第 1 章

概　　述

日常生活中每天都在产生数据，这些原始数据存在数据不完整、数据不一致、数据异常等情况，严重影响数据的质量，甚至可能导致利用上的偏差，因此数据预处理技术应运而生。本章首先概述数据预处理，让读者对其有一个整体的认识；然后介绍 Python 数据预处理的开发工具与运行环境；最后结合中文分词的实战案例，让读者了解数据预处理的工作流程。

1.1　Python 数据预处理

1.1.1　什么是数据预处理

大数据与人工智能时代离不开海量的原始数据做支撑，这些原始数据存在数据不完整、数据不一致、数据异常等问题，很难得到高质量的数据用于数据建模，甚至可能导致工程应用的偏差。因此，要对原始数据做一定的处理。这种从原始数据到挖掘数据，对数据进行的操作叫作数据预处理。数据预处理通常包括数据清理、数据集成、数据归约、数据变换和数据降维，目的是挖掘数据背后的应用价值和社会价值。

数据预处理通俗地理解就是将原始数据转化为机器可以认知的数据形式，以适应相关技术或者算法模型。比如新闻分类案例中，原始数据是一篇篇的新闻文本，分类器并不能直接处理，需要对新闻文本分词、去除干扰词、提取词特征、词特征转化、词特征降维等操作，分类器才能对数据进行学习优化，实现工程应用。

总而言之，原始数据可能存在数据不完整、数据偏态、数据噪声、数据特征维度高、数据缺失值、数据错误值等一系列问题，经过数据预处理后的数据能够达到数据完整干净、数据特征比重合适、数据特征维度合理、数据无缺失值等优点，使数据利用更加准确、高质。

1.1.2 为什么要做数据预处理

早期互联网时代数据量较少，主要存储在数据库、文件系统等介质中，其数据分析主要靠人工统计完成。随着网络的普及，海量数据应运而生，依旧采用人工统计方法对数据处理已不合时宜。伴随着计算能力和硬件设施的提升，先前的算法理论（如神经网络等）有了用武之地，使得计算机处理海量数据成为当今数据分析人员的主要工作。

在大数据与人工智能的时代，甚至未来的一段时间，不管是无人驾驶还是智能机器人，或是其他应用，主要还是在监督式学习下进行的，这里的监督学习即需要有参考意义的历史数据做基础。这些数据不仅仅是数据库文件、文本文件，还包括视频、语音、网页等各种介质的数据。数据的存在形式呈现多样化，我们将其称之为异源数据，顾名思义指的是来自不同数据源的数据。

异源数据也是最原始的数据，包括人们在网上的任何行为记录。这些行为绝大多数是正确的，但是也可能存在错误。比如，有时候收集数据的设备可能出故障；或者是人为输入错误；数据传输中的错误；命名约定或所用的数据代码不一致导致的错误，等等。如何对这些原始数据进行预处理来提高数据质量？如何通过高质量的数据来挖掘数据背后的价值？这就是为什么要做数据预处理的直观原因之一。

数据价值挖掘的研究工作大多都集中在算法的探讨，而忽视对数据本身的研究。事实上，数据预处理对挖掘数据价值十分重要，一些成熟的算法对其处理的数据集合都有一定的要求：比如数据的完整性好、冗余性小、属性的相关性小等。数据预处理是数据建模的重要一环，且必不可少，要挖掘出有效的知识，必须为其提供干净、准确、简洁的数据。实际应用系统中收集的数据通常是“脏”数据。没有高质量的数据，就没有高质量的挖掘结果。

1.1.3 数据预处理的工作流程

构建新闻分类器时，如何正确有效地将不同数据源中的信息整合到一起，直接影响到分类器的最终结果，数据预处理正是解决这一问题的有力方案。数据预处理包含以下几个方面：

- 数据采集。指的是从网页、文件库、数据库等多渠道采集数据，这些数据主要以结构化、半结构化和非结构化的形式存在。
- 数据集成。指的是将从多个数据源中获取到的数据结合起来并统一存储。
- 文本提取。指的是将不同格式存储的文本信息统一处理，转化为文本格式。
- 数据清理。指的是通过填写缺失的值、光滑噪声数据、识别或删除离群点并解决不一致性来清理数据。
- 数据转换。指的是按照预先设计好的规则对抽取的数据进行转换，如把数据压缩到0.0 ~ 1.0区间。
- 数据归约。数据归约技术可以用来得到数据集的归约表示，它虽然小得多，但仍然接近于保持原数据的完整性。

1.1.4 数据预处理的应用场景

大数据和人工智能技术能够将隐藏于海量数据中的信息和知识挖掘出来，为人类的社会经济活动提供依据，从而提高各个领域的运行效率。其应用领域较为广泛，包括以下领域：

- 商业智能技术。
- 政府决策技术。
- 电信数据信息处理与挖掘技术。
- 电网数据信息处理与挖掘技术。
- 气象信息分析技术。
- 环境监测技术。
- 警务云应用系统(视频/网络监控、智能交通、反电信诈骗、指挥调度等公安信息系统）。
- 大规模基因序列分析比对技术。
- Web信息挖掘技术。
- 多媒体数据并行化处理技术。
- 影视制作渲染技术。
- 其他各种行业的云计算和海量数据处理应用技术等。

1.2 开发工具与环境

1.2.1 Anaconda 介绍与安装

Anaconda 是一种 Python 语言的免费增值开源发行版，用于进行大规模数据处

理、预测分析和科学计算，致力于简化包的管理和部署。Anaconda 包含了 Conda、Python 在内的超过 180 个科学包及其依赖项。Anaconda 使用软件包管理系统 Conda 进行包管理。

1. Anaconda 的优点

Anaconda 是一个用于科学计算的 Python 发行版，支持 Linux、Mac、Windows 系统，提供了包管理与环境管理的功能，可以很方便地解决多版本 Python 并存、切换以及各种第三方包的安装问题。Anaconda 利用工具/命令 Conda 来进行 Package 和 Environment 的管理，并且已经包含了 Python 和相关的配套工具。这里先解释一下 Conda、Anaconda 这些概念的差别。Conda 可以理解为一个工具，也是一个可执行命令，其核心功能是包管理与环境管理。包管理与 pip 的使用类似，环境管理则允许用户方便地安装不同版本的 Python 并可以快速切换。Anaconda 则是一个打包的集合，里面预装好了 Conda、某个版本的 Python、众多包（Package）、科学计算工具等，所以也称为 Python 的一种发行版。其实还有 Miniconda，顾名思义，它只包含最基本的内容——Python 与 Conda，以及相关的必须依赖项，对于存储空间要求严格的用户 Miniconda 是一个不错的选择。其有以下优点：

- 开源。
- 安装过程简单。
- 高性能使用 Python 和 R 语言。
- 免费的社区支持。
- Conda 包管理。
- 1,000+开源库

2. 安装 Anaconda

Anaconda 安装包下载地址为：https://www.anaconda.com/download/，进入下载页面会显示出 Python 3.0 以上版本和 Python 2.0 两种版本，如图 1-1 所示。关于 Python 3 和 Python 2 的区别请查看网址：http://www.runoob.com/python/python-2x-3x.html）。推荐下载 Python 3.0 以上版本，请读者根据自己的操作系统是 32 位还是 64 位选择对应的版本下载。本书是基于 Windows 10 操作系统安装的。

（1）安装 Anaconda 集成环境，双击下载后的 Anaconda 安装文件，如图 1-2 所示。

（2）然后一直单击“Next”按钮，直到完成配置（环境变量自动配置），配置完成后，查看是否安装成功。打开主菜单->所有应用查看安装，出现如图 1-3 所示的菜单命令则表示安装成功。

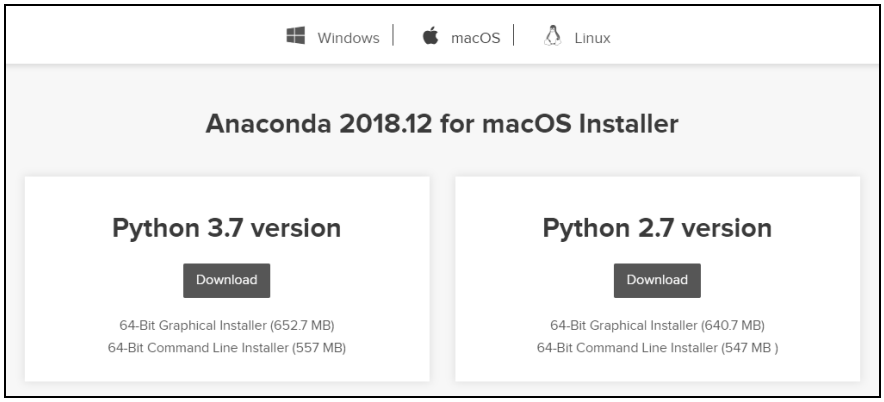

图 1-1 不同 Anaconda 版本

图 1-2 Anaconda 安装

（3）按 WIN+R 组合键输入“cmd”命令以启动命令行环境，然后输入“conda –V”查看版本号，如图 1-4 所示。

图 1-3 安装后的 Anaconda

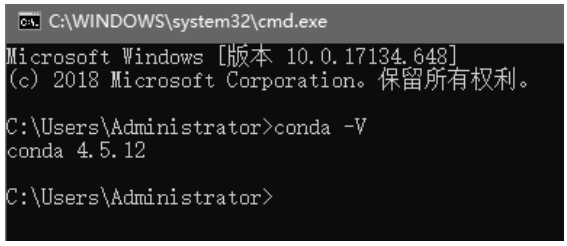

图 1-4 查看 Anaconda 版本

（4）可以看到本机 Anaconda 版本是 4.5.12，上文介绍了其集成了诸多科学包及其依赖项，看看具体包括哪些？在“命令提示符”环境下，输入“conda list”，运行结果如图 1-5 所示（截取部分包）。

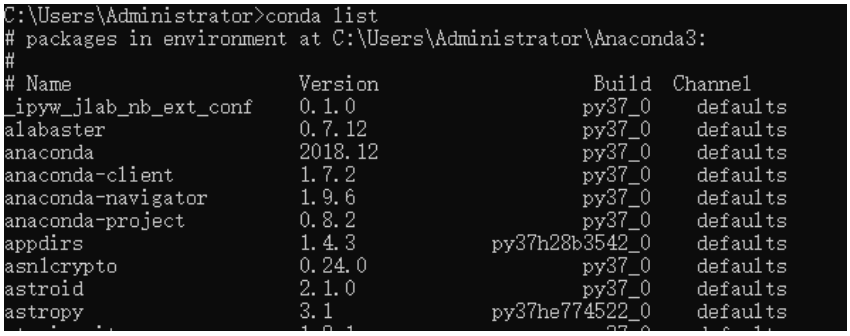

图 1-5　查看 Anaconda 预安装包

（5）Anaconda 里面有一个包名为“pip”，这个包和 Linux 的包安装命令是一样的，当需要安装第三方包的时候，直接使用这个命令即可，比如，需要进行中文分词，会依赖 Python 的分词包 jieba，可以执行如下命令：

```
pip install jieba
```

如图 1-6 所示。

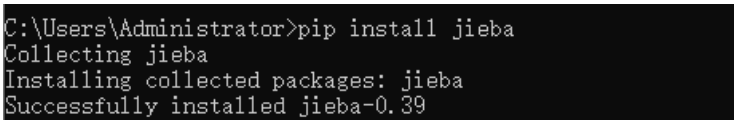

图 1-6　pip 安装结巴分词包

（6）如果不想使用这个包了，也可以直接卸载掉，命令如下：

```
pip uninstall jieba
```

如图 1-7 所示。

图 1-7　pip 卸载结巴分词包

至此，完成了 Anaconda 的安装配置以及包文件的自定义下载。需要注意的是，Anaconda 自身集成了 Python、pip、nltk、NumPy、Matplotlib 等一系列常用包。现在，已经可以使用 Python 了，考虑到熟悉 Python 开发的人员，常用 Pycharm 开发工具，熟悉 Java 的开发人员常用 Eclipse 开发工具，熟悉 C#的开发人员常用 VS 开发工具，因此只要将 Anaconda 集成到 PyDev、Pycharm、Eclipse、VS 等编译环境中即可。总之，Anaconda 是一款极为简便的集成软件包，可以将其与 Sublime、PyCharm、MyEclipse、Visual Studio 等编译环境很巧妙地融合起来（本书采用 Anaconda+Sublime Text），也可以方便地导入第三方工具包，从而极大地简化软件开发工作的流程。

1.2.2 Sublime Text

1. Sublime Text 简介

Sublime Text 是一款跨平台的文本编辑器，同时支持 Windows、Linux、Mac OS X 等操作系统和基于 Python 的插件。Sublime Text 是专有软件，可通过包（Package）扩充本身的功能。大多数的包使用自由软件授权发布，并由社区构建和维护。Sublime Text 是由程序员 Jon Skinner 于 2008 年 1 月份开发出来的，它最初被设计为一个具有丰富扩展功能的 Vim。它具有漂亮的用户界面和强大的功能，例如代码缩略图、Python 的插件、代码段等。还可自定义键绑定、菜单和工具栏。Sublime Text 的主要功能包括：拼写检查、书签、完整的 Python API、Goto 功能、即时项目切换、多选择、多窗口，等等。

Sublime Text 支持众多编程语言，并支持语法上色。内置支持的编程语言包含：ActionScript、AppleScript、ASP、batch files、C、C++、C#、Clojure、CSS、D、Diff、Erlang、Go、Graphviz (DOT)、Groovy、Haskell、HTML、Java、JSP、JavaScript、JSON、LaTeX、Lisp、Lua、Makefiles、Markdown、MATLAB、Objective-C、OCaml、Perl、PHP、Python、R、Rails、Regular Expressions、reStructuredText、Ruby、Scala、shell scripts (Bash)、SQL、Tcl、Textile、XML、XSL 和 YAML。用户可通过下载外挂支持更多的编程语言。

2. Sublime Text 的优点

Sublime Text 主要有如下优点：

- 主流前端开发编辑器。
- 体积较小，运行速度快。
- 文本功能强大。
- 支持编译功能且可在控制台看到输出。

- 内嵌 Python 解释器支持插件开发以达到可扩展目的。
- Package Control（包控制）：Sublime Text 支持的大量插件可通过其进行管理。

3. Sublime Text 的安装

本文介绍 Sublime Windows 10 系统下的安装配置，关于 Linux 和 Mac OS 下的安装基本一致，读者可自行尝试。Sublime Text 3 安装包下载地址是：http://www.sublimetext.com/3。单击该网址进入 Sublime Text 主页，选择对应的操作系统与版本，如图 1-8 所示。

