一化处理（范围缩放到[0-1]）后:
[[1.          0.          1.          0.        ]
 [0.          1.          0.28301887  1.        ]
 [0.33333333  0.9122807   0.          0.17647059]]
原始数据使用binarizer进行二值化处理后:
[[1. 0. 1. 0.]
 [0. 1. 0. 1.]
 [1. 1. 0. 0.]]
```

图 2-1

StandardScaler 标准化的原理是将特征数据的分布调整成标准正态分布，也叫高斯分布，也就是使得数据的均值为 0，方差为 1，这样就可以确保数据的“大小”都是一致的，这样更有利于模型的训练。而 MinMaxScaler 把所有的数据缩放到 0 和 1 之间。除了对数据进行缩放之外，我们还可以使用 Binarizer 对数据进行二值化处理，将不同的数据全部处理为 0 或 1 这两个数值。归一化其实就是标准化的一种方式，只不过归一化是将数据映射到[0,1]这个区间中。

2.2.2 数据预处理案例——缺失值补全、标签化

很多情况下，真实的数据集中会存在缺失值，此时需要对缺失值进行处理。一种策略是将存在缺失值的整条记录直接删除。但是这样做可能会丢失一部分有价值的信息。更好的一种

方法是推定缺失数据，例如根据已有数据推算缺失的数据。SKImputer 类能够提供一些处理缺失值的基本策略，例如使用缺失值所处的一行或者一列的均值、中位数或者出现频率最高的值作为缺失数据的取值。我们得到的数据可能由于各种原因存在缺失。为了不降低机器学习模型的性能，我们可以通过一些方法处理这些数据，比如使用整列数据的平均值或中位数来替换丢失的数据。

Label Encoder 就是把 label 编码。比如 label 是一串地名，是无法直接输入到 sklearn 的分类模型里作为训练标签的，所以需要先把地名转成数字。这里 Label Encoder 就是帮你做这件事的。

范例程序代码如下：

```
import numpy as np
from sklearn.preprocessing import Imputer
print("############缺失值补全###############")
imp = Imputer(missing_values='NaN', strategy='mean', axis=0)
# 训练模型，拟合出作为替换值的均值
imp.fit([[1, 2], [np.nan, 3], [7, 6]])
x = [[np.nan, 2], [6, np.nan], [7, 6]]
print(x)
# 处理需要补全的数据
print(imp.transform(x))
print("##LabelEncoder_标准化标签，将标签值统一转换成 range(标签值个数-1)范围内#")
from sklearn import preprocessing
data=["Japan", "China", "Japan", "Korea", "China"]
print(data)
le = preprocessing.LabelEncoder()
le.fit(data)
print('标签个数:%s' % le.classes_)
print('标签值标准化:%s' % le.transform(data))
data2=["Japan", "China", "China", "Korea", "Korea"]
print(data2)
print('标签值标准化:%s' % le.transform(data2))
```

运行这个范例程序，结果如图 2-2 所示。

```
############缺失值补全###############
[[nan, 2], [6, nan], [7, 6]]
[[4.          2.        ]
 [6.          3.66666667]
 [7.          6.        ]]
###LabelEncoder 标准化标签，将标签值统一转换成range(标签值个数-1)范围内#
['Japan', 'China', 'Japan', 'Korea', 'China']
标签个数:['Japan' 'Korea' 'China']
标签值标准化:[0 2 0 1 2]
['Japan', 'China', 'China', 'Korea', 'Korea']
标签值标准化:[0 2 2 1 1]
```

图 2-2

使用 sklearn.preprocessing 库中的 Imputer 类来完成这项任务。

Imputer 参数解释：

（1）missing_values：缺失值，可以为整数或 NaN，默认为 NaN。

（2）strategy：替换策略，默认用均值“mean”替换，还可以选择中位数“median”或众数“most_frequent”。

（3）axis：指定轴数，默认 axis = 0 代表列，axis = 1 代表行。

2.2.3 数据预处理案例——独热编码

在机器学习算法中，我们经常会遇到分类特征，例如：人的性别有男、女，祖国有中国、美国、法国等。这些特征值并不是连续的，而是离散的、无序的。通常我们需要对其进行特征数字化。其中一种可能的解决方法是采用独热编码（One-Hot Encoding）。独热编码又称为一位有效编码，其方法是使用 N 位状态寄存器来对 N 个状态进行编码，每个状态都有它独立的寄存器位，并且在任意时候，其中只有一位有效。可以这样理解，对于每一个特征，如果它有 m 个可能值，那么经过独热编码后，就变成了 m 个二元特征（如成绩这个特征有好、中、差，变成独热编码就是 100, 010, 001）。并且，这些特征互斥，每次只有一个激活。因此，数据会变成稀疏的数据。

举个例子如下：

性别特征：["男"，"女"]
国家特征：["中国"，"美国，"法国"]
体育运动特征：["足球"，"篮球"，"羽毛球"，"乒乓球"]

假如某个样本（某个人），他的特征是这样的：["女","中国","羽毛球"]，我们如何将特征数字化呢？即转化为数字表示后，上述数据要能直接用在我们的分类器中，分类器往往默认数据是连续的，并且是有序的。

用独热编码来解决上述问题，做法如下：

性别特征：["男","女"]，按照 N 位状态寄存器来对 N 个状态进行编码，处理后应该是这样的（这里只有两个特征，所以 N=2）：

男 => 10
女 => 01
国家特征：["中国"，"美国，"法国"]（这里 N=3）：
中国 => 100
美国 => 010
法国 => 001
运动特征：["足球"，"篮球"，"羽毛球"，"乒乓球"]（这里 N=4）：
足球 => 1000
篮球 => 0100
羽毛球 => 0010
乒乓球 => 0001

所以，当一个样本为["女","中国","羽毛球"]的时候，完整的特征数字化的结果为：

[0，1，1，0，0，0，0，1，0]

范例程序代码如下：

```
from sklearn import preprocessing
enc = preprocessing.OneHotEncoder()
data=[[0, 0, 3], [1, 1, 0], [0, 2, 1], [1, 0, 2]]
print('数据矩阵是 4*3，即 4 个数据，3 个特征维度:')
print(data)
enc.fit(data)    # 调用 fit 来学习编码
x=[[0, 1, 3]]
print('再来看要进行编码的参数:')
print(x)
print('onehot 编码的结果:')
print(enc.transform(x).toarray())
```

这个范例程序的运行结果如图 2-3 所示。数据矩阵是 4×3，即 4 个数据，3 个特征维度。观察左边的数据矩阵，第一列为第一个特征维度，有两种取值 0\1，所以对应的编码方式为 10、01。同理，第二列为第二个特征维度，有三种取值 0\1\2，所以对应的编码方式为 100、010、001。同理，第三列为第三个特征维度，有四种取值 0\1\2\3，所以对应的编码方式为 1000、0100、0010、0001。再来看要进行编码的参数[0, 1, 3]，0 作为第一个特征编码为 10，1 作为第二个特征编码为 010，3 作为第三个特征编码为 0001，故此编码结果为[1 0 0 1 0 0 0 0 1]。

```
数据矩阵是4*3，即4个数据，3个特征维度:
[[0, 0, 3], [1, 1, 0], [0, 2, 1], [1, 0, 2]]
再来看要进行编码的参数:
[[0, 1, 3]]
onehot编码的结果:
[[1. 0. 0. 1. 0. 0. 0. 0. 1.]]
```

图 2-3

独热编码解决了分类器不好处理属性数据的问题，在一定程度上也起到了扩充特征的作用。它的值只有 0 和 1，不同的类型存储在垂直的空间中。缺点是当类别的数量很多时，特征空间会变得非常大。

独热编码用来解决类别型数据的离散值问题，将离散型特征进行独热编码的作用，是为了让距离计算更合理，但如果特征是离散的，并且不用独热编码就可以很合理地计算出距离，这样就没必要进行独热编码。有些基于树的算法在处理变量时，并不是基于向量空间度量，数值只是个类别符号，即没有偏序关系，所以不用进行独热编码。

2.2.4　通过数据预处理提高模型准确率

数据预处理的意义究竟有多大？我们使用酒的数据集来测试一下，这里使用多层神经网络模型，神经网络的概念后续章节中会详细介绍，这里只是给读者一个数据预处理对模型的准

确率的影响究竟有多大的理性认识。

范例程序代码如下：

```
# 导入红酒数据集
from sklearn.datasets import load_wine
# 导入 MLP 多层神经网络
from sklearn.neural_network import MLPClassifier
# 导入数据集拆分工具
from sklearn.model_selection import train_test_split
# 红酒数据集
wine = load_wine()
# 把数据集拆分为训练集和数据集
X_train, X_test, y_train, y_test = train_test_split(wine.data,
                                                    wine.target,
                                                    random_state=62)
print(X_train.shape, X_test.shape)
# 设定神经网络的参数
# MLP 的隐藏层为 2 个，每层有 100 个节点，最大迭代数为 400
# 指定 random_state 的数值为 62，为了重复使用模型，其训练的结果都是一致的
mlp = MLPClassifier(hidden_layer_sizes=[100,100],max_iter=400,
                    random_state=62)
# 拟合数据训练模型
mlp.fit(X_train, y_train)
# 输出模型得分
print('数据没有经过预处理模型得分:{:.2f}'.format(mlp.score(X_test, y_test)))
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X_train)
X_train_pp = scaler.transform(X_train)
X_test_pp = scaler.transform(X_test)
mlp.fit(X_train_pp, y_train)
print('数据预处理后的模型得分:{:.2f}'.format(mlp.score(X_test_pp,y_test)))
MinMaxScaler(feature_range=(0, 1), copy=True)
MaxAbsScaler(copy=True)
```

运行这个范例程序，结果如图 2-4 所示，训练集样本数目为 133，而测试集中样本数量为 45 个。我们用训练数据集来训练一个 MLP 多层神经网络（后面章节会详细介绍神经网络），在没有经过数据预处理的情况下，模型的得分只有 0.24。当对数据集进行预处理后，模型的评分大幅提高，直接提升到了 1.00。