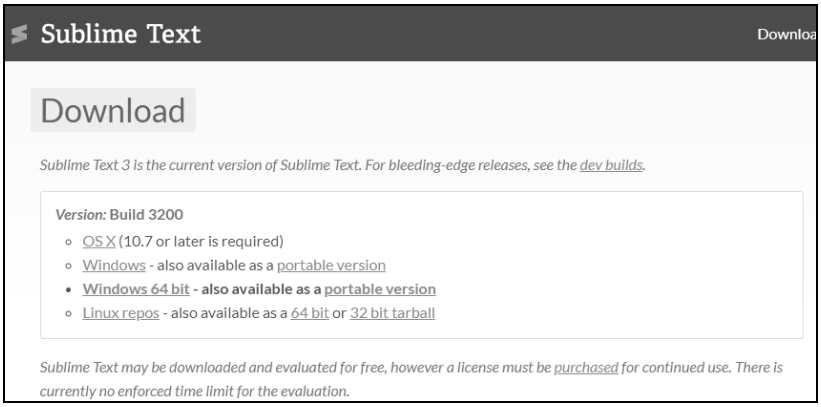

图 1-8　下载 Sublime Text 3

（1）双击下载好的 Sublime Text 3 工具包，出现如图 1-9 所示的界面。

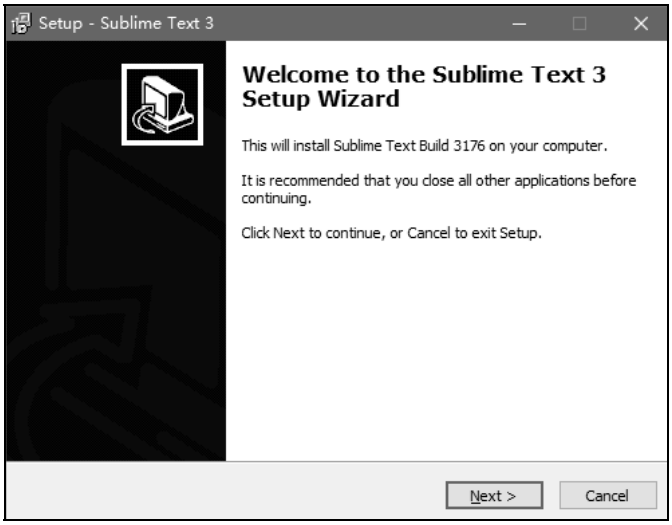

图 1-9　安装 Sublime

（2）一直单击 Next 按钮安装即可，中间保存路径可以自定义。安装成功后的结果如图 1-10 所示。

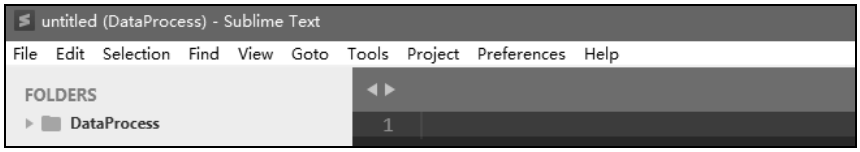

图 1-10 安装完成 Sublime

（3）安装插件 Package Control。

① 自动安装 Package Control。打开 https://packagecontrol.io/installation，复制 Sublime Text 3 中的代码，如图 1-11 所示。

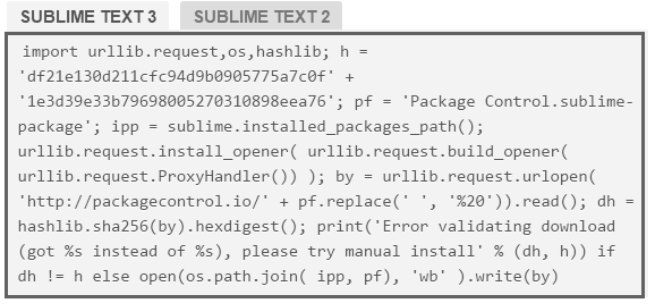

图 1-11 安装插件 Package Control

按“Ctrl+`”组合键，将上述文本代码内容复制粘贴到文本框中，按 Enter 即可。如图 1-12 所示。

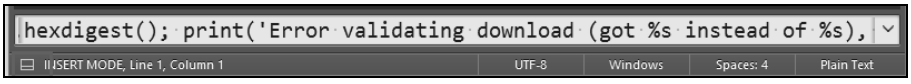

图 1-12 执行 Package Control 代码

② 如果 Package Control 官网无法打开，也可以手动安装 Package Control。

在百度网盘下载 Package Control 的安装包（链接：https://pan.baidu.com/s/14hs2-OF5L_l8UHKUkPGayQ）提取码：m7a9。下载完成后里面包含两个文件分别是：

文件 1：Package Control.sublime-package

文件 2：channel_v3.json

然后，打开 Sublime 存放插件的目录：在 Sublime Text → Preference → Browse Packages...找到“Installed Packages”文件夹，并将以上两个文件复制进去，然后重启，如图 1-13 所示。

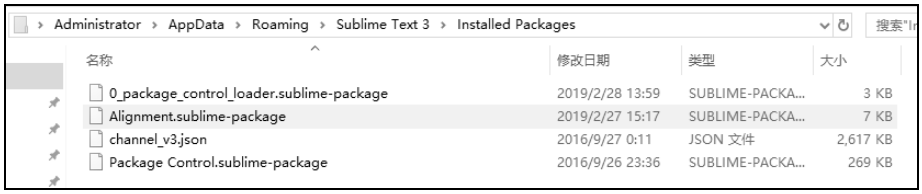

图 1-13　存放 Sublime 插件

最后，在 Sublime 下打开 Preference → Package Settings → Package Control → Setting-User，添加如下代码，如图 1-14 所示。

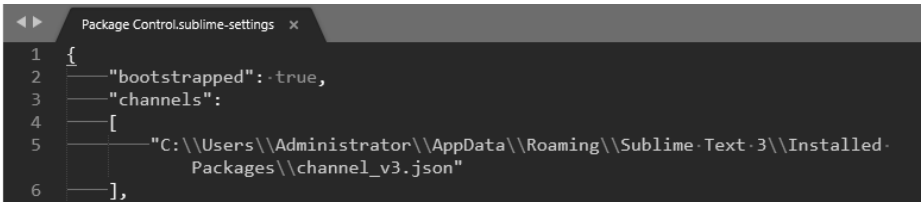

图 1-14　配置 Sublime 插件

（4）成功安装后，在 Sublime Text 3 中同时按住 Ctrl+Shift+P 组合键。最终安装成功，如图 1-15 所示。

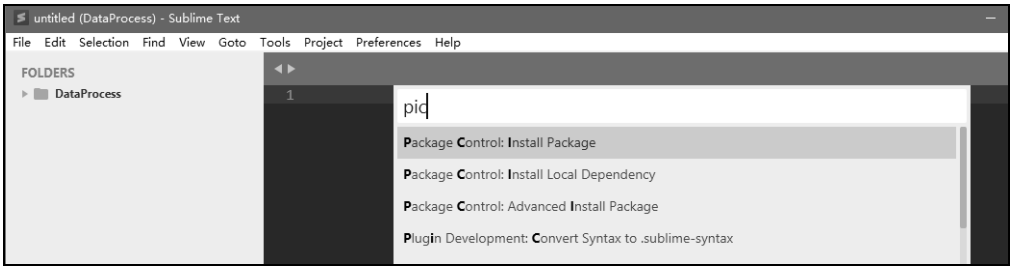

图 1-15　安装所需插件

（5）单击“Package Control:Install Package”进入查找 Python 环境配置插件“SublimeREPL”，下载安装完成后，单击“Preferences->Browse Package...”查看安装的包，如图 1-16 所示。

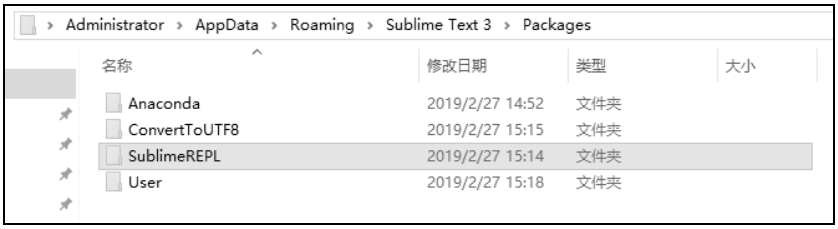

图 1-16　安装运行环境

（6）单击“Package Control:Install Package”查找 Python 环境配置插件 Anaconda，如图 1-17 所示。

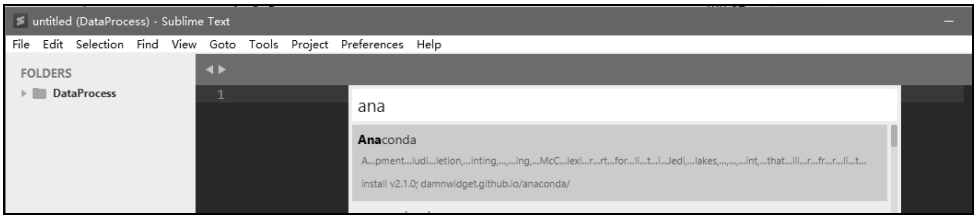

图 1-17　集成 Anaconda

（7)自定义快捷键配置。打开 Preferences → Key Bindings 输入如下代码，其中，F5 运行程序，F6 切换 IDEL 工具，Ctrl+D 自定义删除行，其他快捷键是通用的，网上有很多快捷键的资料，这里不再赘述。

```
[
    {
    "keys": ["F5"],
    "caption": "SublimeREPL: Python - RUN current file",
    "command": "run_existing_window_command",
    "args": {
        "id": "repl_python_run",
        "file": "config/Python/Main.sublime-menu"
    }
}, {
    "keys": ["F6"],
    "caption": "SublimeREPL: Python",
    "command": "run_existing_window_command",
    "args": {
        "id": "repl_python",
        "file": "config/Python/Main.sublime-menu"
    }
},{
    "keys": ["Ctrl+D"],
    "command":"run_macro_file",
    "args": {"file":"res://Packages/Default/
                                Delete Line.sublime-macro"}
}
]
```

（8）激活版本：Help >Enter LICENSE。

```
-----  BEGIN LICENSE  -----
sgbteam
Single User License
EA7E-1153259
8891CBB9 F1513E4F 1A3405C1 A865D53F
115F202E 7B91AB2D 0D2A40ED 352B269B
76E84F0B CD69BFC7 59F2DFEF E267328F
215652A3 E88F9D8F 4C38E3BA 5B2DAAE4
969624E7 DC9CD4D5 717FB40C 1B9738CF
20B3C4F1 E917B5B3 87C38D9C ACCE7DD8
5F7EF854 86B9743C FADC04AA FB0DA5C0
F913BE58 42FEA319 F954EFDD AE881E0B
------  END LICENSE  ------
```

至此，完成了 Sublime Text 3 的安装配置工作，有关详细插件安装的说明，请参考网址：http://www.open-open.com/news/view/26d731，有关快捷键的使用，请查看网址：https://segmentfault.com/a/1190000004463984。

注 意

读者也可以使用 PyCharm、Eclipse 等常用的 Python 开发工具。

1.3 实战案例：第一个中文分词程序

在数据处理工作中，分词是一项必不可少的工作，本节使用Sublime Text完成第一个分词案例。下面介绍什么是中文分词及实现方法。

1.3.1 中文分词

中文分词是指将一个汉字序列切分成一个一个单独的词。分词就是将连续的字序列按照一定的规范重新组合成词序列的过程，在英文行文中，单词间以空格作为自然分界符，中文没有一个形式上的分界符，虽然英文也同样存在短语的划分问题，不过在词这一层上，中文比英文要复杂得多，也困难得多。

例如：

英文句子：I am a student.

中文意思：我是一名学生。

由于英文的语言使用习惯，通过空格很容易拆分出单词，而中文字词界限模糊，

往往不容易区别哪些是“字”、哪些是“词”，这也是为什么要把中文词语进行切分的原因。

1. 中文分词的发展

与英文为代表的印欧语系语言相比，中文由于继承自古代汉语的传统，词语之间常没有分隔。古代汉语中除了联绵词和人名、地名等，词通常是单个汉字，所以当时没有分词书写的必要。而现代汉语中双字或多字词逐渐增多，一个字已经不再等同于一个词了。

在中文里，“词”和“词组”边界模糊，现代汉语的基本表达单元虽然为“词”，且以双字或者多字词居多，但由于人们认识水平的不同，对词和短语的边界还很难去区分。

例如：“对随地吐痰者给予处罚”，“随地吐痰者”本身是一个词还是一个短语，不同的人会有不同的标准，同样的，“海上”“酒厂”等即使是同一个人也可能做出不同的判断。如果汉语真的要分词书写，必然会出现混乱，难度也很大。中文分词的方法其实不局限于中文应用，也被应用于英文处理，例如帮助判别英文单词的边界等。

2. 中文分词的用途

中文分词是文本处理的基础，对于输入的一段中文，成功地进行中文分词，可以达到电脑自动识别语句含义的效果。中文分词技术属于自然语言处理技术的范畴，目前在自然语言处理技术中，中文处理技术比西文处理技术要落后很大一截，而许多西文的处理方法中文却不能直接采用，就是因为中文必须有分词这道工序。中文分词是中文信息处理的基础，搜索引擎就是中文分词的一个应用，其他的比如机器翻译（Machine Translation）、语音合成、自动分类、自动摘要、自动校对等，都需要用到分词。因为中文需要分词，可能会影响一些研究，但同时也为一些企业带来机会，因为国外的计算机处理技术要想进入中国市场，首先也是要解决中文分词问题。

中文分词对于搜索引擎来说，最重要的并不是找到所有的结果，因为在上百亿的网页中找到所有结果没有太多的意义，也没有人能看得完；相反，最重要的是把最相关的结果排在最前面，这也称为相关度排序。中文分词的准确与否，常常直接影响到对搜索结果的相关度排序。从定性分析的角度来看，搜索引擎的分词算法不同、词库的不同都会影响页面的返回结果。

1.3.2 实例介绍

本节实现一个有趣的应用。将电影《流浪地球》中的经典句子“道路千万条，安全第一条；行车不规范，亲人两行泪。”进行中文分词。这里需要使用第三方工具包结巴（jieba）来实现。

1.3.3 结巴实现中文分词

打开 Sublime Text 并在根目录 PyDataPreprocessing 下创建 Chapter1 文件夹，然后在 Chapter1 下面创建 CutWords.py 文件并打开。在编辑代码之前，先查看一下 jieba 包能否正常导入。按住 Alt+Shift+2 组合键进行分屏，然后按 F6 键进入 Python IDE 环境下，成功导入后如图 1-18 所示。

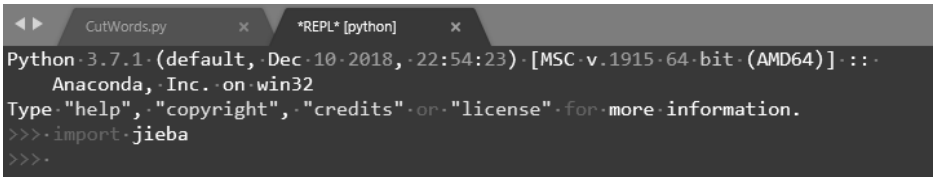

图 1-18 Sublime Text 下运行 Python IDE

图 1-18 说明 jieba 包已经成功导入，编写如下代码（源代码见：Chapter1/CutWords.py）：

```
# coding:utf8

"""
DESC: Python 数据预处理之第一个分词程序范例
Author: 伏草惟存
Prompt: code in Python3 env
"""

import jieba

str = "道路千万条,安全第一条;行车不规范,亲人两行泪。"
print("原句: \n" + str)

seg_list = jieba.cut(str)
print("分词: \n" + "/".join(seg_list))
```

代码说明：

其中第 1 行是对中文编码进行设置；第 3-7 行是注释信息；第 9 行导入 jieba 分词模块；第 14 行的调用 jieba 模块中的 cut 方法对字符串分词，数据以列表（List）形式返回；第 15 行是格式化分词结果，将 List 数据转化为 String 数据打印出来。运行代码查看结果，如图 1-19 所示。

图 1-19　中文分词的结果

1.4　本章小结

本章主要介绍了什么是数据预处理，为什么要做数据预处理，之后介绍了数据预处理的工作流程和应用场景，使读者对数据预处理有一个整体认识。工欲善其事必先利其器，选择合适的开发工具有助于更高效的工作，Anaconda+Sublime Text 就是一组不错的搭档。当然，读者也可以使用 PyCharm、MyEclipse、Visual Studio 等开发工具。下一章将介绍 Python 数据预处理的几个科学库。