```
(133, 13) (45, 13)
数据没有经过预处理模型得分:0.24
数据预处理后的模型得分:1.00
```

图 2-4

2.3　数据降维

2.3.1　什么叫数据降维

对一副宽为 p、高为 q 的二维灰度图，要完整表示该图像，需要 m = p*q 维的向量空间，比如 100×100 的灰度图像，它的向量空间为 100×100=10000。如图 2-5 所示，是一个 3×3 的灰度图和表示它的向量表示。

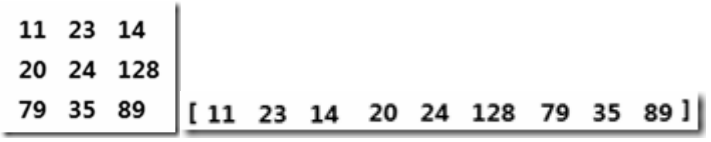

图 2-5

该向量为行向量，共 9 维，用变量表示就是[v0, v1, v2, v3, v4, v5, v6, v7, v8]，其中 v0...v8，的范围都是 0~255。现在的问题是，假如我们用 1×10000 向量表示 100×100 的灰度图，是否向量中的 10000 维对我们同样重要？肯定不是这样的，有些维的值可能对图像更有用，有些维相对来说作用小些。为了节省存储空间，我们需要对 10000 维的数据进行降维操作，降维后会尽量保留更有意义的维数，对于高维的数据集来说，一部分维数表示大部分有意义的数据。

降维就是一种对高维度特征数据预处理的方法。它将高维度的数据保留下最重要的一些特征，去除噪声和不重要的特征，从而实现提升数据处理速度的目的。在实际的生产和应用中，降维在一定的信息损失范围内，可以为我们节省大量的时间和成本。降维也成为应用非常广泛的数据预处理方法。

2.3.2　PCA 主成分分析原理

降维的算法有很多，PCA（Principal Component Analysis）即主成分分析方法，是一种使用最广泛的数据降维算法。PCA 的主要思想是将 n 维特征映射到 k 维上，这 k 维是全新的正交特征也被称为主成分，是在原有 n 维特征的基础上重新构造出来的 k 维特征。

首先看一个表格，表中是某些学生的语文、数学、物理和化学的成绩统计，如图 2-6 所示。

表 1

学生编号	语文	数学	物理	化学
1	90	140	99	100
2	90	97	88	92
3	90	110	79	83

图 2-6

先假设这些科目成绩不相关，也就是说某一科目考多少分与其他科目没有关系，那么如何判断三个学生的优秀程度呢？我们一眼就能看出来，数学、物理、化学这三门课的成绩构成了这组数据的主成分（很显然，数学作为第一主成分，因为数据成绩拉得最开）。

那么为什么我们能一眼看出来呢？当然是我们的坐标轴选对了！

下面继续看一个表格，如图 2-7 所示，图中是一组学生的数学、物理、化学、语文、历史和英语的成绩统计。

表 2

学生编号	数学	物理	化学	语文	历史	英语
1	65	61	72	84	81	79
2	77	77	76	64	70	55
3	67	63	49	65	67	57
4	80	69	75	74	74	63
5	74	70	80	84	82	74
6	78	84	75	62	72	64
7	66	71	67	52	65	57
8	77	71	57	72	86	71
9	83	100	79	41	67	50

图 2-7

那么这个表我们能一眼看出来吗？数据太多了，以至于看起来有些凌乱，无法直接看出这组数据的主成分，因为在坐标系下这组数据分布得很散乱。究其原因，是因为无法拨开遮住肉眼的迷雾，如果把这些数据在相应的空间中表示出来，也许我们就能换一个观察角度找出主成分。

PCA 主成分分析是一种用于探索高维数据的技术。PCA 可以把可能具有线性相关性的高维变量合成为线性无关的低维变量，称为主成分（Principal Component），新的低维数据集会尽可能地保留原始数据的变量，可以在将高维数据集映射到低维空间的同时，尽可能地保留更多变量。通过正交变换将一组可能存在相关性的变量转换为一组线性不相关的变量，转换后的这组变量叫作主成分。

我们如何得到这些包含最大差异性的主成分方向呢？事实上，通过计算数据矩阵的协方差矩阵，然后得到协方差矩阵的特征值特征向量，选择特征值最大（即方差最大）的 k 个特征所对应的特征向量组成的矩阵。这样就可以将数据矩阵转换到新的空间当中，实现数据特征的降维。

由于得到协方差矩阵的特征值特征向量有两种方法，特征值分解协方差矩阵和奇异值分解协方差矩阵，所以 PCA 算法有两种实现方法：基于特征值分解协方差矩阵实现 PCA 算法和基于 SVD 分解协方差矩阵实现 PCA 算法。

2.3.3　PCA 主成分分析实战案例

还是以红酒数据集为例子，演示 PCA 主成分分析的使用。

范例程序代码如下：

```
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.datasets import load_wine
wine=load_wine()
x=wine.data
y=wine.target
print("红酒数据集的数据结构:")
print(wine.data.shape)
print("红酒数据集的特征:")
print(wine.feature_names)
print("红酒数据集的标签:")
print(wine.target_names)
import pandas as pd
sample=pd.concat([pd.DataFrame(x),pd.DataFrame(y)],axis=1)
print("将样本数据和标签连接起来，展示表格前 5 行数据")
print(sample.head())
# 标准化
from sklearn.preprocessing import StandardScaler
stdsc = StandardScaler()
X_train_std = stdsc.fit_transform(x)
print('打印做标准化处理后的数据形态')
print(X_train_std.shape)
# 导入 PCA 模块
from sklearn.decomposition import PCA
# 设置降维后主成分数目为 2
pca = PCA(n_components=2)
reduced_x = pca.fit_transform(X_train_std)
print('打印主成分提取后的数据形态')
print(reduced_x.shape)

# 经过 PCA 降维的数据可视化情况
red_x,red_y=[],[]
blue_x,blue_y=[],[]
green_x,green_y=[],[]
for i in range(len(reduced_x)):
    if y[i] ==0:
        red_x.append(reduced_x[i][0])
        red_y.append(reduced_x[i][1])
    elif y[i]==1:
        blue_x.append(reduced_x[i][0])
        blue_y.append(reduced_x[i][1])
    else:
        green_x.append(reduced_x[i][0])
```

```
        green_y.append(reduced_x[i][1])
plt.scatter(red_x,red_y,c='r',marker='x')
plt.scatter(blue_x,blue_y,c='b',marker='D')
plt.scatter(green_x,green_y,c='g',marker='.')
plt.title("wine-standard-PCA")
plt.xlabel('Dimension1')
plt.ylabel('Dimension2')
plt.legend(wine.target_names,loc='best')
plt.show()
# 使用主成分绘制热度图
plt.matshow(pca.components_,cmap='plasma')
# 纵轴为主成分数
plt.yticks([0,1],['Dimension1','Dimension2'])
plt.colorbar()
# 横轴为原始特征数量
plt.xticks(range(len(wine.feature_names)),wine.feature_names,rotation=60,ha='l
    eft')
plt.show()
```

上面的范例程序的运行结果如图 2-8 所示。数据集样本数量 178，特征数量 13 个，三个类别。PCA 降维后，样本数量还是 178 个，特征数量只剩下 2 个了。PCA 主成分分析方法将数据集的特征向量降至二维，从而可以很方便地进行可视化处理，而又不会丢失太多的信息。

```
红酒数据集的数据结构:
(178, 13)
红酒数据集的特征:
['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium', 'total_phenols',
'flavanoids', 'nonflavanoid_phenols', 'proanthocyanins', 'color_intensity', 'hue',
'od280/od315_of_diluted_wines', 'proline']
红酒数据集的标签:
['class_0' 'class_1' 'class_2']
将样本数据和标签，连接起来，展示表格头5行数据
      0     1     2     3      4     5   ...    8     9     10    11      12  0
0  14.23  1.71  2.43  15.6  127.0  2.80  ...  2.29  5.64  1.04  3.92  1065.0   0
1  13.20  1.78  2.14  11.2  100.0  2.65  ...  1.28  4.38  1.05  3.40  1050.0   0
2  13.16  2.36  2.67  18.6  101.0  2.80  ...  2.81  5.68  1.03  3.17  1185.0   0
3  14.37  1.95  2.50  16.8  113.0  3.85  ...  2.18  7.80  0.86  3.45  1480.0   0
4  13.24  2.59  2.87  21.0  118.0  2.80  ...  1.82  4.32  1.04  2.93   735.0   0