第 2 章

Python 科学计算工具

Python 语言及其应用可谓如火如荼，读者应该对它们并不陌生。本章主要针对科学计算工具包 NumPy、SciPy、Pandas 分别从包的简介、安装、特点和常见方法几个方面进行介绍。科学技术工具不仅在数据预处理中使用，在机器学习和深度学习等领域都有着广泛的应用。

2.1 NumPy

NumPy 是 Python 语言常用的科学计算工具包，本节主要介绍 NumPy 的安装和特点以及 NumPy 的数组、NumPy 的数学函数、NumPy 的线性代数和 NumPy IO 操作等内置模块（源代码见：Chapter2/NumpyDome.py）。

2.1.1 NumPy 的安装和特点

NumPy（官网：http://www.numpy.org/，如图 2-1 所示），是 Python 语言的一个扩展程序库它支持大量的维度数组与矩阵运算，此外也提供大量的数学函数库。NumPy 的前身 Numeric 最早是由 Jim Hugunin 与其他协作者共同开发，2005 年，Travis Oliphant 在 Numeric 中结合了另一个同性质的程序库 Numarray 的特色，并加入了其他扩展而开发了 NumPy。

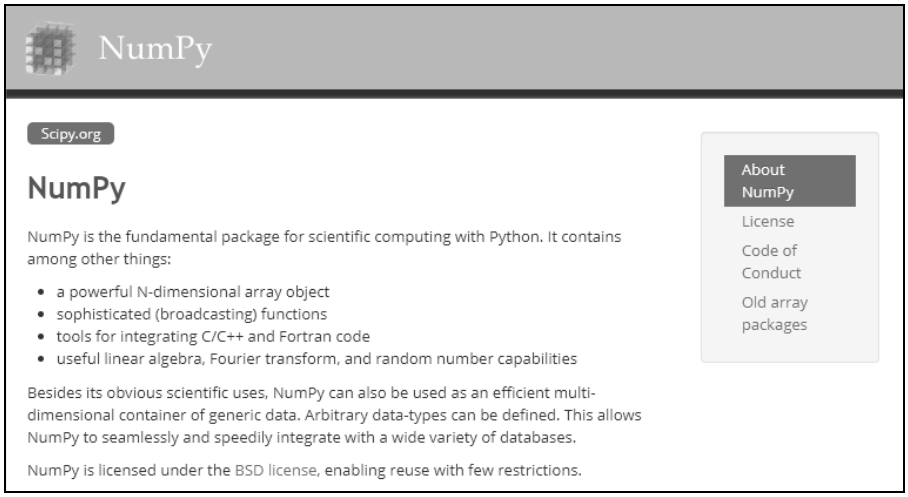

图 2-1　NumPy 官网

1. NumPy 的安装

方法一　Anaconda 已经集成了 NumPy，读者在 DOS 环境下查看：conda list

方法二　读者可以通过 pip 自动安装，执行如下命令：pip install numpy

方法三　读者在 GitHub 上下载 NumPy 源码（https://github.com/numpy/numpy），然后在源码根目录下执行如下命令：python setup.py install

安装完成后，检测是否成功。首先启动 Sublime，然后按 F6 键进入 Python 环境，最后输入 import numpy，得到以下状态即表示安装成功。如图 2-2 所示。

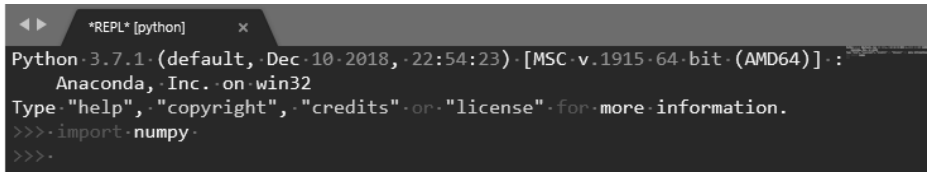

图 2-2　验证 NumPy 是否安装成功

2. NumPy 的特点

NumPy 参考了 CPython（一个使用字节码的解释器），但在 CPython 解释器上所写的数学算法代码通常远比编译过的相同代码要慢。为了解决这个难题，NumPy 引入了多维数组以及可以直接有效率地操作多维数组的函数与运算符。在 NumPy 上只要能被表示为针对数组或矩阵运算的算法，其运行效率几乎都可以与编译过的等效 C 语言代码一样快。

NumPy 通常与 SciPy（Scientific Python）和 Matplotlib（绘图库）一起使用，这种组合广泛用于替代 MatLab，是一个强大的科学计算环境，有助于我们通过 Python 学习数据科学或者机器学习。

2.1.2 NumPy 数组

数组操作是 NumPy 的主要模块之一，这里主要介绍一维数组、多维数组、数组基本运算、常数数组、数值范围创建数组、切片和索引以及数组的其他相关操作的实现。

1. 创建一维数组

```
1  import numpy as np
2  # 创建一维数组
3  arr1 = np.array([1, 2, 3])
4  print(arr1)
```

运行结果：

```
[1 2 3]
```

2. 创建多维数组

```
1  import numpy as np
2  arr2 = np.array([[1, 2], [3, 4],[5, 6]])
3  print(arr2)
```

运行结果：

```
[[1 2]  [3 4]  [5 6]]
```

3. 数组基本运算

```
1  import numpy as np
2
3  arr2 = np.array([[1, 2], [3, 4],[5, 6]])
4  print("数组的维数:",arr2.ndim)
5  print("数组元素总个数",arr2.size)
6  print("元素类型",arr2.dtype)
```

运行结果：

```
数组的维数：2
数组元素总个数:6
元素类型:int32
```

4. 创建常数数组

```
1  import numpy as np
2
3  arr3 = np.zeros((2,3), dtype = float, order = 'C')
4  print("创建 0 数组:",arr3)
5
6  arr4 = np.ones([2,3], dtype = None, order = 'C')
7  print("创建 1 数组:",arr4)
```

代码说明：

- dtype: 表示数据类型，省略时默认为浮点数据类型。
- order: 有 C 和 F 两个选项，分别代表行优先和列优先。

运行结果：

```
创建 0 数组：[[0. 0. 0.]  [0. 0. 0.]]
创建 1 数组：[[1. 1. 1.]  [1. 1. 1.]]
```

5. 使用数值范围创建数组

创建从 1 开始到 10 终止，步长为 2 的浮点型数组。

```
1  import numpy as np
2
3  arr5 = np.arange(1,10,2,dtype=float)
4  print(arr5)
```

运行结果：

```
[1. 3. 5. 7. 9.]
```

6. 切片和索引

```
1  import numpy as np
2
3  arr6 = np.arange(20)
```

```
s = slice(1,20,3)   # 从索引 1 开始到索引 20 停止，步长为 3
print (arr6[s])

b = arr6[2:14:2]   # 从索引 2 开始到索引 14 停止，步长为 2
print(b)
```

运行结果：

```
[ 1  4  7 10 13 16 19]  [ 2  4  6  8 10 12]
```

7. 数组操作的其他方法

NumPy 数组的其他操作方法包括数组排序、数组逆序、最大最小值差、矩阵百分比、统计中位数、算术平均值、加权平均值、标准差、方差等，其操作也比较简单，这里不再逐一示例，请参考表 2-1 自行实现。

表 2-1 NumPy 数组的其他方法

方法描述	函　数	方法描述	函　数
数组排序	numpy.sort()	统计中位数	numpy.median()
数组逆序	numpy.argsort()	算术平均值	numpy.mean()
指定轴的最小值	numpy.amin()	加权平均值	numpy.average()
指定轴的最大值	numpy.amax()	标准差	numpy.std()
最大最小值差	numpy.ptp()	方差	numpy.var()
矩阵百分比	numpy.percentile()		

2.1.3 Numpy 的数学函数

1. 三角函数

```
import numpy as np

# 不同角度的正弦值
arr7 = np.array([0,45,90])
# 通过乘 pi/180 转化为弧度
print (np.sin(arr7*np.pi/180))
```

运行结果：

```
[0. 0.70710678 1. ]
```

2. 三角函数操作的其他方法

三角函数的操作方法包括正弦、余弦、正切、反正弦、反余弦、反正切等，实现方法与上述代码类同，请参考表 2-2 自行使用。

表 2-2　三角函数的操作方法

方法描述	函　　数	方法描述	函　　数
正弦	numpy.sin()	反正弦	numpy.arcsin()
余弦	numpy.cos()	反余弦	numpy.arccos()
正切	numpy.tan()	反正切	numpy.arctan()

3. 矩阵运算

```
1  import numpy as np
2
3  arr8 = np.arange(9, dtype = np.float_).reshape(3,3)
4  print ('第一个数组：\n',arr8)
5  arr9 = np.array([10,10,10])
6  print ('第二个数组：\n',arr9)
7  print ('两个数组之和：\n',np.add(arr8,arr9))
```

运行结果：

```
第一个数组：[[0. 1. 2.]  [3. 4. 5.]  [6. 7. 8.]]
第二个数组：[10 10 10]
两个数组之和：[[10. 11. 12.]  [13. 14. 15.]  [16. 17. 18.]]
```

4. 矩阵运算的其他方法

矩阵运算在数据预处理中的使用频率非常高，涉及的方法有相加、相减、相乘等，限于篇幅不能逐一举例，读者可以参加表 2-3 自行使用。

表 2-3　矩阵运算的其他方法

方法描述	函　　数	方法描述	函　　数
矩阵相加	numpy.add()	矩阵倒数	numpy.reciprocal() 如：1/2 的倒数为 2/1
矩阵相减	numpy.subtract()	矩阵求幂	numpy.power()
矩阵相乘	numpy.multiply()	矩阵取余	numpy.mod()
矩阵相除	numpy.divide()		

2.1.4 NumPy 线性代数运算

1. 点积

```
1  import numpy as np
2
3  arr10 = np.array([[1,2],[3,4]])
4  arr11 = np.array([[5,6],[7,8]])
5  print("数组的点积:\n",np.dot(arr10,arr11))
6  print("向量的点积:\n",np.vdot(arr10,arr11))
7  print ("向量的内积:\n",np.inner(arr10,arr11))
```

运行结果：

```
数组的点积:[[19 22]  [43 50]]
向量的点积:70
向量的内积:[[17 23]  [39 53]]
内积计算：15+26=17，17+28=23  35+46=39，37+48=53
```

2. 线性代数操作的其他方法

线性代数运算也是比较重要的知识，NumPy 使用频率也较高，具体函数如表 2-4 所示，读者可参照上述代码自行使用。

表 2-4 线性代数操作的其他方法

方法描述	函　数	方法描述	函　数
矩阵乘积	numpy.matmul()	矩阵线性方程解	numpy.linalg.solve()
矩阵的行列式	numpy.linalg.det()	逆矩阵	numpy.linalg.inv()

2.1.5 NumPy IO 操作

NumPy 为 ndarray 对象引入了一个简单的文件格式 npy，npy 文件用于存储重建 ndarray 所需的数据、图形、dtype 和其他信息。

1. 数组保存为 npy 文件

```
1  import numpy as np
2
3  arr12 = np.array([[17,23],[39,53]])
```

```
# 保存到 outfile.npy 文件中
np.save('outfile.npy',arr12)

#加载 npy 文件
arr13 = np.load('outfile.npy')
print (arr13)
```

运行结果：

```
[[17. 23.]  [39. 53.]]
```

2. 数组保存为 txt 文件

```
import numpy as np

np.savetxt('out.txt',arr12)
# 加载 txt 文件
arr14 = np.loadtxt('out.txt')
print(arr14)
```

运行结果：

```
[[17. 23.]  [39. 53.]]
```

2.2　SciPy

SciPy 是一个 Python 的常用科学计算工具包，本节主要介绍 SciPy 的安装和特点以及 SciPy Linalg、SciPy 文件操作、SciPy 插值、SciPy Ndimage 和 SciPy 算法优化等常用的内置模块（源代码见：Chapter2/ScipyDome.py）。

2.2.1　SciPy 的安装和特点

SciPy 官网（网址：https://www.scipy.org/）如图 2-3 所示，包含最优化、线性代数、积分、插值、特殊函数、快速傅里叶变换、信号处理和图像处理、常微分方程求解等模块。

图 2-3　SciPy 官网

1. SciPy 的安装

可使用以下 3 种方法安装 SciPy：

方法一　Anaconda 已经集成了 SciPy，读者可使用命令 conda list 在 DOS 环境下查看。

方法二　可以通过 pip 命令自动安装，请执行如下命令：pip install scipy。

方法三　在 GitHub 上下载 SciPy 源码（网址：https://github.com/numpy/numpy），然后在源码根目录下执行如下命令：python setup.py install。

安装完成后，检测是否成功。首先打开 Sublime，然后按 F6 键进入 Python 环境，最后输入 import sicpy，得到如图 2-4 所示的状态即表示安装成功。

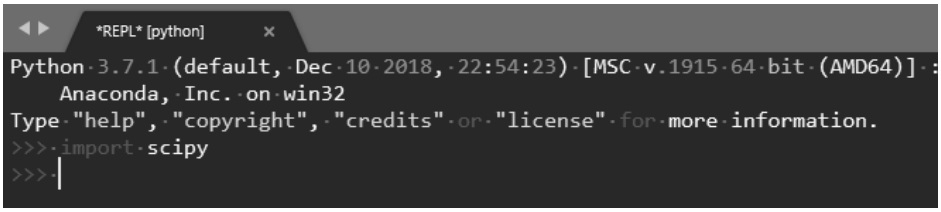

图 2-4　验证 SciPy 是否成功安装

2. SciPy 的特点

SciPy 库依赖于 NumPy，它提供了便捷且快速的 N 维数组操作。SciPy 库与 NumPy 数组一起工作，并提供了许多用户友好和高效的实现方法，它们一起运行在所有流行的操作系统上，安装快速且免费。

SciPy 提供了丰富的子模块，具体参见表 2-5 所示。

表 2-5　SciPy 的子模块

模块描述	子　模　块	模块描述	子　模　块
线性代数	scipy.linalg	向量量化	scipy.cluster
优化算法	scipy.optimize	数学常量	scipy.constants
矩阵线性方程解	scipy.sparse	数据输入输出	scipy.io
统计函数	scipy.stats	信号处理	scipy.signal
快速傅里叶变换	scipy.fftpack	特殊数学函数	scipy.special
积分	scipy.integrate	空间数据结构和算法	scipy.spatial
N 维图像	scipy.ndimage		

2.2.2　SciPy Linalg

SciPy Linalg 模块包含线性代数函数，使用该模块可以计算逆矩阵、求特征值、解线性方程组以及求行列式等。

1. 求解线性方程组

我们使用 Linalg 求解如下方程组：

```
3x+2y=2
x-y=4
5y+z=-2
```

完整的代码实现如下：

```
import numpy as np
from scipy import linalg

a = np.array([[3, 2, 0], [1, -1, 0], [0, 5, 1]])
b = np.array([2, 4, -2])
res1 = linalg.solve(a, b)
print ('线性方程组的解是：',res1)
```

代码说明：

数组 a 表示未知数的系数矩阵，数组 b 表示等号右边方程组值的矩阵。

运行结果：

```
线性方程组的解是： [ 2. -2. 8.]
```

2. 求解行列式

```
1  import numpy as np
2  from scipy import linalg
3
4  A = np.array([[1,2],[3,4]])
5  res2 = linalg.det(A)
6  print ('行列式的解是：',res2)
```

运行结果：

```
行列式的解是：14-23=-2.0
```

3. 求解特征值和特征向量

```
1  <pre>
2  import numpy as np
3  from scipy import linalg
4
5  A = np.array([[1,2],[3,4]])
6  λ, v = linalg.eig(A)
7  print ("特征值:\n",λ,"\n 特征向量:\n", v)
```

代码说明：

- A表示方阵。
- 特征值 λ 和相应的特征向量 v 的关系是：Av＝λv。

运行结果：

```
特征值:[-0.37228132+0.j  5.37228132+0.j]
特征向量:[[-0.82456484 -0.41597356]   [ 0.56576746 -0.90937671]]
```

4. 奇异值分解（SVD）

```
1  import numpy as np
2  from scipy import linalg
3
4  a = np.random.randn(2, 3) + 1.j*np.random.randn(2, 3)
5  U, s, Vh = linalg.svd(a)
6  print ('酉矩阵 1:\n',U,'\n 酉矩阵 2:\n', Vh,'\n 奇异值\n',s)
```

代码说明：

- 矩阵 a 分解为两个酉矩阵（U和Vh）和一个奇异值。
- a＝U * s* Vh。

运行结果：

```
酉矩阵 1：[[-0.55728327+6.19565045e-18j 0.83032244+1.53440325e-16j]
[-0.73499882+3.86279815e-01j -0.4933054 +2.59257451e-01j]]
酉矩阵 2：[[-0.01859058+0.2815052j 0.48875549-0.51402169j
-0.59175181-0.25911153j]  [ 0.05125373-0.51573816j
-0.43163592-0.32192764j -0.03326072-0.66357664j]  [-0.6951096
+0.41063251j -0.24183519-0.38531617j 0.37284682-0.04728662j]]
奇异值：[4.13890535 0.88993684]
```

2.2.3　SciPy 文件操作

1. 保存 MATLAB 文件

```
import numpy as np
import scipy.io as sio

#保存一个 matlab 文件
vect = np.arange(10)
sio.savemat('array.mat', {'vect':vect})
加载 matlab 文件
import numpy as np
import scipy.io as sio

#加载一个 matlab 文件
mat_file_content = sio.loadmat('array.mat')
print (mat_file_content)
```

运行结果：

```
{ '__header__': b'MATLAB 5.0 MAT-file Platform: nt, Created on: Tue
Feb 26 16:32:00 2019', '__version\_': '1.0', '__globals__': [],
'vect': array([[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]]) }
```

2. 查看 MATLAB 文件变量

```
import numpy as np
import scipy.io as sio

# 列出 MATLAB 文件中的变量
mat_file_content = sio.whosmat('array.mat')
print (mat_file_content)
```

运行结果：

```
[('vect', (1, 10), 'int32')]
```

2.2.4 SciPy 插值

插值是在直线或曲线上的两点之间找到值的过程。插值不仅适用于统计学，而且在科学、商业和需要预测两个现有数据点之间的值时也很有用。

1. 绘制二维空间图

```
import numpy as np
from scipy.interpolate import *
import matplotlib.pyplot as plt

# 两个维度空间点绘图
x = np.linspace(0, 4, 12)
y = np.cos(x**2/3+4)
print (x,y)
plt.plot(x, y,'o')
plt.show()
```

二维空间的余弦散列点运行结果如图 2-5 所示。

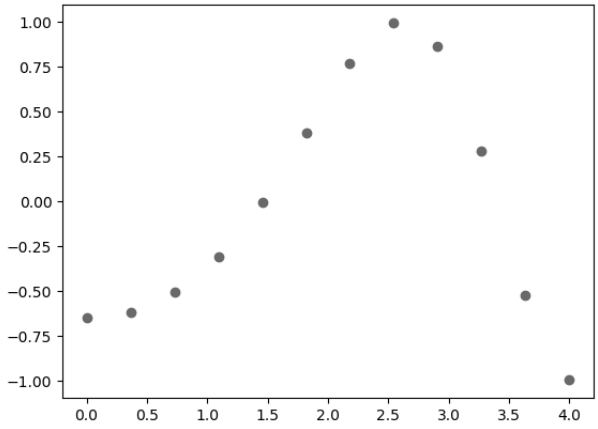

图 2-5　绘制二维空间图

2. 一维插值

一种基于固定数据点来推导出函数的便捷方法，可以使用线性插值在给定数据定义域内的任意位置推导出该函数。

```
import numpy as np
from scipy.interpolate import *
import matplotlib.pyplot as plt

f1 = interp1d(x, y,kind = 'linear')
f2 = interp1d(x, y, kind = 'cubic')
xnew = np.linspace(0, 4,30)
plt.plot(x, y, 'o', xnew, f1(xnew), '-', xnew, f2(xnew), '--')
plt.legend(['data', 'linear', 'cubic','nearest'], loc = 'best')
plt.show()
```

运行结果如图 2-6 所示。

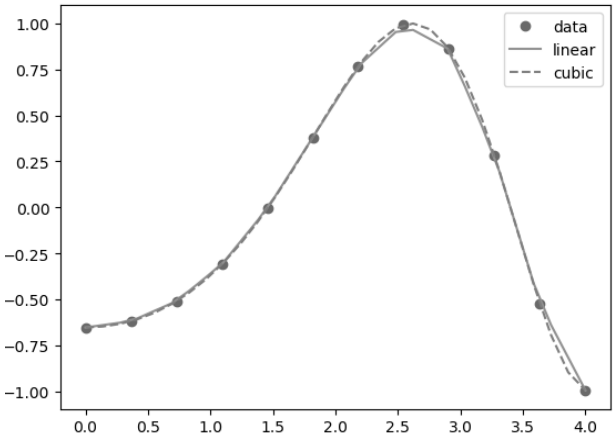

图 2-6　一维插值

3. 样条曲线

```
import numpy as np
from scipy.interpolate import *
import matplotlib.pyplot as plt

# 默认平滑参数
x = np.linspace(-3, 3, 50)
y = np.exp(-x**2) + 0.1 * np.random.randn(50)
plt.plot(x, y, 'ro', ms = 5)

# 手动更改平滑量
spl = UnivariateSpline(x, y)
```

```
xs = np.linspace(-3, 3, 1000)
spl.set_smoothing_factor(0.5)
plt.plot(xs, spl(xs), 'b', lw = 3)
plt.show()
```

运行结果如图 2-7 所示。

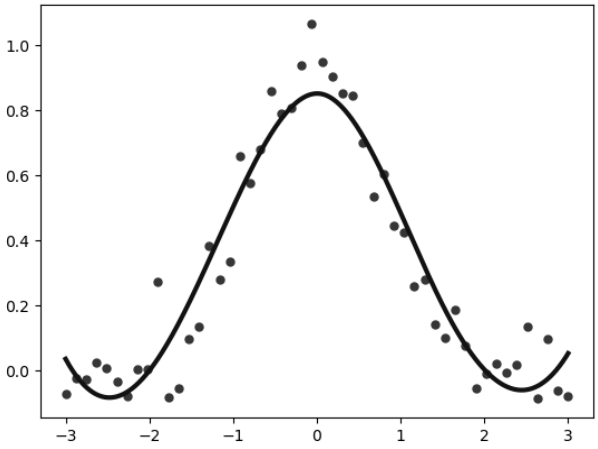

图 2-7　样条曲线插值

2.2.5　SciPy Ndimage

SciPy 的 ndimage 子模块专用于图像处理，ndimage 表示一个 n 维图像。图像处理中一些最常见的任务如下：

- 图像的输入、输出和显示。
- 基本操作，如裁剪、翻转、旋转等。
- 图像过滤，如去噪、锐化等。
- 图像分割，如标记对应于不同对象的像素。
- 分类。
- 特征提取。
- 注册。

下面使用 SciPy 实现其中的一些功能。

1. 打开图像

```
from scipy import misc,ndimage
import scipy.ndimage as nd
```

```
import numpy as np
import matplotlib.pyplot as plt

f = misc.face()
misc.imsave('face.png', f)
plt.imshow(f)
plt.show()
```

运行结果如图 2-8 所示。

图 2-8　打开图像

2. 图像倒置

```
from scipy import misc,ndimage
import scipy.ndimage as nd
import numpy as np
import matplotlib.pyplot as plt

face = misc.face()
flip_ud_face = np.flipud(face)
plt.imshow(flip_ud_face)
plt.show()
```

运行结果如图 2-9 所示。

图 2-9　图像倒置

3. 按指定的角度旋转图像

```
from scipy import misc,ndimage
import scipy.ndimage as nd
import numpy as np
import matplotlib.pyplot as plt

face = misc.face()
rotate_face = ndimage.rotate(face, 45)
plt.imshow(rotate_face)
plt.show()
```

运行结果如图 2-10 所示。

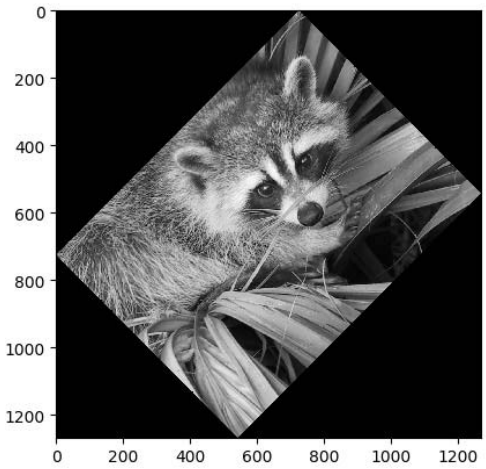

图 2-10　旋转图像

4. 模糊图像

```
from scipy import misc,ndimage
import scipy.ndimage as nd
import numpy as np
import matplotlib.pyplot as plt

face = misc.face()
blurred_face = ndimage.gaussian_filter(face, sigma=5)
plt.imshow(blurred_face)
plt.show()
```

代码说明：

sigma 值表示 5 级模糊程度。通过调整 sigma 值，可以看到图像质量的变化。

运行结果如图 2-11 所示。

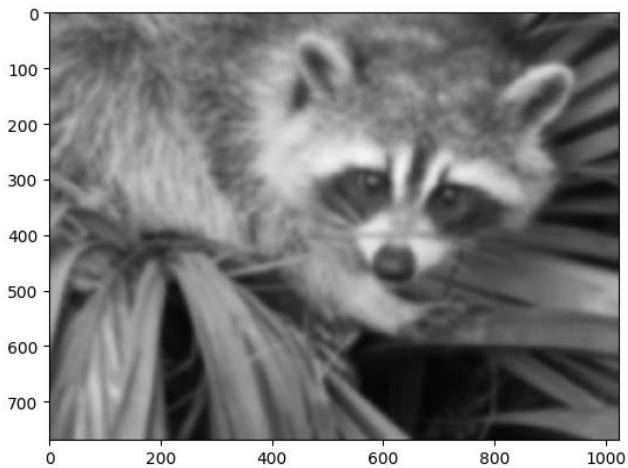

图 2-11　模糊图像

2.2.6　SciPy 优化算法

1. 梯度下降优化算法

求解方程：x^2-2x 的最小值，求解代码如下：

```
def f(x):
    return x**2-2*x

```

```
x = np.arange(-10, 10, 0.1)
plt.plot(x, f(x))
plt.show()
```

运行结果如图 2-12 所示。

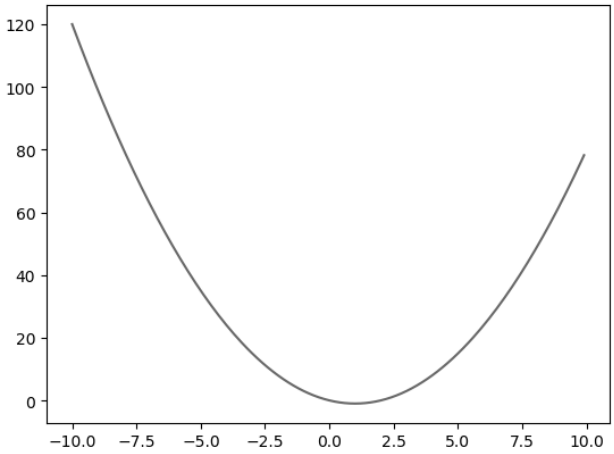

图 2-12　函数最小值

通过可视化结果可以发现，最小值比 0 略小。下面是梯度下降的实现代码：

```
from scipy import optimize

# 梯度下降优化算法
def f(x):
    return x**2-2*x
initial_x = 0
optimize.fmin_bfgs(f,initial_x)
```

运行结果：

```
Optimization terminated successfully.  Current function value:
-1.000000  Iterations: 2  Function evaluations: 9
Gradient evaluations: 3
```

结果显示最优值为-1，也满足可视化的判断。关于梯度下降算法的理论知识，读者可参考相关资料。

2. 最小二乘法优化算法

```
# 最小二乘法
from scipy.optimize import least_squares
def fun_rosenbrock(x):
    return np.array([10 * (x[1] - x[0]**2), (1 - x[0])])
input = np.array([2, 2])
res = least_squares(fun_rosenbrock, input)
print ('最小值是：',res)
```

运行结果：

```
最小值是： = [1.0,1.0]
```

本节展示了 SciPy 在优化算法等领域的应用，有关具体理论知识，读者可以自行查找资料学习。

2.3　Pandas

Pandas 也是一个常用的 Python 科学计算工具包，被广泛用于包括金融、经济、统计、分析等学术和商业领域的数据分析、数据清洗和准备等工作，是数据预处理的核心模块。使用 Pandas 可以完成数据处理和分析的 5 个典型步骤即数据加载、数据准备、操作、模型和分析。本节主要介绍 Pandas 的安装和特点以及 Pandas 的数据结构、数据统计、处理缺失值、稀疏数据、文件操作和可视化技术。

Pandas 的官网：https://pandas.pydata.org/，如图 2-13 所示。

图 2-13　Pandas 官网

（源代码见：Chapter2/PandasDome.py）

2.3.1 Pandas 的安装和特点

1. Pandas 的安装

可以通过以下 3 种方法安装 Pandas：

方法一 Anaconda 已经集成了 Pandas，读者在命令提示符环境下查看：conda list。

方法二 可以通过 pip 自动安装，执行如下命令：pip install pandas。

方法三 在 GitHub 上下载 Pandas 源码（网址：https://github.com/pandas-dev/pandas），然后在源码根目录下执行如下命令：python setup.py install。

安装完成后，检测是否成功。首先启动 Sublime，然后按 F6 键进入 Python 环境，输入 import pandas，得到以下状态即安装成功。如图 2-14 所示。

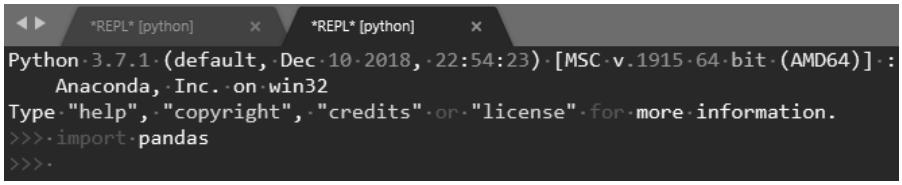

图 2-14 验证 Pandas 是否安装成功

2. Pandas 的特点

Pandas 快速高效、对缺失值自动处理及支持异构数据类型等诸多特点深受 Python 爱好者的喜欢，其主要特点包括：

- 快速高效的 DataFrame 对象，具有默认和自定义的索引。
- 将数据从不同文件格式加载到内存中的数据对象工具。
- 丢失数据的数据对齐和综合处理。
- 重组和摆动日期集。
- 基于标签的切片、索引和大数据集的子集。
- 可以删除或插入来自数据结构的列。
- 按数据分组进行聚合和转换。
- 高性能合并和数据加入。
- 时间序列功能。
- 具有异构类型列的表格数据，例如 SQL 表格或 Excel 数据。