[5 rows x 14 columns]
打印做标准化处理后的数据形态
(178, 13)
打印主成分提取后的数据形态
(178, 2)
```

图 2-8

如图 2-9 所示，经过降维的酒数据集，可以看出，降维后的数据仍能够清晰地分成三类。这样不仅能削减数据的维度，降低分类任务的工作量，还能保证分类的质量。

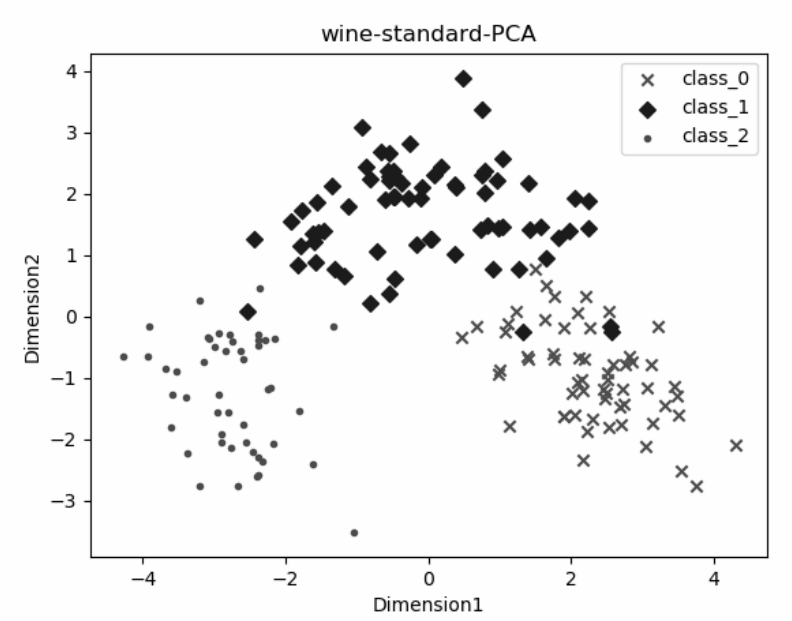

图 2-9

而主成分与各个特征值之间的关系，如图 2-10 所示。颜色由深到浅代表从-0.5 到 0.4 的数值，如果某个特征对应的数字是正数，说明它与主成分之间是正相关的关系；如果是负数，说明它与主成分之间是负相关的关系。

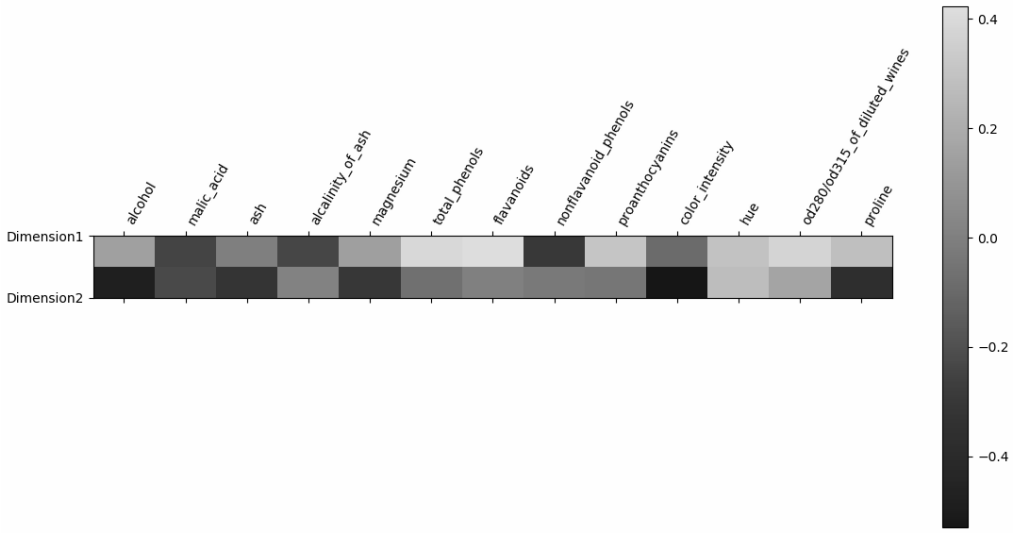

图 2-10

第 3 章

k 最近邻算法

俗话说“近朱者赤，近墨者黑”，想象一下我们的数据集里面有一半是“朱”，一半是“墨”，现在有了一个新数据点，我们怎么判断它属于哪一个分类？基于一个假设前提“属性越接近的人，行为偏好也越相似”。k 最近邻算法（k-Nearest Neighbors，简称 kNN）的原理简单地说，就是新数据点离谁最近，就和谁属于同一类。kNN 是非常简单的算法，也是新手入门机器学习的常用算法。

3.1 k 最近邻算法的原理

如图 3-1 所示，有两类不同的样本数据，分别用小正方形和小三角形表示，而图正中间的那个圆所标示的数据则是待分类的数据。这也就是我们的目的，来了一个新的数据点，我要得到它的类别是什么？我们根据 k 最近邻的思想来给中间的那个圆点进行分类。

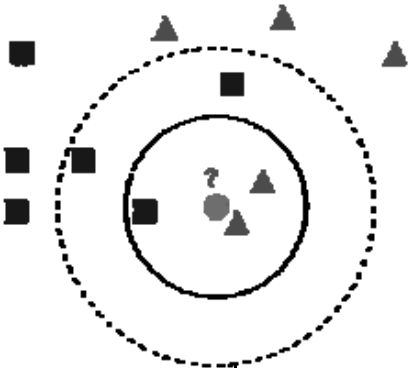

图 3-1

如果 k=3，圆点的最邻近的 3 个点是 2 个小三角形和 1 个小正方形，少数从属于多数，基于统计的方法，判定圆点的这个待分类点属于三角形一类。

如果 k=5，圆点的最邻近的 5 个邻居是 2 个三角形和 3 个正方形，还是少数从属于多数，基于统计的方法，判断圆点的这个待分类点属于正方形一类。

从上面的例子可以看出，根据 k 最近邻的算法思想如何给新来的点进行归类，只要找到离它最近的 k 个实例，哪个类别最多即可。简单来说，kNN 可以看成：有那么一堆你已经知道分类的数据，当一个新数据进入的时候，就开始跟训练数据里的每个点求距离，然后挑离这个训练数据最近的 k 个点看看这几个点属于什么类型，最后用少数服从多数的原则，给新数据归类。

kNN 算法在确定分类决策上只依据最邻近的一个或者几个样本的类别来决定待分样本所属的类别。

kNN 算法的核心思想是：如果一个数据在特征空间中最相邻的 k 个数据中的大多数属于某一个类别，则该样本也属于这个类别（类似投票），并具有这个类别上样本的特性。通俗地说，对于给定的测试样本和基于某种度量距离的方式，通过最靠近的 k 个训练样本来预测当前样本的分类结果。

总结一下 kNN 的工作原理，kNN 算法就是根据距离待分类样本 A 最近的 k 个样本数据的分类来预测 A 可能属于的类别，基本的计算步骤如下：

（1）存在一个样本数据集合，也称为训练样本集，并且样本集中每个数据都存在标签，即我们知道样本集中每一个数据与所属分类的对应关系。

（2）计算待分类数据与训练集中每一个样本之间的距离。

（3）找出与待分类样本距离最近的 k 个样本。

（4）观测这 k 个样本的分类情况。

（5）把出现次数最多的类别作为待分类数据的类别。

3.2　k 最近邻算法过程详解

举个简单的例子，我们可以使用 k 最近邻算法分类一个电影是爱情片还是动作片。我们以接吻次数和打斗次数来区分是爱情电影还是动作电影。我们已有的数据集，这个数据集有两个特征，即打斗镜头数和接吻镜头数。除此之外，我们也知道每个电影的所属类型，即分类标签，现在要预测电影 5 是属于哪个类型的电影（爱情片或动作片），如表 3-1 所示。

表 3-1　每部电影的打斗镜头数、接吻镜头数以及电影类型

电影名称	打斗镜头	接吻镜头	电影类型
电影 1	1	101	爱情片
电影 2	5	89	爱情片
电影 3	108	5	动作片

（续表）

电影名称	打斗镜头	接吻镜头	电影类型
电影 4	115	8	动作片
电影 5	101	20	预测属于哪个类型

用经验法则粗略地观察：接吻镜头多的是爱情片；而打斗镜头多的是动作片。如果现在给我一部电影，你告诉我这个电影打斗镜头数和接吻镜头数，不告诉我这个电影类型，我可以根据你给我的信息，按经验法则来判断这个电影是属于爱情片还是动作片。而 k 最近邻算法也可以像我们人一样做到这一点，但 k 最邻近算法是靠已有的数据来预测的。比如，你告诉我这个电影打斗镜头数为 101，接吻镜头数为 20，k 最近邻算法会提取样本集中特征最相似数据（最邻近）的分类标签，得到的结果取决于数据集的大小以及最近邻的判断标准因素。

我们已经知道 k 近邻算法根据特征比较，然后提取样本集中特征最相似数据（最邻近）的分类标签。那么，如何进行比较呢？如图 3-2 所示，我们怎么判断图中坐标为（101, 20）的圆点标记，它的电影所属的类别呢？

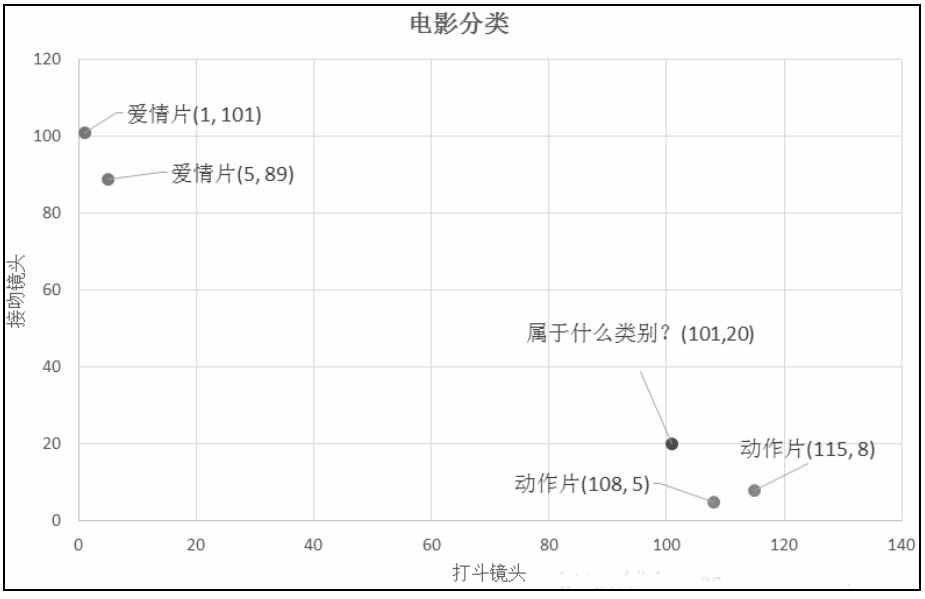

图 3-2

我们可以从散点图大致推断，这个坐标为（101, 20）的圆点标记的电影可能属于动作片，因为距离已知的那两个动作片的圆点更近。

k 最近邻算法用什么方法进行判断呢？k 最近邻算法的步骤描述如下：

（1）计算已知类别数据集中的点与当前测试点之间的距离。

（2）按照距离递增次序排序。

（3）选取与当前测试点距离最小的 k 个点。

（4）确定前 k 个点所在类别的出现频率。

（5）返回前 k 个点所出现频率最高的类别作为当前测试点的预测分类。

关于这里的距离度量，这个电影分类的例子有 2 个特征，也就是 2 维的，可以使用我们高中学过的两点距离公式（两点距离公式其实就是欧氏距离在二维空间上的公式，也就是欧氏距离 n 的值为 2 的情况，欧氏距离也称为欧几里德距离）来计算距离，如图 3-3 所示。

$$|AB| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

图 3-3

这样我们得到了样本集中所有电影与未知电影的距离，按照距离递增排序，可以找到 k 个距离最近的电影。假定 k=3，则三个最靠近的电影分别是动作片电影 3（108,5）、动作片电影 4（115,8）、爱情片电影 2（5,89）。在这三个点中，动作片出现的频率为三分之二，爱情片出现的频率为三分之一，所以坐标为（101, 20）的圆点标记的电影类型为动作片。这个判别过程就是 k 最近邻算法。

3.3　kNN 算法的注意事项

3.3.1　k 近邻的 k 值该如何选取

如果当 k 的取值过小时，一旦有噪声的成分存在，将会对预测产生比较大的影响，例如取 k 值为 1 时，一旦最近的一个点是噪声，那么就会出现偏差，k 值的减小就意味着整体模型变得复杂，容易发生过拟合（在训练集上准确率非常高，而在测试集上准确率低），而忽略了数据真实的分布。

如果 k 的值取得过大时，就相当于用较大邻域中的训练实例进行预测，学习的近似误差会增大，这时与输入目标点较远的实例也会对预测起作用，使预测发生错误。k 值的增大就意味着整体的模型变得简单，比如如果 k=N（N 为训练样本的个数），那么无论输入实例是什么，都将简单地预测它属于在训练实例中最多的类。这时相当于你压根就没有训练模型，直接拿训练数据统计了一下各个数据的类别，再找最大的类别而已！

所以说 k 值既不能过大，也不能过小（也就是说，选取 k 值的关键是实验调参）。k 的取值尽量要取奇数，以保证在计算结果最后会产生一个较多的类别，如果取偶数则可能会产生相等的情况，不利于预测。

3.3.2　距离的度量

k 最近邻算法是在训练数据集中找到与该实例最邻近的 k 个实例，这 k 个实例的多数属于某个类，我们就说预测点属于哪个类。定义中所说的最邻近是如何度量呢？我们怎么知道谁跟测试点最邻近。这里就会引出我们几种度量两个点之间距离的标准。

一般可以有以下几种度量方式：

（1）欧式距离

欧式距离其实就是应用勾股定理计算两个点的直线距离。二维空间的公式，如图 3-4 所示，P 为点(x2,y2)与点(x1,y1)之间的欧氏距离，|x|为点(x2,y2)到原点的欧式距离。

$$\rho = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

$$|X| = \sqrt{x_2^2 + y_2^2}$$

图 3-4

同理，N 维空间的公式，如图 3-5 所示。

$$d(x,y) := \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2} = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

图 3-5

（2）曼哈顿距离

曼哈顿距离就是表示两个点在标准坐标系上的绝对轴距之和。如图 3-6 所示，折线 A 代表曼哈顿距离，线段 C 代表欧氏距离，也就是直线距离，而折现 B 和折现 D 代表等价的曼哈顿距离。曼哈顿距离就是两个点在南北方向上的距离加上在东西方向上的距离，即 $d(i,j)=|x_i-x_j|+|y_i-y_j|$。

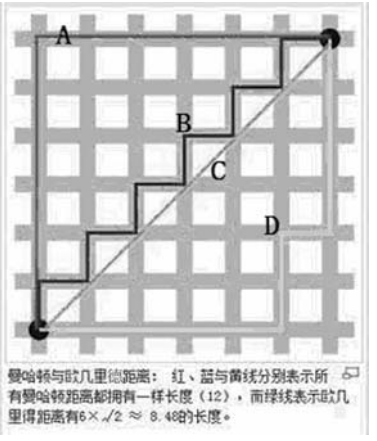

图 3-6

对于一个具有正南正北、正东正西方向规则布局的城镇街道，从一点到达另一点的距离正是在南北方向上旅行的距离加上在东西方向上旅行的距离，因此，曼哈顿距离又称为出租车距离。

（3）切比雪夫距离

切比雪夫距离得名自俄罗斯数学家切比雪夫。如图 3-7 所示，若将国际象棋棋盘放在二维直角坐标系中，格子的边长定义为 1，坐标的 x 轴及 y 轴和棋盘方格平行，原点恰落在某一格

的中心点，则王从一个位置走到其他位置需要的步数恰为二个位置的切比雪夫距离，因此切比雪夫距离也称为棋盘距离。例如位置 f6 和位置 e2 的切比雪夫距离为 4。任何一个不在棋盘边缘的位置，和周围八个位置的切比雪夫距离都是 1。

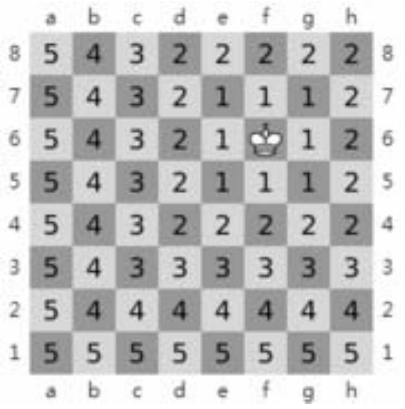

图 3-7

国际象棋棋盘上 2 个位置间的切比雪夫距离是指王要从一个位置移至另一个位置需要走的步数。由于王可以往斜前或斜后方向移动一格，因此可以较有效率地到达目的格子。

3.3.3　特征归一化的必要性

k 最近邻算法（kNN），这个最近邻的定义是通过距离函数来定义的，我们最常用的是欧式距离。为了保证每个特征同等重要性，我们这里对每个特征进行归一化。对数据进行归一化，避免量纲对计算距离的影响。

举例如下，我们用一个人身高（cm）与脚码（尺码）大小来作为特征值，类别为男性或者女性。如果现在有 5 个训练样本，分布如下：

A [(179,42),男]　B [(178,43),男]　C [(165,36)女]　D [(177,42),男]　E [(160,35),女]

通过上述训练样本，我们看出了问题吗？

很容易看到第一维身高特征是第二维脚码特征的 4 倍左右，那么在进行距离度量的时候，我们就会偏向于第一维特征。这样会造成两个特征并不是等价重要的，最终可能会导致距离计算错误，从而导致预测错误。举例如下：

现在给出一个测试样本 F(167,43)，让我们来预测一下他是男性还是女性，我们采取 k=3 来预测。

下面我们用欧式距离分别算出 F 离训练样本的欧式距离，然后选取最近的 3 个，多数类别就是我们最终的结果，计算如图 3-8 所示。

由计算可以得到，最近的前三个分别是 C、D、E 三个样本，那么由 C、E 为女性，D 为男性，女性多于男性得到我们要预测的结果为女性。

$$AF=\sqrt{(167-179)^2+(43-42)^2}=\sqrt{145}$$
$$BF=\sqrt{(167-178)^2+(43-43)^2}=\sqrt{121}$$
$$CF=\sqrt{(167-165)^2+(43-36)^2}=\sqrt{53}$$
$$DF=\sqrt{(167-177)^2+(43-42)^2}=\sqrt{101}$$
$$EF=\sqrt{(167-160)^2+(43-35)^2}=\sqrt{103}$$

图 3-8

这样问题就来了，一个女性的脚 43 码的可能性，远远小于男性脚 43 码的可能性，那么为什么算法还是会预测 F 为女性呢？那是因为由于各个特征量纲的不同，在这里导致了身高的重要性已经远远大于脚码了，这是不客观的。所以我们应该让每个特征都具有同等的重要性！这也是我们要归一化的原因！

3.4 k 最近邻算法案例分享

3.4.1 电影分类 kNN 算法实战

一个电影分类演示例子（电影名称与分类来自于优酷网；镜头数量则纯属虚构），为已知的电影分类，分为喜剧片、动作片、爱情片三个种类，使用的特征值分别为搞笑镜头、打斗镜头、拥抱镜头的数量。那么来了一部新电影《唐人街探案》，用 kNN 算法猜测它属于上述 3 个电影分类中的哪个类型？

kNN 算法分类是一种监督式的分类方法，首先根据已标记的数据对模型进行训练，然后根据模型对新的数据点进行预测，预测新数据点的标签（Label），也就是该数据所属的分类。首先我们构建一个已分好类的数据集，如图 3-9 所示，这里的新样本就是："唐人街探案": [23, 3, 17, "？片"]。

电影名称	搞笑镜头	拥抱镜头	打斗镜头	电影类型
宝贝当家	45	2	9	喜剧片
美人鱼	21	17	5	喜剧片
澳门风云3	54	9	11	喜剧片
功夫熊猫3	39	0	31	喜剧片
谍影重重	5	2	57	动作片
叶问3	3	2	65	动作片
伦敦陷落	2	3	55	动作片
我的特工爷爷	6	4	21	动作片
奔爱	7	46	4	爱情片
夜孔雀	9	39	8	爱情片
代理情人	9	38	2	爱情片
新步步惊心	8	34	17	爱情片
唐人街探案	23	3	17	？

图 3-9

Python 程序代码和注释如下：