2.3.2 Pandas 的数据结构

Pandas 可处理以下 3 种数据结构：

1. 系列（Series）

系列是具有均匀数据的一维数组结构。表 2-6 中的系列是整数 1,3,5,6...的集合。

表 2-6　一维数组

A	B	C	D	E	F
1	3	5	6	8	10

代码实现：

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

data = np.array(['a','b','c','d'])
index=[100,101,102,103]
obj = pd.Series(data,index)
print(obj)
```

代码说明：

代码中使用了 pandas.Series 方法处理一位数组，该方法共有 4 个参数，即 pandas.Series(data, index,dtype, copy)，各参数的说明如下：

- data：传入的数据参数，数据类型支持数组、列表等。
- index：索引，与数据的长度相同。
- dtype ：用于表示数据类型，省略是默认为浮点数据类型。
- copy：复制一份数据，默认为false。

运行结果：

```
100 a  101 b  102 c  103 d  dtype: object
```

2. 数据帧（DataFrame）

数据帧是一个具有异构数据的二维数组，如表 2-7 所示。

表 2-7　二维数组

姓　　名	年　　龄	性　　别	籍　　贯
张三	25	男	四川-成都
李四	22	男	河南-郑州
王五	25	女	陕西-西安

代码实现：

```
1  import pandas as pd
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  data = {'Name':['Tom','Jack','Steve','Ricky'],'Age':[28,34,29,42]}
6  df = pd.DataFrame(data, index=['rank1','rank2','rank3','rank4'],
   dtype=float)
7  print(df[:2])
```

代码说明：

代码中使用了 pandas.DataFrame 方法处理二维数据，该方法共有 5 个参数，即 pandas.DataFrame(data, index, columns, dtype, copy)，各参数说明如下：

- data：传入的数据参数，数据类型支持数组、列表、字典等。
- index：索引，与数据的长度相同。
- columns：列参数。
- dtype：用于表示数据类型，省略是默认为浮点型。
- copy：复制数据，默认为false（不复制）。

运行结果：

```
       Name   Age
rank1   Tom  28.0
rank2  Jack  34.0
```

3. 面板（Panel）

面板是具有异构数据的三维数据结构，在图形表示中很难表示出面板，但是一个面板可以说明为 DataFrame 的容器。

代码实现：

```
1  import pandas as pd
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  data = {'Item1' : pd.DataFrame(np.random.randn(4, 3)),
6          'Item2' : pd.DataFrame(np.random.randn(4, 2))}
7  p = pd.Panel(data)
8  print( p['Item1'])
```

代码说明：

代码中使用了 pandas.Panel 方法，该方法表示创建一个面板。该方法共有 6 个参数，即 pandas.Panel (data, items, majoraxis, minoraxis, dtype, copy)，各参数的含义如下：

- data：传入的数据参数，数据类型支持数组、列表、字典等。
- items: axis 0，每个项目对应于内部包含的数据帧（DataFrame）。
- major_axis: axis 1，表示每个数据帧（DataFrame）的行。
- minor_axis: axis 2 ，表示每个数据帧（DataFrame）的列。
- dtype：用于表示数据类型，省略是默认为浮点数据类型。
- copy：复制数据，默认为false（不复制）。

运行结果：

```
          0         1         2
0 -1.056665 -2.042454  0.761872
1 -0.973774 -1.263141  0.586314
2  0.463656 -0.786289 -0.791540
3 -1.533797 -0.891689  0.035241
```

2.3.3　Pandas 的数据统计

1. 利用 DataFrame 创建二维数组

代码实现：

```
1  import pandas as pd
2  import numpy as np
3
4  d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve',
   'Minsu','Jack','Lee','David','Gasper','Betina','Andres']),
5       'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),
6     'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,
   2.98, 4.80,4.10,3.65])}
7  df = pd.DataFrame(d)
8  print(df)
```

运行结果：

```
      Name   Age  Rating
0      Tom   25    4.23
1    James   26    3.24
2    Ricky   25    3.98
3      Vin   23    2.56
```

```
4    Steve   30    3.20
5    Minsu   29    4.60
6     Jack   23    3.80
7      Lee   34    3.78
8    David   40    2.98
9   Gasper   30    4.80
10  Betina   51    4.10
11  Andres   46    3.65
```

2. 数据统计

代码实现：

```
# 统计数据之和
print('数据之和:\n',df.sum())
# 统计数据的均值
print('数据的均值:\n',df.mean())
# 统计数据的标准偏差
print('数据的标准偏差:\n',df.std())
```

运行结果：

```
数据之和:
Name      TomJamesRickyVin...
Age       382
Rating    44.92
dtype: object

数据的均值:
Age       31.833333
Rating     3.743333
dtype: float64

数据的标准偏差:
Age       9.232682
Rating    0.661628
dtype: float64
```

Pandas 的其他统计方法，如表 2-8 所示。

表 2-8　Pandas 的其他统计方法

方法描述	函　　数	方法描述	函　　数
非空观测数量	pandas.count()	所有值中的最小值	pandas.min()
所有值之和	pandas.sum()	所有值中的最大值	pandas.max()
所有值的平均值	pandas.mean()	绝对值	pandas.abs()
所有值的中位数	pandas.median()	数组元素的乘积	pandas.prod()
值的模值	pandas.mode()	数组元素的累加和	pandas.cumsum()
值的标准偏差	pandas.std()	累计乘积	pandas.cumprod()

2.3.4　Pandas 处理丢失值

机器学习和数据挖掘等领域由于数据缺失导致的数据质量差，在模型预测的准确性上面临着严重的问题。为了提高模型效果可以使用 Pandas 对数据缺失值进行处理。

1. 构造一个含有缺失值的数据集

使用重构索引（Reindexing），创建有一个缺少值的 DataFrame。代码实现：

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e', 'f',
'h'],columns=['one', 'two', 'three'])
df = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])
print (df)
```

运行结果：

```
        one       two     three
a  0.749993 -0.524868  0.996772
b       NaN       NaN       NaN
c -2.129368 -1.000072  1.515642
d       NaN       NaN       NaN
e  1.617229 -0.046227 -1.003343
f  0.763989 -0.283858 -0.009689
g       NaN       NaN       NaN
h  1.246244  0.236078 -2.142169
```

在输出结果中，NaN 表示不是数字的值。

2. 检查缺失值

Pandas 提供了 isnull()和 notnull()函数，它们也是 Series 和 DataFrame 对象的方法，可用于检查缺失值，代码如下：

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e', 'f',
'h'],columns=['one', 'two', 'three'])
df = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])
print (df['one'].isnull())
```

运行结果：

```
    one    two  three
a  False  False  False
b   True   True   True
c  False  False  False
d   True   True   True
e  False  False  False
f  False  False  False
g   True   True   True
h  False  False  False
```

3. 清理/填充缺少数据

（1）用标量值 0 替换缺失值，其中 g 行表示缺少的数据。

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e', 'f',
'h'],columns=['one', 'two', 'three'])
df = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])
print ("NaN replaced with '0':")
print (df.fillna(0))
```

运行结果：

```
       one       two     three
a -0.514238  2.088999  0.432063
b  0.000000  0.000000  0.000000
c -1.252190  1.228565  0.593918
```

```
d  0.000000  0.000000  0.000000
e -0.356422 -0.316063  2.038978
f -1.124368  1.445957  0.080762
g  0.000000  0.000000  0.000000
h  0.492614 -0.259457 -0.935228
```

（2）用缺失值的前一行替换缺失值，例如 a,b 行，其中 b 为缺少数据。

实现代码如下：

```
1  import pandas as pd
2  import numpy as np
3
4  df = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e', 'f',
5  'h'],columns=['one', 'two', 'three'])
6  df = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])
7  print (df.fillna(method='pad'))
```

运行结果：

```
        one       two     three
a -1.643899  0.189767 -0.164862
b -1.643899  0.189767 -0.164862
c  1.016934  0.283606  0.906205
d  1.016934  0.283606  0.906205
e -0.046524 -0.060849 -0.046224
f -0.631722 -1.034078  0.416126
g -0.631722 -1.034078  0.416126
h  0.485579 -1.348188 -0.099870
```

（3）剔除缺失值。

```
1  import pandas as pd
2  import numpy as np
3
4  df = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e', 'f',
5  'h'],columns=['one', 'two', 'three'])
6  df = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])
7  print (df.dropna(axis=0))
```

代码说明：

- axis=0：表示作用在行上，如果axis=1，则表示作用在列上。

运行结果：

```
       one       two     three
a  0.617105  1.021967 -0.991709
c  0.238544 -0.100752 -0.131364
e  1.213844  0.091464  1.189781
f -0.061523 -0.229137 -0.032633
h -0.455861 -1.027256 -0.236553
```

（4）设置任意值替换缺失值，如 one 列的 2000 被 60 替换，two 列的 1000 被 10 替换。

代码如下：

```
import pandas as pd
import numpy as np

df = pd.DataFrame({'one':[10,20,30,40,50,2000],
'two':[1000,0,30,40,50,60]})
print (df.replace({1000:10,2000:60}))
```

运行结果：

```
   one  two
0   10   10
1   20    0
2   30   30
3   40   40
4   50   50
5   60   60
```

（5）忽略缺失值。

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e', 'f',
'h'],columns=['one', 'two', 'three'])
df = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])
print("df.dropna():\n{}\n".format(df.dropna()))
```

运行结果：

```
       one       two     three
a -1.183113  0.669814 -0.416021
c  0.478092 -0.065262  0.145993
e  0.006571 -1.060615 -1.099261
f -0.245773  0.596491 -0.213138
h -0.794740  0.252769  0.108073
```

2.3.5　Pandas 处理稀疏数据

当任何匹配特定值的数据（NaN/缺失值或任何值）被省略时，稀疏对象被“压缩”。

稀疏数据处理

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(10000, 4))
df.ix[:9998] = np.nan
sdf = df.to_sparse()
# 调用 to_dense 标准密集进行稀疏数据的处理
print ('稀疏数据集：\n',sdf.to_dense())
# 稀疏率
print ('稀疏率：\n',sdf.density)
```

运行结果：

稀疏数据集：

```
            0         1         2         3
0         NaN       NaN       NaN       NaN
1         NaN       NaN       NaN       NaN
2         NaN       NaN       NaN       NaN
3         NaN       NaN       NaN       NaN
4         NaN       NaN       NaN       NaN
...       ...       ...       ...       ...
9996      NaN       NaN       NaN       NaN
9997      NaN       NaN       NaN       NaN
9998      NaN       NaN       NaN       NaN
9999 -1.104343 -0.516193  1.156061 -0.328376
```

```
[10000 rows x 4 columns]
稀疏率：
 0.0001
```

2.3.6 Pandas 的文件操作

Pandas 库提供了一系列的 read_函数来读取各种格式的文件，这些 read_函数如下所示：

```
read_csv            read_table          read_fwf
read_clipboard      read_excel          read_hdf
read_html           read_json           read_msgpack
read_pickle         read_sas            read_sql
read_stata          read_feather
```

下面通过示例来说明部分函数的使用。

1. 操作 Excel 文件

```
1  import pandas as pd
2  import numpy as np
3
4  df1 = pd.read_excel("data/test1.xlsx")
5  print("df1:\n{}\n".format(df1))
```

代码说明：

本例表示使用 Pandas 的 read_excel 方法来处理 Excel 格式文件并打印输出 Excel 文件的内容。

运行结果：

```
df1:
      学号  姓名 性别  年龄
0  212001  张三  男    23
1  212002  李四  男    22
2  212003  王五  男    23
3  212004  赵六  男    21
4  212005  钱七  男    24
```

2. 操作 CSV 文件

```
1  import pandas as pd
2  import numpy as np
```

```
3
4  df2 = pd.read_csv("data/test2.csv", sep=",")
5  print("df2:\n{}\n".format(df2))
```

代码说明：

本例使用 Pandas 的 read_csv 方法来处理 CSV 格式文件并打印出文件的内容。

运行结果：

```
df2:
      学号  姓名 性别  年龄
0  212001  张三  男    23
1  212002  李四  男    22
2  212003  王五  男    23
3  212004  赵六  男    21
4  212005  钱七  男    24
```

3. 操作 JSON 文件

```
1  import pandas as pd
2  import numpy as np
3
4  # 转存 JSON 文件
5  df = pd.DataFrame([['a', 'b'], ['c', 'd']],index=['row 1', 'row
   2'],columns=['col 1', 'col 2'])
6  data_json=df.to_json(orient='columns')
7  print(data_json)
8
9  # 读取 json 文件
10 df3 = pd.read_json("data/test3.json")
11 print("df3:\n{}\n".format(df2))
```

代码说明：

本例首先将列表数据转化为 JSON 格式数据，然后调用 Pandas 的 read_json 方法处理 JSON 格式文件并打印输出内容。

运行结果：

```
JSON 文件
 {"col 1":{"row 1":"a","row 2":"c"},"col 2":{"row 1":"b","row 2":"d"}}
df3:
```

```
     学号  姓名 性别  年龄
0  212001  张三  男    23
1  212002  李四  男    22
2  212003  王五  男    23
3  212004  赵六  男    21
4  212005  钱七  男    24
```

2.3.7 Pandas 可视化

1. 绘制折线图

```
1  import matplotlib.pyplot as plt
2  import pandas as pd
3  import numpy as np
4
5  # 折线图
6  df = pd.DataFrame(np.random.randn(10,4), index=
7       pd.date_range('2019/2/27',periods=10), columns=list('ABCD'))
8  df.plot()
9  plt.show()
```

运行结果如图 2-15 所示。

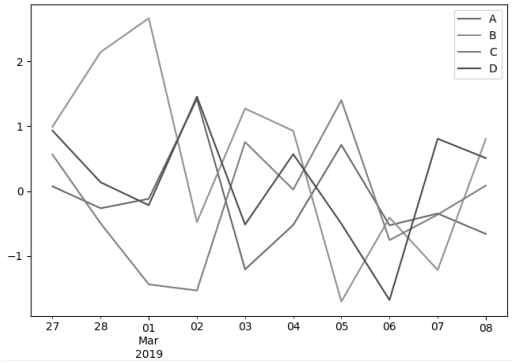

图 2-15 折线图

2. 绘制条形图

```
1  import matplotlib.pyplot as plt
2  import pandas as pd
3  import numpy as np
4
5  df = pd.DataFrame(np.random.rand(10,4),columns=['a','b','c','d'])
```

```
df.plot.bar()
plt.show()
```