```
#########################################################################
import math
# 为了方便验证，使用 Python 的字典 dict 构造数据集。
movie_data = {"宝贝当家": [45, 2, 9, "喜剧片"],
              "美人鱼": [21, 17, 5, "喜剧片"],
              "澳门风云 3": [54, 9, 11, "喜剧片"],
              "功夫熊猫 3": [39, 0, 31, "喜剧片"],
              "谍影重重": [5, 2, 57, "动作片"],
              "叶问 3": [3, 2, 65, "动作片"],
              "伦敦陷落": [2, 3, 55, "动作片"],
              "我的特工爷爷": [6, 4, 21, "动作片"],
              "奔爱": [7, 46, 4, "爱情片"],
              "夜孔雀": [9, 39, 8, "爱情片"],
              "代理情人": [9, 38, 2, "爱情片"],
              "新步步惊心": [8, 34, 17, "爱情片"]}
# 测试样本：唐人街探案": [23, 3, 17, "? 片"]
# 下面为求与数据集中所有数据的距离代码，欧式距离是一个简单又最常用的距离计算方法。
x = [23, 3, 17]
KNN = []
for key, v in movie_data.items():
    d = math.sqrt((x[0] - v[0]) ** 2 + (x[1] - v[1]) ** 2 + (x[2] - v[2]) ** 2)
    KNN.append([key, round(d, 2)])
# 输出所有电影到“唐人街探案”的距离
print(KNN)
# 按照距离大小进行递增排序
KNN.sort(key=lambda dis: dis[1])
# 选取距离最小的 k 个样本，这里取 k=5
KNN=KNN[:5]
print(KNN)
# 确定前 k 个样本所在类别出现的频率，并输出出现频率最高的类别
labels = {"喜剧片":0,"动作片":0,"爱情片":0}
for s in KNN:
    label = movie_data[s[0]]
    labels[label[3]] += 1
labels =sorted(labels.items(),key=lambda l: l[1],reverse=True)
print(labels,labels[0][0],sep='\n')
#########################################################
```

这个程序的输出结果如图 3-10 所示，告诉我们是喜剧片。

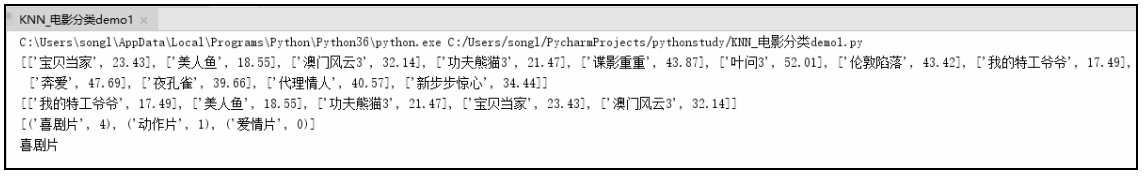

图 3-10

根据 k 最近邻算法，这里距离的计算值，用欧式距离计算方法，比如《伦敦陷落》电影

和《唐人街探案》电影（x 为："唐人街探案":[23, 3, 17, "？片"]，y 为："伦敦陷落": [2, 3, 55, "动作片"]）两者之间的距离，如图 3-11 所示，计算距离为 43.42。

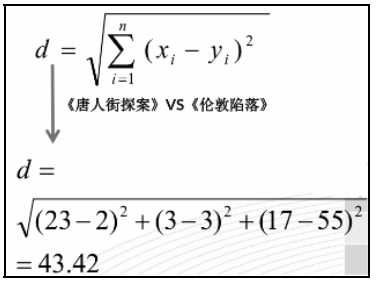

图 3-11

这里 k=5，取距离最小的 5 个样本，如图 3-12 所示，分别是：[['我的特工爷爷', 17.49]、 ['美人鱼', 18.55]、['功夫熊猫 3', 21.47]、['宝贝当家', 23.43]、['新步步惊心',34.44]]。确定前 5 个样本所在类别出现的频率，发现还是喜剧片居多，根据 k 最近邻算法思想，距离越近，就越相似，属于这一类的可能性就越大，说明《唐人街探案》电影属于喜剧片的概率比较大。

所以 kNN 算法的指导思想就是由你的邻居来推断出你的类别，总结九个字“算距离、找邻居、算距离、做分类”。“算距离”是指给定测试对象，计算它与训练集中的每个对象的距离）。“找邻居”是指圈定距离最近的 k 个训练对象，作为测试对象的近邻；“做分类”是指根据这 k 个近邻归属的主要类别，来对测试对象进行分类，少数服从多数，近邻中哪个类别的点最多就分为该类。

电影名称	搞笑镜头	拥抱镜头	打斗镜头	电影类型	距离	K=5时
我的特工爷爷	6	4	21	喜剧片	17.49	√
叶问3	3	2	65	动作片	52.01	
伦敦陷落	2	3	55	动作片	43.42	
代理情人	9	38	2	爱情片	40.57	
新步步惊心	8	34	17	爱情片	34.44	√
谍影重重	5	2	57	动作片	43.87	
功夫熊猫	39	0	31	喜剧片	21.47	√
美人鱼	21	17	5	喜剧片	18.55	√
宝贝当家	45	2	9	喜剧片	23.43	√
唐人街探案	23	3	17	?	——	?

图 3-12

3.4.2 使用 scikit-learn 机器学习库内置的 kNN 算法实现水果识别器

scikit-learn 是最受欢迎的机器学习库之一，它提供了各种主流的机器学习算法的 API 接口供使用者调用，让使用者可以方便快捷地搭建一些机器学习模型，并且通过调参可以达到很高的准确率。基于前文所说的 k 最近邻原理，我们这个案例主要是利用 scikit-learn 工具包实现机器学习，从简单的 kNN 开始入手。首先我们需要安装 sklearn 这个 Python 的扩展库，同时由于 sklearn 基于 NumPy 和 SciPy 这两个库，我们也要事先安装好它们，然后再使用“pip install

scikit-learn”命令安装 sklearn，安装好 sklearn 之后，便可以在 Python 脚本中使用来自 sklearn 中的各种数据、功能函数，等等。

数据源下载地址：https://video.mugglecode.com/fruit_data.csv，文件 fruit_data.csv 中包含了 59 个水果的数据样本，共 5 列数据：

- fruit_name：水果类别
- mass：水果质量
- width：水果的宽度
- height：水果的高度
- color_score：水果的颜色数值，范围 0-1。
  - 0.85 - 1.00：红色
  - 0.75 - 0.85：橙色
  - 0.65 - 0.75：黄色
  - 0.45 - 0.65：绿色

这里需要利用 Pandas 模块中的 map()方法进行字符串到数字的映射转换来处理样本的字符串标签。调用 scikit-learn 模块中的 KNeighborsClassifier()建立 kNN 模型，调用 scikit-learn 模块中的 fit()方法训练模型，调用 scikit-learn 模块中的 score()方法验证模型。

案例 Python 源代码和注释如下：

```
############################################################3
import pandas as pd
# 导入数据集拆分工具
from sklearn.model_selection import train_test_split
# 导入 KNN 分类模型
from sklearn.neighbors import KNeighborsClassifier

# 特征文字
feat_cols =['mass','width','height','color_score']

# 读取数据
data = pd.read_csv('c:\\pythoncode\\fruit_data.csv')

# 预处理
fruit2num = {'apple'    :  0 ,
             'mandarin' :  1 ,
             'orange'    :  2 ,
             'lemon'     :  3
            }
data['label'] = data['fruit_name'].map(fruit2num)
# 取出 X 和 y
X = data[feat_cols].values
y = data['label'].values

# 划分数据，生成训练数据集和测试数据集
X_train_set, X_test_set ,y_train_set, y_test_set = train_test_split(X,y,
```

```
    random_state = 20, test_size= 1/5)
print('原始数据集共{}个样本，其中训练集样本数为{}，测试集样本数为{}'.format(
    X.shape[0], X_train_set.shape[0], X_test_set.shape[0]))

# 指定模型的 n_neighbors 参数值为 5，距离度量用欧氏距离
knn_model = KNeighborsClassifier(n_neighbors=5,weights='distance',p=2)
# 通过 fit()函数来训练模型
knn_model.fit(X_train_set, y_train_set)

# 准确率检测
accur = knn_model.score(X_test_set,y_test_set)

print(f'准确率为{accur*100}%')

# 试试看
num2fruit = dict(zip(fruit2num.values(),fruit2num.keys()))

for idx in range(X_test_set.shape[0]):
    test_feat = [X_test_set[idx]]
# 使用 predict 进行预测
    y_pridict = num2fruit.get(int(knn_model.predict(test_feat)))
    y_real = num2fruit.get(y_test_set[idx])
    YorN = '对' if y_pridict == y_real else '错'
    print(f'第{idx+1}个测试水果的结果是{y_pridict}， '
          f'本来应该是{y_real}，所以测{YorN}了')
#################################################
```

这个程序的输出结果如图 3-13 所示。

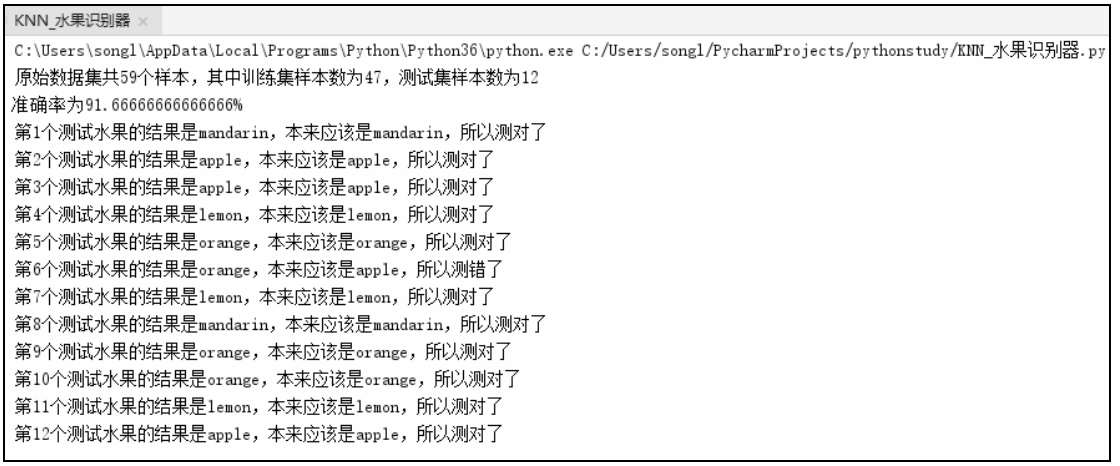

图 3-13

在机器学习的过程中，为了测试模型的性能，需通过将数据集划分为训练集和验证集，然后对验证集进行预测评估，否则我们无法知道它对于新的水果所进行的分类是否准确。所以程序代码中把数据集分为两个部分：一部分称为训练数据集（对于监督学习，训练数据集包括两部分：输入和结果，每一行输入都对应一行结果，结果是输入的正确分类即标签）；另一部

分称为测试数据集。在 scikit-learn 中，有一个 train_test_split()函数，它就是用来帮助用户把数据集拆分的工具。我们调用 train_test_split()函数将数据集中的数据分为训练数据集和测试数据集。它有四个参数，第一个参数是待划分样本数据；第二个参数是待划分样本数据的结果（标签）；第三个参数 test_size 测试数据占样本数据的比例，若整数则是样本数量；第四个参数 random_state 设置随机数种子，这是因为 train_test_split()函数会生成一个伪随机数，并根据这个伪随机数对数据集进行拆分。而我们有时候需要在一个项目中，让多次生成的伪随机数相同，方法就是通过固定 random_state 参数的数值，相同的 random_state 参数会一直生成同样的伪随机数。

另外，在 scikit-learn 中，机器学习模块都是在其固定的类中运行的，而 k 最近邻算法是在 neighbors 模块中的 KNeighborsClassifier 类中运行的。对于 KNeighborsClassifier 类来说，最关键的参数就是近邻的数量（scikit-learn 使用 kNN，而 kNN 算法的一个重要的参数就是 k 的取值），也就是 n_neighbors 参数。参数 n_neighbors=5 表示 k=5，即抽取未知样本附近最近的 5 个点进行投票。参数 weights='distance'时，表示一个已知样本点距离未知点的距离越小，其投票时所占权重越大。另外怎样度量距离，计算距离的方法有 Euclidean（欧氏距离）、Chebyshev（切比雪夫距离）、Manhattan（曼哈顿距离）等，而参数 p=2 等价于 Euclidean 欧氏距离，参数 p=1 代表曼哈顿距离。

代码中 knn_model 是我们在 KNeighborsClassifier 类中创建的一个对象。我们要调用 knn_model 的对象中名为 fit()的方法来进行建模（也称为训练模型，其中 fit 就是拟合的意思），建模的依据就是训练数据集中的样本数据 x_train 和其对应的标签 y_train。

现在我们可以使用刚刚建好的模型对新的样本分类进行预测了，测试数据集并不参与建模，但是我们可以用模型对测试数据集进行分类，然后和测试数据集中的样本实际分类进行对比，看吻合度有多高。吻合度越高，模型的得分越高，说明模型的预测越准确。

对于训练之后的模型，调用 predict()函数来预测数据的结果。predict()预测函数，它接收输入的数组类型测试样本（一般是二维数组，每一行是一个样本，每一列是一个属性），它返回数组类型的预测结果。score()函数是计算模型准确率的函数，输出为一个浮点数（float 类型），表示准确率。

3.5　kNN 算法优缺点

kNN 算法作为一个简单实用的机器学习算法，日常的使用中也能处理很多问题，这里总结一下。

kNN 的优点：

- kNN 可以处理分类问题，同时天然可以处理多分类问题。
- kNN 简单、易懂，同时也很强大，对于手写数字的识别、鸢尾花这一类问题来说，准确率很高。

- kNN 还可以处理回归问题，也就是预测。

kNN 的缺点：

- 效率低，因为每一次分类或者回归，都要把训练数据和测试数据都算一遍，如果数据量很大的话，需要的算力会很惊人，但是在机器学习中，大数据处理又是很常见的一件事。
- 对训练数据依赖度特别大，虽然所有机器学习的算法对数据的依赖度很高，但是 kNN 尤其严重，因为如果我们的训练数据集中，有一两个数据是错误的，刚刚好又在我们需要分类的数值旁边，这样就会直接导致预测的数据的不准确，对训练数据的容错性太差。

维数灾难，kNN 对于多维度的数据处理也不是很好，随着维度的增加，“看似相近”的两个点之间的距离越来越大，对于 kNN 这种高度依赖距离的算法来说，这个也会影响准确率。

第4章

回归算法

回归是数学建模、分类和预测中最古老但功能非常强大的工具之一，在工程、物理学、生物学、金融、社会科学等各个领域都有应用，是数据科学家常用的基本工具。回归算法是通过利用测试集数据来建立模型，再利用这个模型训练集中的数据进行处理的算法。线性回归旨在寻找到一条线，这条线到达所有样本点的距离之和是最小的，常用在预测和分类领域。

4.1 线性回归

4.1.1 什么是线性回归

我们首先用弄清楚什么是线性，线性简单地说两个变量之间的关系是一次函数关系，对应的图像是直线（非线性：两个变量之间的关系不是一次函数关系，对应的图像不是直线）。在监督学习中，如果预测的变量是离散的，我们称其为分类（如决策树等），如果预测的变量是连续的，我们称其为回归。回归是指人们在测量事物的时候因为客观条件所限，求得的都是测量值，而不是事物真实的值，为了能够得到真实值，无限次地进行测量，最后通过这些测量数据计算回归到真实值，这就是回归的由来。

假定输入变量（X）和单个输出变量（Y）之间呈线性关系，我们就是要找到预测值 Y 的线性方程，假设线性回归是个黑盒子，那按照程序员的思维来说，这个黑盒子就是个函数，然后呢，我们只要往这个函数传一些参数作为输入，就能得到一个结果作为输出。直接说就是要用这个函数逼近这个真实值。

所以线性回归的定义就是，寻找一条直线，最大程度“拟合”样本特征和样本输出标记之间的关系。如果样本特征只有一个，就称为简单线性回归。

举个例子，比如说在房价上，房子的居住面积和房子的价格有着明显的关系。如果横坐标 X 代表房子居住面积大小，纵坐标 Y 代表房价，那么在坐标系中可以看到这些点，要求得这个简单线性回归模型 y=ax+b，它就要对样本点进行线性拟合以预测尽可能准确的函数，这个过程就是线性回归。如图 4-1 所示，线性回归就是要找一条直线，并且让这条直线尽可能地拟合图中的数据点。

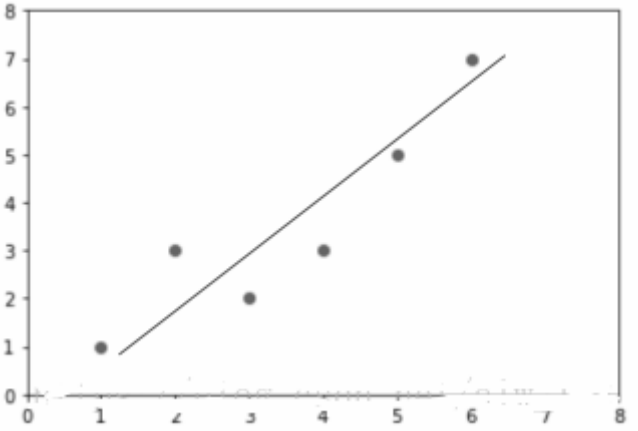

图 4-1

刚刚我们举的例子是一维的例子，特征只有房子居住面积的大小，即一元线性回归分析，只包括一个自变量和一个因变量，且二者的关系可用一条直线近似表示。一元线性回归分析研究单变量对因变量的影响，我们能够给出单变量线性回归模型。如图 4-2 所示，如果回归分析中包括两个或两个以上的自变量，且因变量和自变量之间是线性关系，则称为多元线性回归分析模型。而多元线性回归则是研究多个自变量对因变量的影响，比如影响体重的因素不仅仅有身高，可能还有一些疾病等其他因素会导致体重的变化。

这里一个变量，可以写成：

$$h_\theta(x) = \theta_0 + \theta_1 x$$

两个变量可以写成：

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

多个变量即可以写成：

$$h_\theta(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T x$$

图 4-2

现在假设我们的 Price 房价数据中有两个特征是 Living area（居住面积），bedrooms（卧室个数），那图像可能就是如图 4-3 所示，对于二维空间线性是一条直线，对于三维空间线性是一个平面。

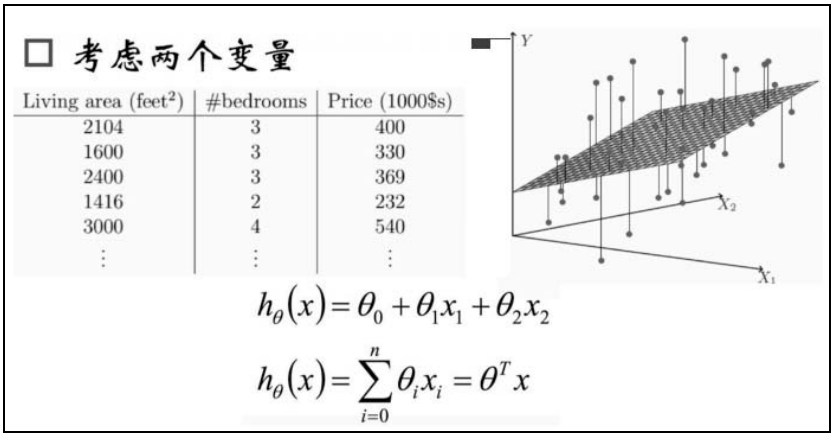

图 4-3

4.1.2　最小二乘法

其实找直线的过程就是在做线性回归，但如果 100 人来找这条直线就可能找出 100 种直线来，那究竟是哪种直线更好呢？肯定是要有一个评判的标准，来评判哪条直线才是最好的。

一个简单的想法是，只要算一下实际房价和我找出的直线根据房子面积大小预测出来的房价之间的差距即可。说白了就是计算两点的距离。当我们把所有实际房价和预测出来的房价的差距（距离）算出来然后做个加和，就能量化出现在我们预测的房价和实际房价之间的误差。如图 4-4 所示，我们画了很多条小竖线，每一条小竖线就是实际房价和预测房价的差距（离差）。

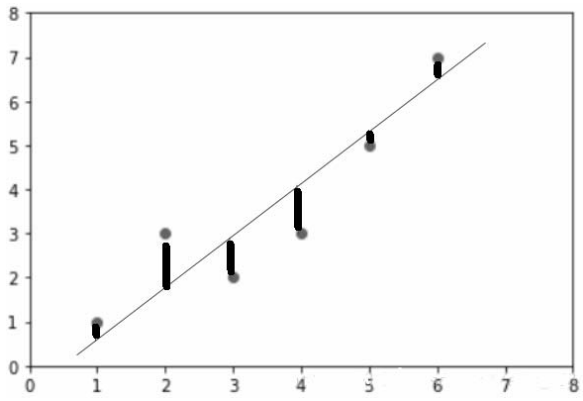

图 4-4

然后把每条小竖线的长度（离差）加起来就等于我们现在通过这条直线预测出的房价与实际房价之间的差距。也就是说这 n 个离差构成的总离差越小越好，只有如此才能使直线最贴近已知点。

一个很自然的想法是把各个离差加起来作为总离差。但是，由于离差有正有负，直接相加会互相抵消，如此就无法反映这些数据的贴近程度，即这个总离差不能用 n 个离差之和来表示，一般做法是我们用离差的平方和，其实就是欧式距离加和，公式如图 4-5 所示（其中 $y^{(i)}$

表示的是实际房价，$y^{(i)}$表示的是预测房价）。

$$\sum_{i=1}^{m}(y^{(i)}-\hat{y}^{(i)})^2$$

图 4-5

作为总离差，并使之达到最小。这样回归直线就是所有直线中 Q 取最小值的那一条。由于平方又叫二乘方，所以这种使“离差平方和为最小”的方法，叫作最小二乘法。

最小二乘法是勒让德（A. M. Legendre）于 1805 年在其著作《计算慧星轨道的新方法》中提出的。它的主要思想就是求解未知参数，使得理论值与观测值之差（即误差，或者说残差）的平方和达到最小。我们求回归直线方程的过程其实就是求离差最小值的过程，在一元线性回归中，最小二乘法就是试图找到一条直线，使所有样本到直线的欧氏距离之和最小。它是通过最小化误差的平方和，使得拟合对象无限接近目标对象（即拟合函数更接近于目标函数）。

再介绍一个在机器学习中的术语叫损失函数（就是计算误差的函数）。损失函数是贯穿整个机器学习过程中非常重要的一个概念，它就是量化预测结果和真实结果的误差的一个函数。这个损失函数有点类似马戏团训练猴子中的用于判断猴子做得“好”还是“不好”，给予奖惩这样的机制，并且对“好”和“不好”进行量化。如果损失函数做得好，我们就能训练出一个更好的模型。

最小二乘法以估计值与观测值之差的平方和作为损失函数，有了这个函数，我们就相当于有了一个评判标准，利用损失函数来衡量。要通过最小化损失函数，也就是最小化偏差来得到最好的参数。当这个函数的值越小，就越说明我们找到的这条直线越能拟合我们的房价数据。

4.1.3 梯度下降法

我们想要让这个方程拟合得非常好，那么就要使误差尽量小，评价误差小的方法就是所有误差的平方和最小，计算误差平方和最小的方法除了常见的就是最小二乘法，还有一种方法叫梯度下降法。

梯度下降（Gradient Descent）在机器学习中应用十分广泛，它的主要目的是通过迭代找到目标函数的最小值，想办法使真实值和预测值之间的差异越小越好。

梯度下降法的基本思想可以类比为一个下山的过程。

假设这样一个场景，一个人被困在山上，需要从山上下来（找到山的最低点，也就是山谷）。但此时山上的浓雾很大，导致可视度很低；因此，下山的路径就无法确定，必须利用自己周围的信息一步一步地找到下山的路。这个时候，便可利用梯度下降算法来帮助自己下山。怎么做呢？就是决定走一步算一步，也就是每次沿着当前位置最陡峭最易下山的方向前进一小步，然后继续沿下一个位置最陡方向前进一小步。这样一步一步走下去，一直走到觉得我们已经到了山脚下。这里下山最陡的方向就是梯度的负方向，如图 4-6 所示。

我们可以假设这座山最陡峭的地方是无法通过肉眼立马观察出来的，而是需要一个复杂的工具来测量，同时，这个人此时正好拥有测量出最陡峭方向的能力。所以，此人每走一段距

离，都需要一段时间来测量所在位置最陡峭的方向，这是比较耗时的。那么为了在太阳下山之前到达山底，就要尽可能地减少测量方向的次数。这是一个两难的选择，如果测量得频繁，可以保证下山的方向是绝对正确的，但又非常耗时；如果测量得过少，又有偏离轨道的风险。所以需要找到一个合适的测量方向的频率，来确保下山的方向正确，同时又不至于耗时太多！

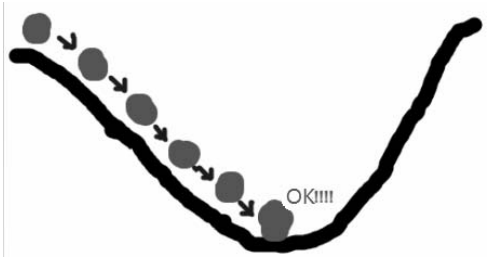

图 4-6

梯度下降的基本过程就和下山的场景非常类似。首先，这个函数就代表着一座山。我们的目标就是找到这个函数的最小值，也就是山底。根据之前的场景假设，最快的下山的方式就是找到当前位置最陡峭的方向，然后沿着此方向向下走，对应到函数中，就是找到给定点的梯度，然后朝着梯度相反的方向，就能让函数值下降得最快！因为梯度的方向就是最快下山，或者说沿着变化率最大的那个方向，这正是我们所需要的。我们只要沿着梯度的方向一直走，就能走到局部的最低点！所以，我们重复利用这个方法，反复求取梯度，最后就能到达局部的最小值，这就类似于我们下山的过程。而求取梯度就确定了最陡峭的方向，也就是场景中测量方向的手段。在梯度下降算法中被称作为步长，意味着我们可以通过学习率来控制每一步走的距离，以保证不要走太快，错过了最低点。同时也要保证不要走得太慢，以致太阳下山了，还没有走到山下。

至此，就基本介绍完了梯度下降法的基本思想和算法流程，下面用 Python 实现一个简单的梯度下降算法拟合直线的案例。

场景是一个简单的线性回归的例子：假设现在我们有一系列的点，如图 4-7 所示。

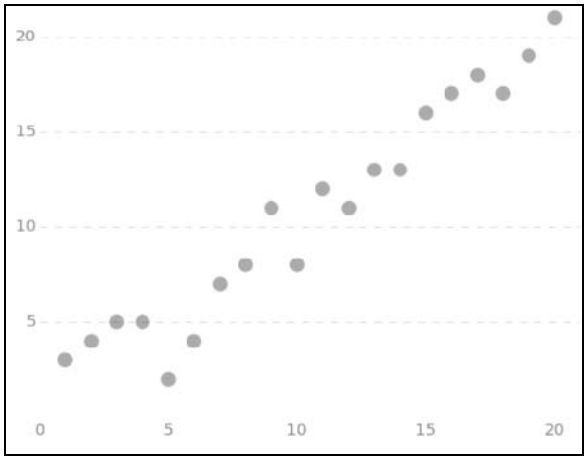

图 4-7

我们将用梯度下降算法来拟合出这条直线！

首先需要定义一个代价函数，在此我们选用均方误差代价函数（也称平方误差代价函数），如图 4-8 所示。

$$J(\Theta)=\frac{1}{2m}\sum_{i=1}^{m}(h_\Theta(x^{(i)})-y^{(i)})^2$$

图 4-8

此公式中：

（1）m 是数据集中数据点的个数，也就是样本数。

（2）1/2 是一个常量，这样是为了在求梯度的时候，二次方乘下来的 2 就和这里的 1/2 抵消了，自然就没有多余的常数系数，方便后续的计算，同时对结果不会有影响。

（3）y 是数据集中每个点的真实 y 坐标的值，也就是类标签。

（4）h 是我们的预测函数（假设函数），根据每一个输入 x，根据 Θ 计算得到预测的 y 值。

从代价函数中可以看到，代价函数中的变量有两个，所以是一个多变量的梯度下降问题，求解出代价函数的梯度，也就是分别对两个变量进行微分。明确了代价函数和梯度，以及预测的函数形式，下面就可以开始编写代码了。但在此之前需要说明一点，就是为了方便代码的编写，我们会将所有的公式都转换为矩阵的形式，Python 中计算矩阵是非常方便的，同时代码也会变得非常的简洁。然后我们将代价函数和梯度转化为矩阵向量相乘的形式。

案例的程序代码和注释如下：