运行结果如图 2-16 所示。

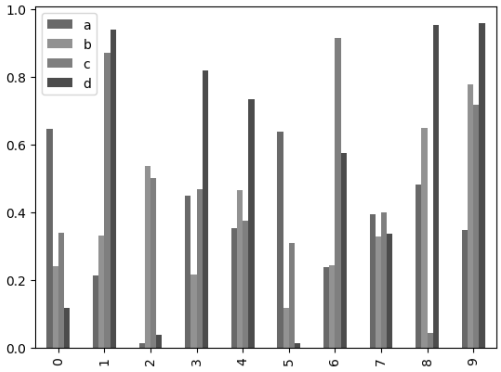

图 2-16　条形图

3. 绘制堆积条形图

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# 堆积条形图
df = pd.DataFrame(np.random.rand(10,4),columns=['a','b','c','d'])
df.plot.bar(stacked=True)
plt.show()
```

运行结果如图 2-17 所示。

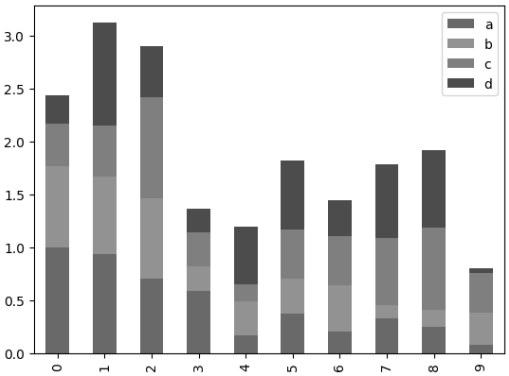

图 2-17　堆积条形图

4. 绘制水平条形图

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# 水平条形图
df = pd.DataFrame(np.random.rand(10,4),columns=['a','b','c','d'])
df.plot.barh(stacked=True)
plt.show()
```

运行结果如图 2-18 所示。

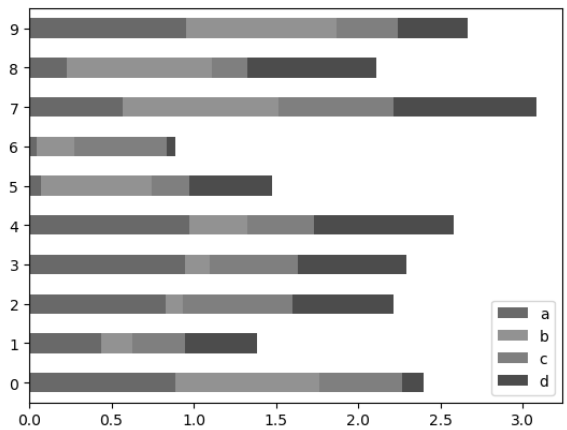

图 2-18　水平条形图

5. 绘制直方图

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# 直方图
df = pd.DataFrame({'a':np.random.randn(1000)+1,
'b':np.random.randn(1000),'c':
np.random.randn(1000) - 1}, columns=['a', 'b', 'c'])
df.plot.hist(bins=20)
plt.show()
```

运行结果如图 2-19 所示。

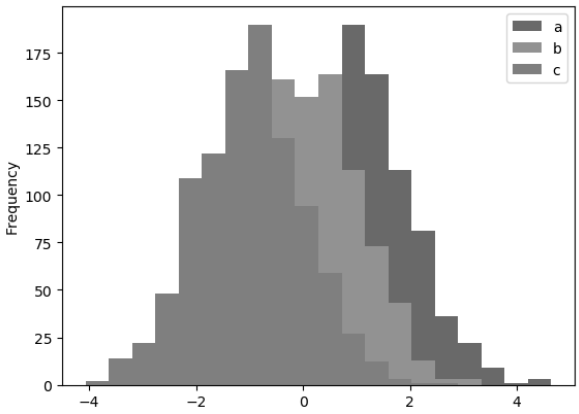

图 2-19　直方图

6. 绘制多个直方图

```
1  import matplotlib.pyplot as plt
2  import pandas as pd
3  import numpy as np
4
5  # 多个直方图
6  df=pd.DataFrame({'a':np.random.randn(1000)+1,'b':np.random.randn
   (1000),'c':
7  np.random.randn(1000) - 1}, columns=['a', 'b', 'c'])
8  df.hist(bins=20)
9  plt.show()
```

运行结果如图 2-20 所示。

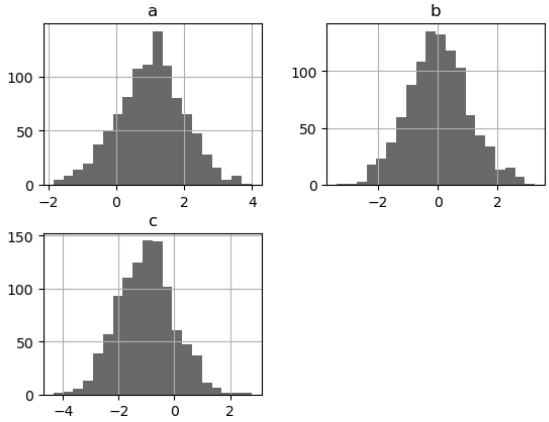

图 2-20　多个直方图

7. 绘制箱形图

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# 箱形图
df = pd.DataFrame(np.random.rand(10, 5), columns=['A', 'B', 'C', 'D',
'E'])
df.plot.box()
plt.show()
```

运行结果如图 2-21 所示。

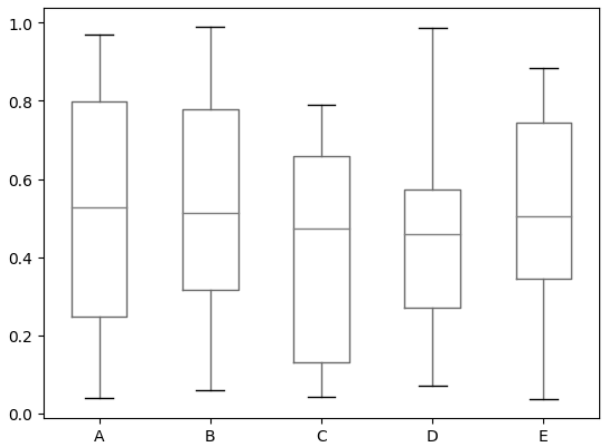

图 2-21　箱形图

8. 绘制区域块图形

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# 区域块图形
df = pd.DataFrame(np.random.rand(10, 4), columns=['a', 'b', 'c',
'd'])
df.plot.area()
plt.show()
```

运行结果如图 2-22 所示。

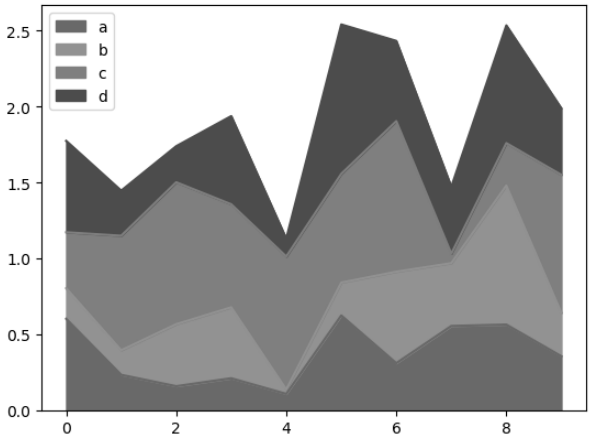

图 2-22　区域块图形

9. 绘制散点图形

```
1  import matplotlib.pyplot as plt
2  import pandas as pd
3  import numpy as np
4
5  # 散点图形
6  df = pd.DataFrame(np.random.rand(50, 4), columns=['a', 'b', 'c',
   'd'])
7  df.plot.scatter(x='a', y='b')
8  plt.show()
```

运行结果如图 2-23 所示。

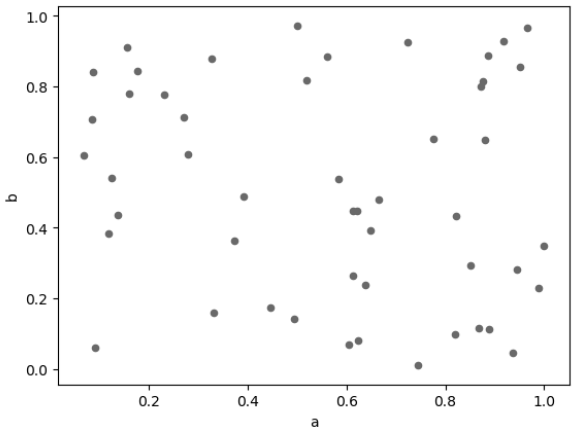

图 2-23　散点图形

10. 绘制饼状图

```
1  import matplotlib.pyplot as plt
2  import pandas as pd
3  import numpy as np
4
5  # 饼状图
6  df = pd.DataFrame(3 * np.random.rand(4), index=['A', 'B', 'C', 'D'],
   columns=['x'])
7  df.plot.pie(subplots=True)
8  plt.show()
```

运行结果如图 2-24 所示。

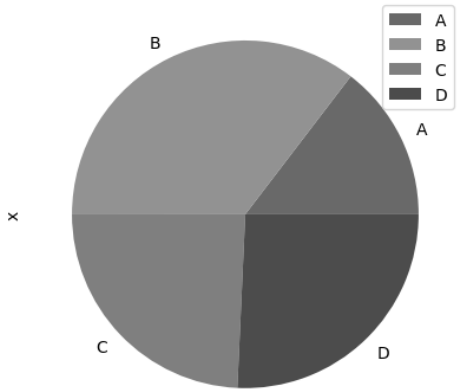

图 2-24　饼状图

2.4　本章小结

本章介绍了数据预处理中常用的 Python 科学工具包 NumPy、SciPy 和 Pandas，系统地学习了这些包的概念和使用。NumPy 工具包中的数学函数和线性代数知识，对于科研和阅读文献非常有帮助，IO 和数组操作也是在文本处理中常见的。SciPy 工具包常用于算法优化和实验结果分析等方面。Pandas 工具包对缺失值、稀疏值处理方面较为擅长。下一章将介绍数据采集与存储以及使用 Scrapy 实现爬取数据并存储在 MySQL 数据库中等知识。

第 3 章

数据采集与存储

随着网络和信息技术的不断普及，人类产生的数据量正在呈指数级增长，数据的形式也更加丰富，主要有结构化数据、半结构化数据、非结构化数据。面对各种形式的数据应当采用什么样的数据采集策略，如何实现网络爬虫爬取网页信息，如何对抓取到的网页信息进行本地化存储，都是数据预处理过程中经常会遇到的问题。本章从数据的分类入手，分别介绍数据采集和存储的常用方法与技术。

3.1 数据与数据采集

数据是指未经过处理的原始记录，如一堆杂志、一叠报纸、开会记录或整本病人的病历记录等，数据因缺乏组织和分类，是无法明确地表达事物代表的意义的。人工智能领域中的数据主要有 3 类：结构化数据、半结构化数据和非结构化数据，其表现形式不仅仅指文字，也包括图片、音频、视频等一系列可以存储知识的原始资料。

世界上每时每刻都在产生大量的数据，包括物联网传感器数据、社交网络数据、商品交易数据等。为了挖掘这些数据背后的价值，首先需要采集数据，因面对的场景的不同，采集数据的策略也会有差异。比如，针对关系型数据库中的数据、本地存储的文件、图片、音视频数据，直接拷贝即可；如果面对的是庞杂无序的网络数据，则需要采用网络爬虫技术进行处理了。接下来我们将从不同层面对数据的采集和存储方法进行介绍。

3.2 数据类型与采集方法

本节介绍大数据领域中三种主流的数据形式，即结构化数据、半结构化数据和非结构化数据，然后介绍常用的数据采集方法——爬虫技术。

3.2.1 结构化数据

结构化数据是指可以使用关系型数据库表示和存储，表现为二维形式的数据。一般特点是，数据以行为单位，一行数据表示一个实体的信息，每一行数据的属性是相同的。如表 3-1 所示。

表 3-1 二维数据表

Id	Name	Age	Gender
1	张三	21	男
2	李花	24	女
3	王五	22	男

- 数据特点：关系模型数据，关系数据库表示。
- 常见格式：如MySQL、Oracle、SQL Server格式等。
- 应用场合：数据库、系统网站、数据备份、ERP等。
- 数据采集：DB导出、SQL等方式。

结构化数据的存储和排列是很有规律的，这对查询和修改等操作很有帮助，但其扩展性较差。

3.2.2 半结构化数据

半结构化数据是结构化数据的一种形式，它并不符合关系型数据库或其他以数据表的形式关联起来的数据模型结构，其通过相关标记用来分隔语义元素以及对记录和字段进行分层。因此，它也被称为自描述的结构。半结构化数据同一类实体可以有不同的属性，但这些属性的顺序并不重要，即使它们被组合在一起。如 XML 格式文件的数据就是半结构数据的一种，以下是一个简单的示例：

```
<person>
    <name>李花</name>
    <age>13</age>
```

```
4      <gender>女</gender>
5  </person>
```

- 数据特点：非关系模型数据，有一定的格式。
- 常见格式：比如Email、HTML、XML、JSON格式等。
- 应用场合：邮件系统、档案系统、新闻网站等。
- 数据采集：网络爬虫、数据解析等方式。

不同的半结构化数据的属性的个数是不定的。有人说半结构化数据是以树或者图的数据结构存储的数据，如上面的例子中，<person>标签是树的根节点，<name>标签是子节点。这样的数据格式可以自由地表达很多有用的信息，如个人描述信息（元数据）等。可见，半结构化数据的扩展性是很好的。

3.2.3　非结构化数据

指没有固定结构的数据，各种文档、图片、视频/音频等都属于非结构化数据。对于这类数据可整体进行存储，一般存储为二进制的数据格式。如图 3-1 所示。

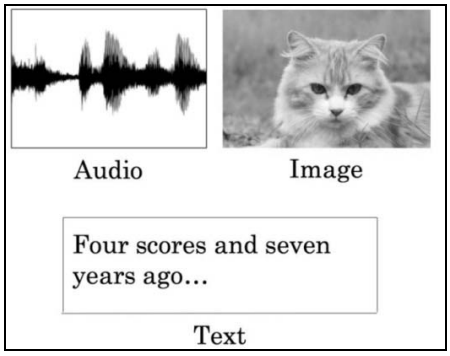

图 3-1　非结构化数据

- 数据特点：没有固定格式的数据。
- 常见格式：Word、PDF、PPT、图片、音视频等。
- 应用场合：图片识别、人脸识别、医疗影像、文本分析等。
- 数据采集：网络爬虫、数据存档等方式。

3.3　网络爬虫技术

网络数据采集是指通过网络爬虫或网站公开 API 等方式从网站上获取数据信息，

该方法可以将半结构化数据、非结构化数据从网页中抽取出来，将其存储为统一的本地数据，支持图片、音频、视频等数据采集。

3.3.1 前置条件

可以搭建以下环境，进行网络爬虫的开发：

- 软件系统：Windows 10 / Linux。
- 开发环境：Sublime Text3 / PyCharm。
- 数据库：MySQL 5.0 + Navicat 10.0.11。
- 编程语言：Python 3.7+Anaconda 4.4。
- 爬虫框架：Scrapy。
- 目标网站：http://blog.jobbole.com/all-posts/。

3.3.2 Scrapy 技术原理

Scrapy 是一个为爬取网站数据、提取结构化数据而设计的应用程序框架，通常我们可以很简单地通过 Scrapy 框架实现一个爬虫，抓取指定网站的内容或图片。

Scrapy 爬虫完整架构如图 3-2 所示（此图来源于网络），其中箭头线表示数据流向。

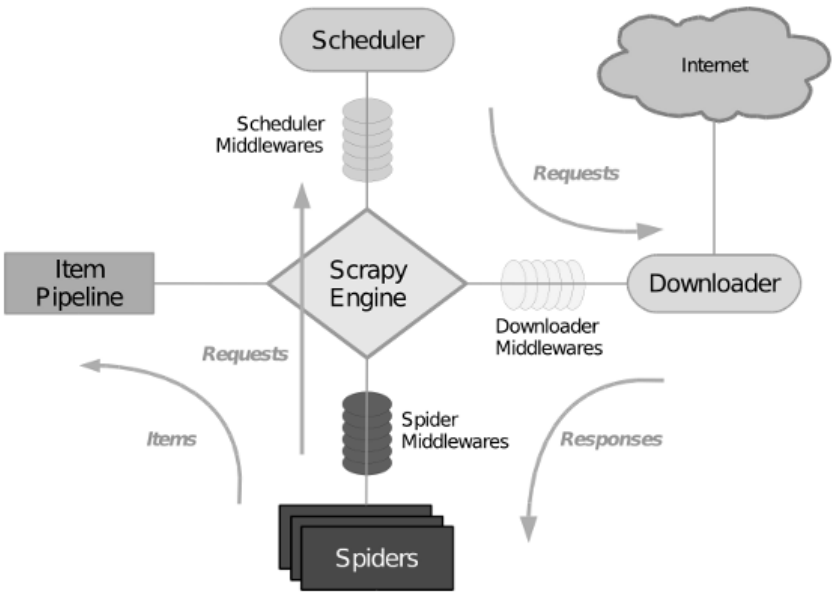

图 3-2 Scrapy 架构图

- Scrapy Engine（引擎）：负责Spider、ItemPipeline、Downloader、Scheduler中间的通信，信号和数据传递等。
- Scheduler（调度器）：它负责接受引擎发送过来的Request请求，并按照一定的方式进行整理排列和入队，当引擎需要时交还给引擎。
- Downloader（下载器）：负责下载Scrapy Engine（引擎）发送的所有Requests请求，并将其获取到的Responses（响应）交还给Scrapy Engine（引擎），由引擎交给Spider来处理。
- Spider（爬虫）：它负责处理所有的Responses，从中分析提取数据，获取Item字段需要的数据，并将需要跟进的URL提交给引擎，再次进入Scheduler（调度器）。
- Item Pipeline(管道)：它负责处理Spider中获取到的Item，并进行后期处理（详细分析、过滤、存储等）的地方。
- Downloader Middlewares（下载中间件）：一个可以自定义扩展下载功能的组件。
- Spider Middlewares（Spider中间件）：一个可以自定义扩展的操作引擎和Spider中间通信的功能组件（比如进入Spider的Responses和从Spider出去的Requests）。

3.3.3　Scrapy 新建爬虫项目

（1）安装 Scrapy 爬虫框架，按 WIN+R 组合键调出命令行环境，执行如下命令：