```
#################################################################
from numpy import *
# 我们需要定义数据集和学习率
# 数据集大小，即 20 个数据点
m = 20
# x 的坐标以及对应的矩阵
X0 = ones((m, 1))       # 生成一个 m 行 1 列的向量，也就是 x0，全是 1
X1 = arange(1, m+1).reshape(m, 1)  # 生成一个 m 行 1 列的向量，也就是 x1，从 1 到 m
X = hstack((X0, X1))    # 按照列堆叠形成数组，其实就是样本数据
# 对应的 y 坐标
Y = array([
    3, 4, 5, 5, 2, 4, 7, 8, 11, 8, 12,
    11, 13, 13, 16, 17, 18, 17, 19, 21
]).reshape(m, 1)
# 学习率
alpha = 0.01

# 我们以矩阵向量的形式定义代价函数和代价函数的梯度
# 定义代价函数
def cost_function(theta, X, Y):
    diff = dot(X, theta) - Y    # dot() 数组需要像矩阵那样相乘，就需要用到 dot()
```

```
    return (1/(2*m)) * dot(diff.transpose(), diff)


# 定义代价函数对应的梯度函数
def gradient_function(theta, X, Y):
    diff = dot(X, theta) - Y
    return (1/m) * dot(X.transpose(), diff)

# 算法的核心部分，梯度下降迭代计算
# 梯度下降迭代
def gradient_descent(X, Y, alpha):
    theta = array([1, 1]).reshape(2, 1)
    gradient = gradient_function(theta, X, Y)
# 当梯度小于 1e-5 时，说明已进入了比较平滑状态，类似于山谷的状态，可以退出循环
    while not all(abs(gradient) <= 1e-5):
        theta = theta - alpha * gradient
        gradient = gradient_function(theta, X, Y)
    return theta

optimal = gradient_descent(X, Y, alpha)
print('optimal:', optimal)
print('cost function:', cost_function(optimal, X, Y)[0][0])

# 根据数据画出对应的图像
def plot(X, Y, theta):
    import matplotlib.pyplot as plt
    ax = plt.subplot(111)
    ax.scatter(X, Y, s=30, c="blue")
    plt.xlabel("X")
    plt.ylabel("Y")
    x = arange(0, 21, 0.2)  # x 的范围
    y = theta[0] + theta[1]*x
    ax.plot(x, y)
    plt.show()
plot(X1, Y, optimal)
################################################################################
```

上面的案例程序的运行结果如图 4-9 所示。

```
optimal: [[0.51583286]
 [0.96992163]]
cost function: 1.0149624062331013
```

图 4-9

所拟合出的直线如图 4-10 所示。

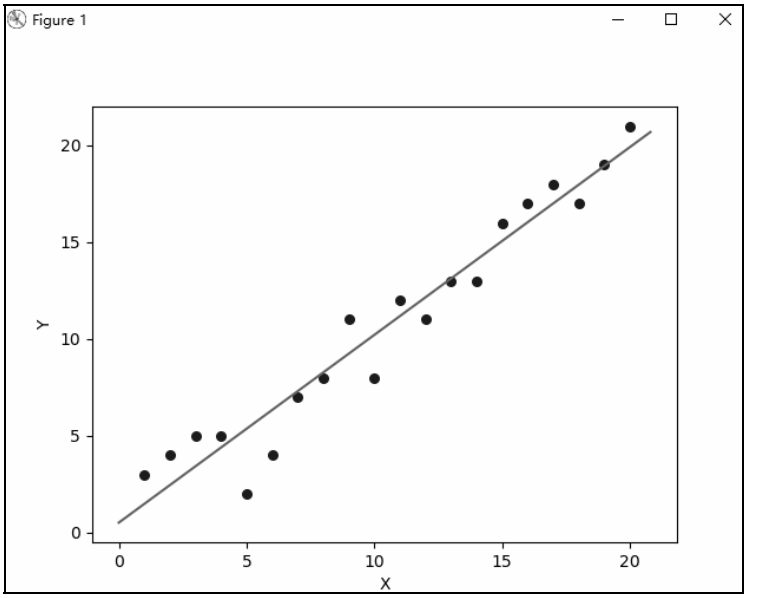

图 4-10

梯度下降策略对比，有三种常见策略，基本思想是一致的，只是在计算过程中选择的样本的数量有所不同，下面分别进行讨论。

1. 批量梯度下降法（Batch Gradient Descent）

批量梯度下降法，是梯度下降法最常用的形式，具体做法也就是在更新参数时使用所有的样本来进行更新。由于批量梯度下降法一次迭代进行了所有样本的计算，因此能准确地朝着极值前进，迭代次数也会少，但因为一次迭代用了所有样本，所以速度会慢很多。

2. 随机梯度下降法（Stochastic Gradient Descent）

随机梯度下降法，其实和批量梯度下降法原理类似，区别在于求梯度时没有用所有的 m 个样本的数据，而是仅仅选取一个样本来求梯度。对于训练速度来说，随机梯度下降法由于每次仅仅采用一个样本来迭代，因此训练速度很快，而批量梯度下降法在样本量很大的时候，训练速度不能让人满意。对于准确度来说，随机梯度下降法由于仅仅用一个样本决定梯度方向，导致得到的解很有可能不是最优的。它的优点是速度快，缺点是只用了一个样本不能代表全体样本的一个总趋势，因而可能只会收敛到局部最优。

3. 小批量梯度下降法（Mini-batch Gradient Descent）

小批量梯度下降法是批量梯度下降法和随机梯度下降法的折中，也就是对于 m 个样本，我们采用 x 个样本来迭代，即若干个样本的平均梯度作为更新方向，1<x<m。一般可以取 x=10，当然根据样本的数据，可以调整这个 x 的值。

4.2　线性回归案例实战

4.2.1　房价预测线性回归模型案例一

项目要求：利用马萨诸塞州波士顿郊区的房屋信息数据训练和测试一个模型，并对模型的性能和预测能力进行测试。

项目中使用数据集的三个字段解释如下：

- RM：房间数量。
- LSTAT：区域中被认为是低收入阶层的比率，为房屋所在地区的人口比例。
- MEDV：房屋的平均价格（目标特征，即我们所要预测的值）。

在项目的第一个部分，你会对波士顿房地产数据进行初步的观察并给出你的分析。通过对数据的探索来熟悉数据可以让你更好地理解和解释你的结果。由于这个项目的最终目标是建立一个预测房屋价值的模型，我们需要将数据集分为特征（Feature）和目标变量（Target Variable）。

- 特征：'RM' 和 'LSTAT'，给我们提供了每个数据点的数量相关的信息。
- 目标变量：'MEDV'，是我们希望预测的变量。

范例程序代码和注释如下：

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
# 读取数据
df = pd.read_csv('c:\\pythoncode\\boston.csv')
# print(df.head())
# seaborn 中 pairplot 函数可视化探索数据特征间的关系，可以通过 pip install seaborn 安装
# 绘制的散点图矩阵可以查看数据集内部特征之间的关系
# 简单起见，只绘制了三列（RM、MEDV(标记)、LSTAT）特征和标记之间的联系
sns.set(context = 'notebook')
cols = ['LSTAT', 'RM','MEDV']
sns.pairplot(df[cols], height=2)
plt.show()
```

这个范例程序的运行结果如图 4-11 所示。

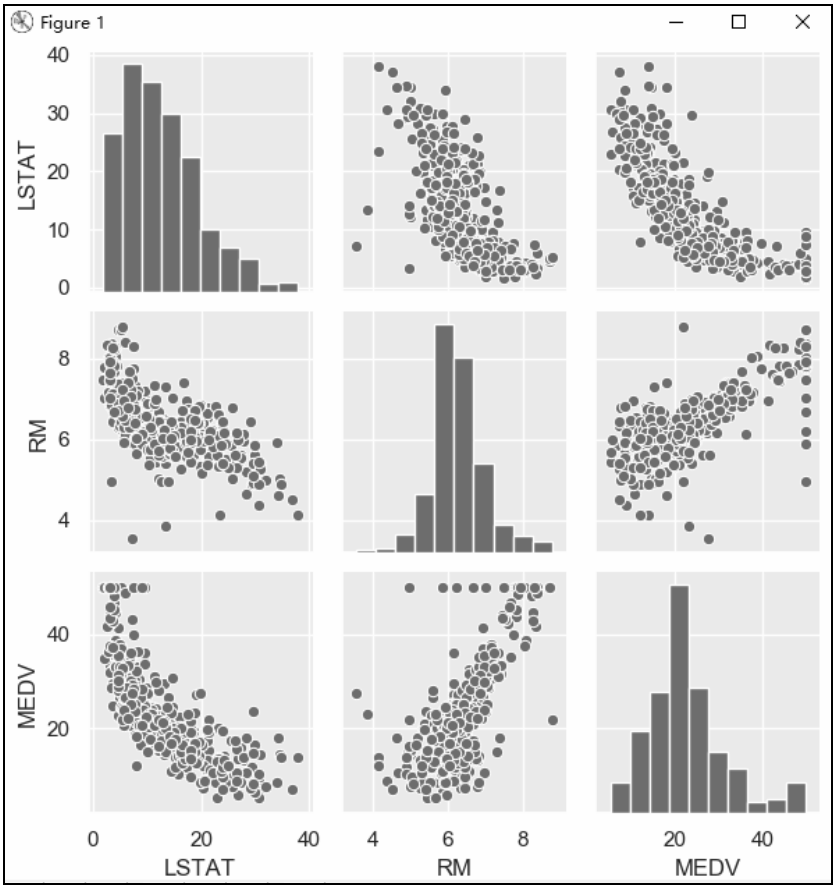

图 4-11

不难发现，MEDV 与 LSTAT 存在反相关关系，MEDV 与 RM 存在正相关关系。这里我们看出和波士顿房价相关性最强的两个因素，分别是 RM（每栋住宅的房间数）、LSTAT（地区中有多少房东属于低收入人群）。还是具备一定逻辑性的，首先，房子越大房价自然高（不管在哪个地域）；关于有多少房东属于低收入人群和房价的负相关关系，这个也比较好理解。

调用 sns.heatmap()方法绘制的关联矩阵可以看出特征之间的相关性大小，关联矩阵是包含皮尔森积矩相关系数的正方形矩阵，用来度量特征对之间的线性依赖关系。

```
# 可视化相关系数矩阵，理论：皮尔逊相关系数
# 在统计学中，皮尔逊相关系数( Pearson Correlation Coefficient)，又称皮尔逊积矩相关系数
# 是用于度量两个变量 X 和 Y 之间的相关（线性相关）
cm = np.corrcoef(df[cols].values.T)
sns.set(font_scale=1.5)
hm = sns.heatmap(cm,
                 cbar=True,
                 annot=True,
                 square=True,
                 fmt='.2f',
                 annot_kws={'size':15},
                 yticklabels=cols,
```

```
                xticklabels=cols)
plt.show()
```

上面程序代码的运行结果如图 4-12 所示。可以看出特征 LSTAT 和标记 MEDV 的具有最高的相关性-0.74，但是在散点图矩阵中会发现 LSTAT 和 MEDV 之间存在着明显的非线性关系；而特征 RM 和标记 MEDV 也具有较高的相关性 0.70，并且从散点矩阵中会发现特征 RM 和标记 MEDV 之间存在着线性关系。因此接下来将使用 RM 作为线性回归模型的特征。

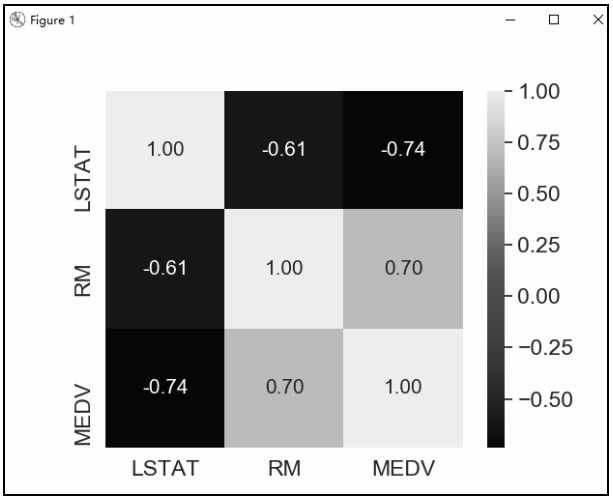

图 4-12

```
from sklearn.linear_model import LinearRegression
sk_model = LinearRegression()
X = df[['RM']].values
y = df['MEDV'].values
sk_model.fit(X, y)
print('Slope: %.3f' % sk_model.coef_[0])           # 斜率
print('Intercept : %.3f' % sk_model.intercept_)    # 截距

def Regression_plot(X, y, model):
    plt.scatter(X, y, c='blue')
    plt.plot(X, model.predict(X), color='red')
    return None
Regression_plot(X, y, sk_model)
plt.xlabel('RM')
plt.ylabel('House price')
plt.show()
```

上面程序的运行结果如图 4-13 所示，普通线性回归斜率为：9.102。

```
Slope: 9.102
Intercept : -34.671
```

图 4-13

画出的图形如图 4-14 所示。

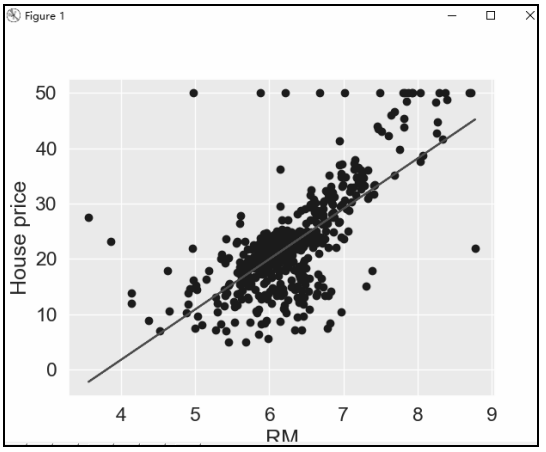

图 4-14

4.2.2 房价预测线性回归模型案例二

数据集 ex1data2.txt 中的数据是两个特征，作一个最简单的多元（在此为二元）线性回归。对 ex1data2.txt 中的数据进行线性回归，所有样本都用来训练和预测。