```
1  pip install scrapy
```

（2）按 WIN+R 组合键调出命令行环境进入根目录 Chapter3 文件夹下，创建爬虫项目：

```
1  scrapy startproject 项目名称
```

如图 3-3 所示。

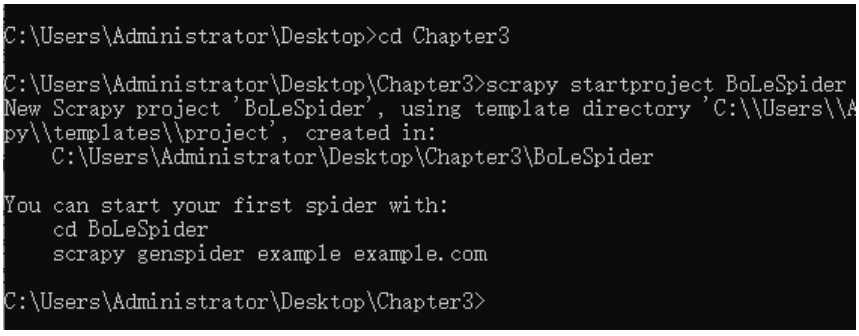

图 3-3　创建 BoLeSpider 项目

（3）BoLeSpider 为项目名称，可以看到将会创建一个名为 BoLeSpider 的目录，其目录结构大致如下：

```
BoLeSpider/
    scrapy.cfg
    BoLeSpider/
        __init__.py
        items.py
        pipelines.py
        settings.py
        spiders/
            __init__.py
            ...
```

下面简单介绍一下主要文件的作用，这些文件分别是：

- scrapy.cfg：项目的配置文件。
- BoLeSpider/：项目的Python模块，将会从这里引用代码。
- BoLeSpider/items.py：项目的目标文件。
- BoLeSpider/pipelines.py：项目的管道文件。
- BoLeSpider/settings.py：项目的设置文件。
- BoLeSpider/spiders/：存储爬虫代码的目录。

（4）在 BoLeSpider 项目下创建爬虫目录：

```
>> cd BoLeSpider
>> Scrapy genspider jobbole http://www.jobbole.com/
```

如图 3-4 所示。

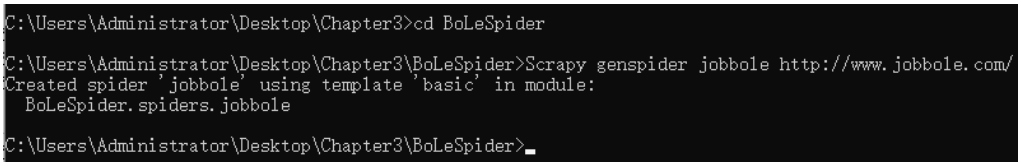

图 3-4　爬虫目标网站

（5）在同级目录下，执行如下命令，调用爬虫主程序。

```
scrapy crawl jobbole
```

如图 3-5 所示。

```
C:\Users\Administrator\Desktop\Chapter3\BoLeSpider>scrapy crawl jobbole
2019-03-25 10:27:01 [scrapy.utils.log] INFO: Scrapy 1.5.1 started (bot: BoLeSpider)
2019-03-25 10:27:01 [scrapy.utils.log] INFO: Versions: lxml 4.2.5.0, libxml2 2.9.8, cssselect 1.0.3,
parsel 1.5.1, w3lib 1.20.0, Twisted 18.9.0, Python 3.7.1 (default, Dec 10 2018, 22:54:23) [MSC v.1915
 64 bit (AMD64)], pyOpenSSL 18.0.0 (OpenSSL 1.1.1a  20 Nov 2018), cryptography 2.4.2, Platform Window
s-10-10.0.17134-SP0
2019-03-25 10:27:01 [scrapy.crawler] INFO: Overridden settings: {'BOT_NAME': 'BoLeSpider', 'NEWSPIDER
_MODULE': 'BoLeSpider.spiders', 'ROBOTSTXT_OBEY': True, 'SPIDER_MODULES': ['BoLeSpider.spiders']}
2019-03-25 10:27:02 [scrapy.middleware] INFO: Enabled extensions:
['scrapy.extensions.corestats.CoreStats',
 'scrapy.extensions.telnet.TelnetConsole',
 'scrapy.extensions.logstats.LogStats']
```

图 3-5　运行爬虫项目

（6）在 BoLeSpider 目录下创建 main.py：

```
# -*- coding: utf-8 -*-

import sys,os
from scrapy.cmdline import execute

sys.path.append(os.path.dirname(os.path.abspath(__file__)))
execute(["scrapy", "crawl", "jobbole"])  # scrapy crawl jobbole
```

执行 main.py，效果如图 3-6 所示。

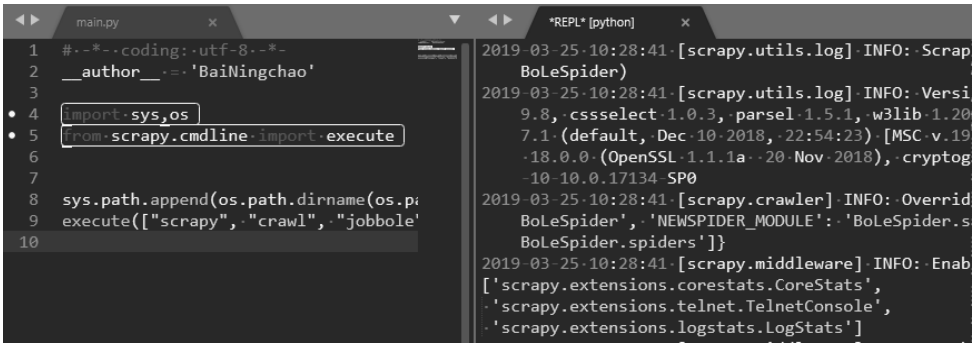

图 3-6　封装 main 函数运行爬虫项目

main.py 中的方法与在命令行环境下 scrapy crawl jobbole 的执行效果是一致的，之所以单独封装，是为了调试和运行的便利。

3.3.4　爬取网站内容

3.3.3 节完成了爬虫项目的构建，接下来主要做 4 个方面的工作：一是对项目进行相关配置；二是对目标爬取内容分析及提取文本特征信息；三是完成数据爬取工作；四是将数据进行本地化存储。

1. 爬虫项目配置

在做数据爬取工作时，如果不符合爬虫协议爬取工作将会中断。比如遇到 404 错误页面等情况，爬虫会自动退出。显然，这不符合对爬虫结果的期望，我们希望跳过错误页面继续爬取。因此，为了实现对不符合协议的网页继续爬取，需要打开 Scrapy 爬虫架构中的 setting.py 文件进行修改，具体修改如下：

```
ROBOTSTXT_OBEY = False。   # 此处的 True 改为 False
ITEM_PIPELINES = {
   'BoLeSpider.pipelines.BolespiderPipeline': 1,
}
```

2. 分析爬取的内容

在文章列表 http://blog.jobbole.com/all-posts/（实验时可正常访问，目前该官网处于关闭状态，读者重点明白原理及技术细节）中随机打开一篇文章，对单篇文章信息进行分析，并根据需求获取目标数据。假设需要获取文章的【新闻题目、创建时间、URL、点赞数、收藏数、评论数】，如图 3-7 所示。

图 3-7　分析特征数据

3. 爬取文章数据

（1）获取单篇文章的数据有两种方式，分别是基于 xpath 和 CSS 的方法。基于 xpath 的方法操作过程是，使用 http://blog.jobbole.com/114638/网址打开单篇文章，按 F12 键查看源代码，比如想获取文章，可以用光标选中题目的区域，并在右侧用鼠标右键单击源码处，再选中 Copy，继续单击“Copy Xpath”。此时的路径为：//*[@id="post-114638"]/div[1]/h1，如图 3-8 所示。

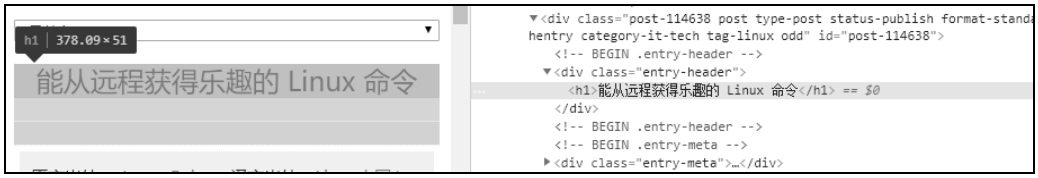

图 3-8　xpath 方法获取数据

（2）在命令行环境下执行以下 shell 命令：

```
cd BoLeSpider
scrapy shell http://blog.jobbole.com/114638/
```

（3）开始对每个特征进行测试，具体特征数据的测试代码如下：

```
 title = response.xpath('复制选中的 xpath 路径/text()').extract()
```

（4）完整的特征数据的提取结果如图 3-9 所示。

```
In [2]:  title = response.xpath('//*[@id="post-114638"]/div[1]/h1/text()').extract()

In [3]: title
Out[3]: ['能从远程获得乐趣的 Linux 命令']

In [4]:
```

图 3-9　测试特征数据提取文本信息

注　意

有时候按照以上方法操作，但却没有提取到文本信息，则有可能是代码错误，请仔细检查代码。也有可能是反爬虫技术的作用，此时，需要登录后再进行数据爬取，这属于更深层次的爬虫技术，本文不再涉及。

（5）使用 xpath 方法获取数据。

以上逐个特征测试无误后，将代码放在 Chapter3/BoLeSpider/ BoLeSpider/spiders/ jobbole.py 文件中的 parse 方法下：

```
# 获得单页的信息
def parse(self, response):
# xpath 获取内容
    title = response.xpath('//*[@id="post-114638"]/div[1]/h1/
text()').extract()   # 新闻题目
    create_date = response.xpath('//*[@id="post-114638"]/div[2]/p/
text()').extract()[0].strip().replace('·','')  # 创建时间
    url = response.url     # url
    dianzan = self.re_match(response.xpath('//*[@id="post-114638"]/
div[3]/ div[5]/span[1]/text()').extract()[1]) # 点赞数
```

```
    soucang = self.re_match(response.xpath('//*[@id="post-114638"]/
div[3]/ div[5]/span[2]/text()').extract()[0]) # 收藏数
    comment = self.re_match(response.xpath('//*[@id="post-114638"]/
div[3]/ div[5]/a/span/text()').extract()[0]) # 评论数
print('标题:',title,'\n','发布时间:',create_date,'\n','文章地址:',
url,'\n','点赞数: ',dianzan,'\n','收藏数',soucang,'\n','评论数',
comment)
```

（6）使用 CSS 方法获取数据。

```
 # 获得单页的信息
def parse(self, response):
    # css 获取内容
    title = response.css('.entry-header h1::text').extract()
    # 新闻题目
    create_date = response.css ('p.entry-meta-hide-on-
mobile::text').extract()[0].strip().replace('·','')  # 创建时间
    url = response.url    # url
    dianzan = self.re_match(response.css('.vote-post-up
h10::text').extract()[0]) # 点赞数
    soucang = self.re_match(response.css
('.bookmark-btn::text').extract()[0])
# 收藏数
    comment = self.re_match(response.css ('a[href=
"#article-comment"] span::text').extract()[0]) # 评论数
       print(title,'\n',create_date,'\n',url,'\n',dianzan, '\n',
soucang,'\n',comment)
```

（7）获取页面信息（源代码见：Chapter3/BoLeSpider/BoLeSpider/spiders/jobbole.py）。

```
# -*- coding: utf-8 -*-
import scrapy,re

# 获取单页信息
class JobboleSpider(scrapy.Spider):
    name = 'jobbole'
    allowed_domains = ['http://www.jobbole.com/']
    start_urls = ['http://blog.jobbole.com/114638']

    # 获得单页的信息
```

```
    def parse(self, response):
        # css 获取内容
        title = response.css('.entry-header h1::text').extract()
                # 新闻题目
        create_date = response.css ('p.entry-meta-hide-on-
mobile::text').extract()[0].strip().replace('·','')  # 创建时间
        url = response.url    # url
        dianzan = self.re_match(response.css('.vote-post-up
h10::text').extract()[0]) # 点赞数
        soucang = self.re_match(response.css
('.bookmark-btn::text').extract()[0]) # 收藏数
        comment = self.re_match(response.css
('a[href="#article-comment"] span::text').extract()[0]) # 评论数

        print('标题:',title,'\n','发布时间:',create_date,'\n',
              '文章地址:',url,'\n','点赞数: ',dianzan,'\n','收藏数',
              soucang,'\n','评论数',comment)

    # 对点赞数、收藏数、评论数等进行正则数字提取
    def re_match(self,value):
    match_value = re.match('.*?(\d+).*',value)
    if match_value:
        value = int(match_value.group(1))
    else:
        value = 0
    return value
```

（8）运行 main.py 函数，获取到所有的信息，如图 3-10 所示。

```
2019-03-25·11:06:37·[scrapy.extensions.telnet]·DEBUG:·Telnet·console·listening·
    on·127.0.0.1:6024
2019-03-25·11:06:37·[scrapy.downloadermiddlewares.redirect]·DEBUG:·Redirecting·
    (301)·to·<GET·http://blog.jobbole.com/114638/>·from·<GET·http://blog.
    jobbole.com/114638>
2019-03-25·11:06:37·[scrapy.core.engine]·DEBUG:·Crawled·(200)·<GET·http://blog.
    jobbole.com/114638/>·(referer:·None)
标题:·['能从远程获得乐趣的·Linux·命令']·
·发布时间:·2019/01/13··
·文章地址:·http://blog.jobbole.com/114638/·
·点赞数：·1·
·收藏数·3·
·评论数·1
```

图 3-10　获取单篇文章特征数据

4. 爬取列表页所有文章

（1）实现列表页所有文章信息的爬取工作

按 F12 键分析网页，找到下一页并获取所有列表页的文章链接。

执行以下代码获取所有列表页的文章链接，如图 3-11 所示。