范例程序代码如下：

```
#################################################################
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from mpl_toolkits.mplot3d import Axes3D        # 不要去掉这个 import
from sklearn.metrics import mean_squared_error, r2_score
plt.rcParams['font.sans-serif'] = ['SimHei']   # 用来正常显示中文标签
plt.rcParams['axes.unicode_minus'] = False     # 用来正常显示负号

# 数据格式：城市人口，房间数目，房价

# 读取数据
data = np.loadtxt('c:\\pythoncode\\ex1data2.txt', encoding='gbk',delimiter=',')
data_X = data[:, 0:2]
data_y = data[:, 2]

# 训练模型
model = LinearRegression()
model.fit(data_X, data_y)

# 利用模型进行预测
y_predict = model.predict(data_X)

# 结果可视化
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.scatter(data_X[:, 0], data_X[:, 1], data_y, color='red')
```

```
ax.plot(data_X[:, 0], data_X[:, 1], y_predict, color='blue')
ax.set_xlabel('城市人口')
ax.set_ylabel('房间数目')
ax.set_zlabel('房价')
plt.title('线性回归——城市人口、房间数目与房价的关系')
plt.show()

# 模型参数
print(model.coef_)
print(model.intercept_)
# MSE
print(mean_squared_error(data_y, y_predict))
# R^2
print(r2_score(data_y, y_predict))
##############################################################
```

这个范例程序的运行结果如图 4-15 所示，可知函数的形式以及 *R*2 值为 0.73。

```
[  139.21067402 -8738.01911233]
89597.90954279754
4086560101.2056565
0.7329450180289143
```

图 4-15

结果可视化如图 4-16 所示，是个二元线性回归，使用一个回归平面来拟合样本点。

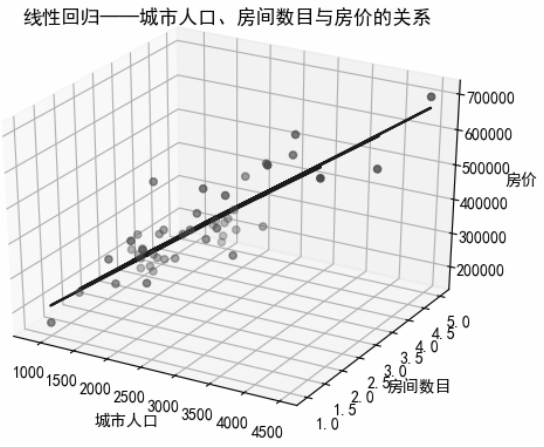

图 4-16

4.3 逻辑回归

4.3.1 逻辑回归概念和原理

和其他机器学习算法一样，逻辑回归（或称为 Logistic 回归）也是从统计学中借鉴来的，尽管名字里有“回归”这两个字，但它不是一个需要预测连续结果的回归算法。逻辑回归得到一个离散的结果，但线性回归得到一个连续的结果。预测房价的模型算是返回连续结果的一个例子。该值随着房子大小或位置等参数的变化而变化。离散的结果总是一件事（患者有癌症）或另一个（患者没有癌症）。逻辑回归是二分类任务的首选方法。它输出一个 0~1 之间的离散二值结果。简单地说，它的结果不是 1 就是 0。

（1）逻辑回归出现的背景

线性回归能对连续值结果进行预测，而现实生活中常见的另一类问题是分类问题。最简单的情况是：是与否的二分类问题。比如说，医生需要判断病人是否患癌症，银行要判断一个人的信用程度是否达到可以给他发信用卡的程度，邮件收件箱要自动对邮件分类为正常邮件和垃圾邮件等。癌症检测算法可看作是逻辑回归问题的一个简单例子，这种算法输入病理图片并且应该辨别患者是患有癌症（1）或没有癌症（0）。

当然，我们最直接的想法是，既然能够用线性回归预测出连续值的结果，那根据结果设定一个阈值应该可用于分类问题。事实上，对于非常标准的情况，确定是可以的。但是，在很多实际的情况下，我们需要学习的分类数据并没有这么准确，那我们的线性回归模型，就不那么适用了，并且设定的阈值也就失效了，这个时候我们借助线性回归和阈值的方式，已经很难完成一个表现良好的分类器。

在这样的场景下，逻辑回归就诞生了。它的核心思想是，如果线性回归的结果输出是一个连续值，而值的范围是无法限定的，那么我们有没有办法把这个结果映射为可以帮助我们判断的结果呢。如果输出的结果是(0,1)的一个概率值，那么这个问题就能很清楚地解决了。

（2）逻辑回归基本的知识点

逻辑回归虽然名字里带“回归”字样，但是它实际上是一种分类方法，主要用于二分类问题（即输出只有两种，分别代表两个类别），所以利用了逻辑函数（或称为 Sigmoid 函数）估计概率，来衡量因变量（我们想要预测的标签）与一个或多个自变量（特征）之间的关系。这些概率必须二值化才能真正地进行预测。

我们先来画出 Sigmoid 函数图像，如图 4-17 所示，对应的函数图像是一个取值在 0 和 1 之间的 S 型曲线。从函数图上可以看出，函数 y=g(z)在 z=0 的时候取值为 1/2，而随着 z 逐渐变小，函数值趋于 0，z 逐渐变大的同时函数值逐渐趋于 1，这正是一个概率的范围。一个 S 形曲线，它可以将任意实数值映射到介于 0 和 1 之间的值，但并不会取到 0/1，然后使用阈值分类器将 0 和 1 之间的值转换为 0 或 1。

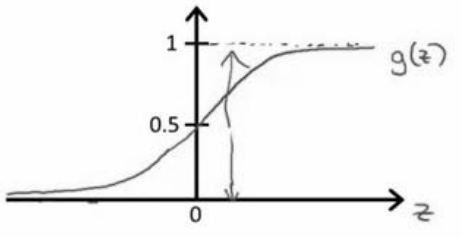

图 4-17

如图 4-18 所示，说明了逻辑回归得出预测所需要的所有步骤。

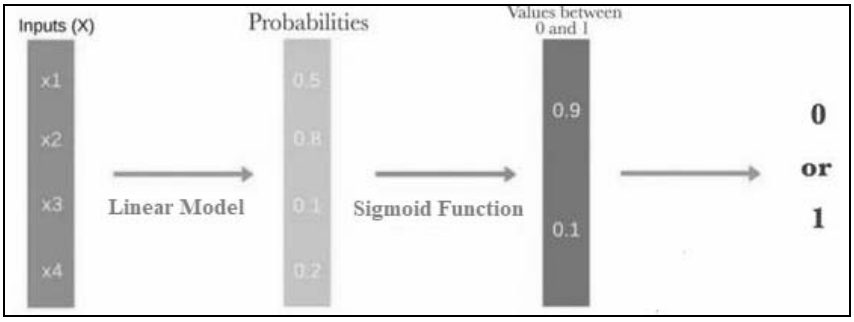

图 4-18

（3）何时适用？

逻辑回归通过线性边界将输入分成两个“区域”，每个类别划分一个区域。因此，我们的数据应当是线性可分的，如图 4-19 所示的数据点，换句话说，当 Y 变量只有两个值时（例如，当面临分类问题时），我们应该考虑使用逻辑回归。

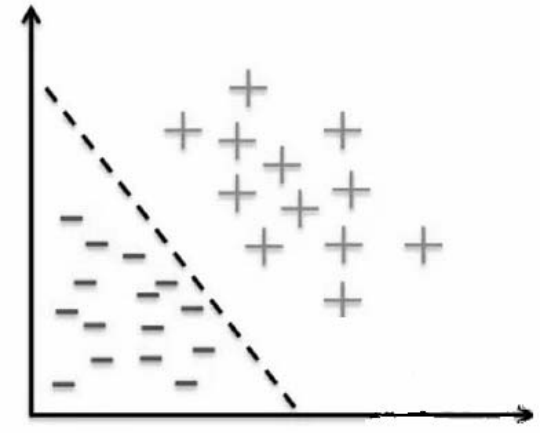

图 4-19

逻辑回归的一大优点是它非常容易实现，且训练起来很高效。在研究中，我们通常以逻辑回归模型作为基准，再尝试使用更复杂的算法。逻辑回归并非最强大的算法之一，它可以很容易地被更为复杂的算法所超越。另一个缺点是它高度依赖正确的数据表示。这意味着逻辑回

归在你已经确定了所有重要的自变量之前还不会成为一个有用的工具。由于其结果是离散的，因此逻辑回归只能预测分类结果。另外，逻辑回归同时也以其容易过拟合而闻名。

4.3.2 逻辑回归案例实战

但是在某些时候，我们要做的可能是一个分类模型，那么这里就可能用到线性回归模型的变种——逻辑回归。本节我们就逻辑回归来做一个详细的说明。

实例引入

张三、李四、王五、赵六都要贷款了，贷款时银行调查了他们的月薪和住房面积等因素，两个因素决定了贷款款项是否可以立即到账，表 4-1 列出来了他们四个人的工资情况、住房面积和到账的具体情况。

表 4-1 四个人的工资情况、住房面积和到账的具体情况

姓名	工资（元）	房屋面积（平方米）	是否可以立即到账
张三	6000	58	否
李四	9000	77	否
王五	11000	89	是
赵六	15000	54	是

看到了这样的数据，又来了一位孙七，他工资是 12000 元，房屋面积是 60 平方米，那他的贷款可以立即到账吗？

这里直接用 sklearn 机器学习库中封装好的 LogisticRegression 逻辑回归算法。

范例程序代码如下：

```
##########################################
from sklearn.linear_model import LogisticRegression
x_data = [
    [6000, 58],
    [9000, 77],
    [11000, 89],
    [15000, 54]
]
y_data = [
    0, 0, 1, 1
]
lr = LogisticRegression()
lr.fit(x_data, y_data)
x_test = [[12000, 60]]
print('Intercept', lr.intercept_)
print('Coef', lr.coef_)
print('款项是否可以立即到账', lr.predict(x_test)[0])
######################################
```

这个范例程序的运行结果如图 4-20 所示。

```
Intercept [-0.03142387]
Coef [[ 0.00603919 -0.72587703]]
款项是否可以立即到账 1
```

图 4-20

使用 sklearn 中的 API 来实现逻辑回归模型，使用的库为 LogisticRegression，结果中输出了截距项和系数项，然后预测了是否可以立即到账，结果为 1，这表明张三进行贷款，款项可以立即到账。以上就是逻辑回归的基本推导和代码应用实例。

4.4　回归算法总结和优缺点

逻辑回归是一种被人们广泛使用的算法，因为它非常高效，不需要太大的计算量，又通俗易懂，不需要缩放输入特征，不需要任何调整，且很容易调整，并且输出校准好的预测概率。与线性回归一样，当我们去掉与输出变量无关的属性以及相似度高的属性时，逻辑回归效果确实会更好。因此特征处理在逻辑和线性回归的性能方面起着重要的作用。

总结一下，回归算法具有以下优点：

（1）思想简单，实现容易，建模迅速，对于小数据量、简单的关系很有效。由于其简单且可快速实现的原因，因此可以用它来衡量其他更复杂算法的性能。

（2）是许多强大的非线性模型的基础。

（3）线性回归模型十分容易理解，结果具有很好的可解释性，有利于决策分析。

（4）蕴含机器学习中的很多重要思想，能解决回归问题。

它的缺点主要有：

（1）对于非线性数据或者数据特征间具有相关性的多项式回归，则难以建模。我们不能用逻辑回归来解决非线性问题，因为它的决策面是线性的。

（2）难以很好地表达高度复杂的数据。