```
scrapy shell http://blog.jobbole.com/all-posts/
response.css("#archive .floated-thumb .post-thumb a::attr(href)").
extract()
```

```
In [1]: response.css('#archive .floated-thumb .post-thumb a::attr(href)').extract()
Out[1]:
['http://blog.jobbole.com/114666/',
 'http://blog.jobbole.com/114663/',
 'http://blog.jobbole.com/114655/',
 'http://blog.jobbole.com/114648/',
 'http://blog.jobbole.com/114641/',
 'http://blog.jobbole.com/114638/',
 'http://blog.jobbole.com/114636/',
 'http://blog.jobbole.com/114633/',
 'http://blog.jobbole.com/114605/',
 'http://blog.jobbole.com/114630/',
 'http://blog.jobbole.com/114628/',
 'http://blog.jobbole.com/114610/',
 'http://blog.jobbole.com/114591/',
 'http://blog.jobbole.com/114589/',
 'http://blog.jobbole.com/114573/',
 'http://blog.jobbole.com/114570/',
 'http://blog.jobbole.com/114561/',
 'http://blog.jobbole.com/114551/',
 'http://blog.jobbole.com/114549/']

In [2]:
```

图 3-11　获取列表页所有文章的链接

（2）设置目标特征的实体类

打开 Scrapy 框架内置的 Chapter3/BoLeSpider/items.py 文件，设计爬虫目标特征的实体类（这里可以将爬虫目标特征的实体类作为数据库操作中实体类来理解）。代码如下：

```
# -*- coding: utf-8 -*-
# Define here the models for your scraped items
# See documentation in:
# https://doc.scrapy.org/en/latest/topics/items.html
import scrapy
from scrapy.loader.processors import MapCompose

class BolespiderItem(scrapy.Item):
    # define the fields for your item here like:
    # name = scrapy.Field()
```

```
    pass

# 设置提取字段的实体类
class JobBoleItem(scrapy.Item):
    title = scrapy.Field()         # 文章题目
    create_date = scrapy.Field()   # 发布时间
    url =  scrapy.Field()          # 当前文章 url 路径
    dianzan = scrapy.Field() # 点赞数
    soucang = scrapy.Field() # 收藏数
    comment = scrapy.Field() # 评论数
```

（3）修改代码文件 jobbole.py

首先将 BoLeSpider/spiders/jobbole.py 文件的 starturls 修改为列表页的路径，然后在 parse 方法中解析文章，并提取下一页交给 Scrapy 提供下载。parsesdetail 方法负责每一篇文章的下载，re_match 方法是使用正则表达式对文本信息数据进行处理。完整的代码如下：

```
import datetime
from scrapy.http import Request
from urllib import parse
from BoLeSpider.items import JobBoleItem

class JobboleSpider(scrapy.Spider):
    name = 'jobbole'
    allowed_domains = ['http://www.jobbole.com/']
    # start_urls = ['http://blog.jobbole.com/114638']
    start_urls = ['http://blog.jobbole.com/all-posts/'] # 所有页信息

    # 获取列表下所有页信息
    def parse(self, response):
        # 1. 获取文章列表中具体文章的 url 并交给解析函数进行字段的解析
        post_urls =
response.css("#archive .floated-thumb .post-thumb
a::attr(href)").extract()
        for post_url in post_urls:
            yield Request(url=parse.urljoin(response.url, post_url),
callback=self.parses_detail, dont_filter=True) # scrapy 下载

        #  2. 提取下一页并交给 scrapy 下载
```

```
20          next_url = response.css(".next.page-numbers::
    attr(href)").extract_first("")
21          if next_url:
22              yield Request(url=parse.urljoin(response.url, post_url),
    callback=self.parse, dont_filter=True)
23
24      # scrapy shell http://blog.jobbole.com/114638/
25      def parses_detail(self, response):
26          article_item =JobBoleItem()
27          article_item['title'] = response.css('.entry-header
    h1::text').extract()
28          article_item['create_date'] = date_convert(response.css
    ("p.entry-meta-hide-on-mobile::text").extract()[0].strip().replac
    e("·","").strip())
29          article_item['url'] = response.url
30          article_item['dianzan'] = re_match(response.css
    ('.vote-post-up h10::text').extract()[0])
31          article_item['soucang'] = re_match(response.css
    ('.bookmark-btn::text').extract()[0])
32          article_item['comment'] = re_match(response.css
    ('a[href="#article-comment"] span::text').extract()[0])
33          yield article_item
34
35
36  # *****************使用正则表达式对字段进行格式化处理***************
37
38  # 对点赞数、收藏数、评论数等用正则表达式进行数字的提取
39  def re_match(value):
40      match_value = re.match('.*?(\d+).*',value)
41      if match_value:
42          nums = int(match_value.group(1))
43      else:
44          nums = 0
45      return nums
46
47
48  # 对时间进行格式化处理
49  def date_convert(value):
50      try:
```

```
51          create_date = datetime.datetime.strptime(value,
    "%Y/%m/%d").date()
52      except Exception as e:
53          create_date = datetime.datetime.now().date()
54      return create_date
```

（4）运行 main.py

提取列表页数据，如图 3-12 所示。

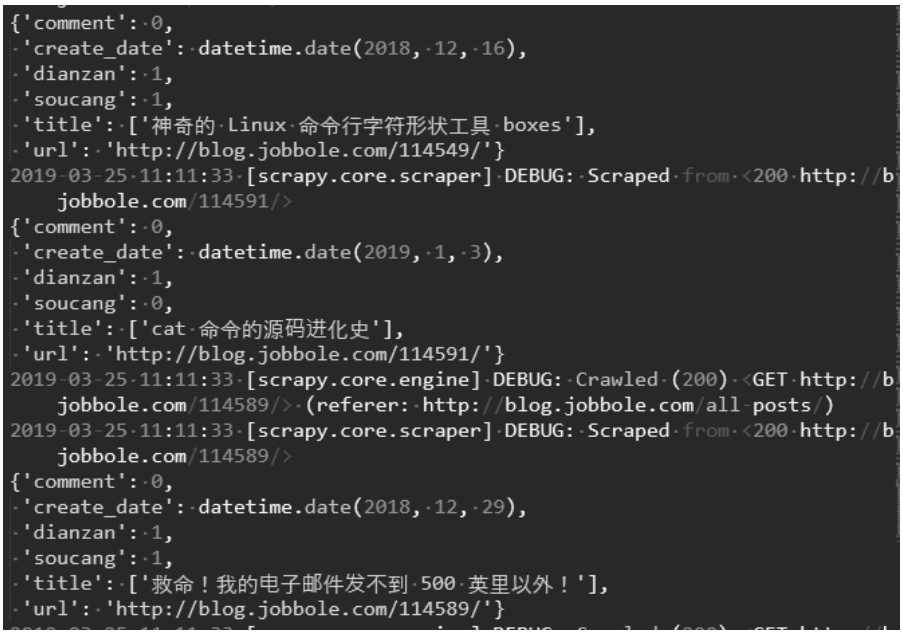

图 3-12　获取列表页所有文章的特征数据

3.4　爬取数据以 JSON 格式进行存储

上一节介绍了如何分析网页数据并爬取数据，得到数据以后如何进行存储呢？本节主要介绍 JSON 格式数据的本地化存储，具体操作步骤如下。

1. 修改管道文件

使用 pipline.py 文件作为管道文件，负责处理 Spider 中获取到的实体特征信息，并进行存储。这里需要导入 JsonItemExporter 模块进行 JSON 操作，然后在 JsonExporterPipleline 方法中执行具体的写操作，完整的代码如下：

```
from scrapy.exporters import JsonItemExporter
import codecs

class BolespiderPipeline(object):
    def process_item(self, item, spider):
        return item


# 调用 scrapy 提供的 json export 导出 json 文件
class JsonExporterPipleline(object):
    def __init__(self):
        self.file = open('articleexport.json', 'wb')
        self.exporter = JsonItemExporter(self.file,
encoding="utf-8", ensure_ascii=False)
        self.exporter.start_exporting()

    def close_spider(self, spider):
        self.exporter.finish_exporting()
        self.file.close()

    def process_item(self, item, spider):
        self.exporter.export_item(item)
        return item
```

2. 修改设置文件

在 setting.py 设置文件中修改方法的执行优先级。数字从小到大，数字越小，优先级就越高，具体设置如下：

```
ITEM_PIPELINES = {
   # 'BoLeSpider.pipelines.BolespiderPipeline': 1,
   'BoLeSpider.pipelines.JsonExporterPipleline': 1,
}
```

3. JSON 格式数据的本地化存储

运行 main.py 文件，实现本地 JSON 文件存储。执行完成后，打开 articleexport.json 文件查看结果，如图 3-13 所示。

图 3-13　JSON 格式数据的本地化存储

3.5　爬取数据的 MySQL 存储

相对于 JSON 存储，大家更为熟悉数据库存储，本节介绍爬取数据的本地数据库存储方法。本书选用的是通用的 MySQL 数据库，当然也可以选择 Oracle、SQL Server 等数据库。

3.5.1　MySQL 与 Navicat 部署

MySQL 原本是一个开放源码的关系数据库管理系统，原开发者为瑞典的 MySQL AB 公司，该公司于 2008 年被 Sun 公司收购。2009 年，Oracle 公司（甲骨文公司）收购了 Sun 公司，于是 MySQL 成为 Oracle 旗下的产品。

MySQL 由于性能高、成本低、可靠性好，已经成为最流行的开源数据库，被广泛地应用在 Internet 上的中小型网站中。随着 MySQL 的不断成熟，它也逐渐用于更多大规模网站和应用，比如维基百科、Google 和 Facebook 等网站。非常流行的开源软件组合 LAMP 中的“M”指的就是 MySQL。

Navicat 是香港卓软数码科技有限公司生产的一系列 MySQL、MariaDB、MongoDB、Oracle、SQLite、PostgreSQL 及 Microsoft SQL Server 的图形化数据库管理及开发软件。它有一个类似浏览器的图形用户界面，支持本地和远端数据库的多重连接。它的设计合乎各种用户的需求，这些用户包括数据库管理员，程序员，各种类型的个人客户，以及与合作伙伴共享信息的不同企业或公司。

由于 MySQL 与 Navicat 为大家所熟悉，基本安装和部署有关的资料较多，本书不再赘述。

3.5.2 MySQL 存储爬虫数据

1. 数据库表的设计

打开本地 MySQL 数据库，其中用户名为 root，密码为 root。成功登录后新建数据库名为 test，并在 test 数据库下新建 myarticles 数据表。具体设计如表 3-2 所示。

表 3-2 数据表设计

字 段 名	数据类型	是否主键	备 注
Id	int	是	新闻编号
Title	varchar	否	新闻题目
Createdata	datetime	否	创建时间
url	varchar	否	新闻网址
Dianzan	int	否	点赞数
Soucang	int	否	收藏数
Comment	int	否	评论数

数据表的设计如图 3-14 所示。

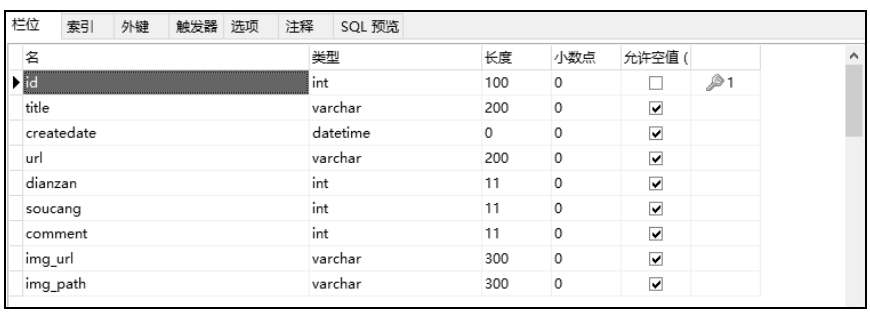

图 3-14 MySQL 数据库表的设计

2. 数据库存储的同步机制

打开 pipline.py 修改代码如下，本方法采用的是同步读写机制，即提取文件的同时执行写操作。首先建立本地数据库的连接并把中文编码格式设置为 UTF-8，之后执行一般的 SQL 插入操作。

```
1  # -*- coding: utf-8 -*-
2  from scrapy.exporters import JsonItemExporter
3
```

```
class BolespiderPipeline(object):
    def process_item(self, item, spider):
        return item

# 将爬取的数据字段存储在 MySQL 数据库中
import MySQLdb
import MySQLdb.cursors

class MysqlPipeline(object):
    #采用同步的机制写入 MySQL
    def __init__(self):
        self.conn = MySQLdb.connect('127.0.0.1', 'root', 'root',
'test', charset="utf8", use_unicode=True)
        self.cursor = self.conn.cursor()

    def process_item(self, item, spider):
        insert_sql = """
            insert into myarticles(title,
createdate,url,dianzan,soucang,comment) VALUES(%s,%s,%s,%s,%s,%s)
        """
        self.cursor.execute(insert_sql, (item["title"],
item["create_date"], item["url"],
item["dianzan"],item["soucang"],item["comment"]))
        self.conn.commit()
```

3. 数据库存储的异步机制

当爬取海量网络数据的时候，爬取速度与存储速度往往会产生冲突，采用前文介绍的数据库存储技术很可能会造成数据阻塞。基于此，需要改进数据存储方式，即采用异步存储机制。下面来介绍异步存储机制的代码实现。

首先在 pipline.py 文件中修改代码如下：

```
from twisted.enterprise import adbapi

class MysqlTwistedPipline(object):
    def __init__(self, dbpool):
        self.dbpool = dbpool
@classmethod
def from_settings(cls, settings): # cls 即 MysqlTwistedPipline
    dbparms = dict(
```

```
        host = settings["MYSQL_HOST"],
        db = settings["MYSQL_DBNAME"],
        user = settings["MYSQL_USER"],
        passwd = settings["MYSQL_PASSWORD"],
        charset='utf8',
        cursorclass=MySQLdb.cursors.DictCursor,
        use_unicode=True
    )
    dbpool = adbapi.ConnectionPool("MySQLdb", **dbparms)
    return cls(dbpool)

def process_item(self, item, spider):
    #使用 twisted 将 MySQL 插入变成异步执行
    query = self.dbpool.runInteraction(self.do_insert, item)
    query.addErrback(self.handle_error, item, spider) #处理异常

def handle_error(self, failure, item, spider):
    #处理异步插入的异常
    print (failure)

def do_insert(self, cursor,item):
    insert_sql = """
        insert into myarticles(title, createdate,url,dianzan,
soucang,comment,img_url,img_path) VALUES(%s,%s,%s,%s,%s,%s,%s,%s)
    """
    cursor.execute(insert_sql, (item["title"], item["create_date"],
item["url"], item["dianzan"], item["soucang"], item["comment"],
item["front_image_url"],item["front_image_path"]))
```

其次，在 setting.py 文件末行添加如下代码，用于实现全局变量的设置。

```
# 数据库设置
MYSQL_HOST = "127.0.0.1"
MYSQL_DBNAME = "test"
MYSQL_USER = "root"
MYSQL_PASSWORD = "admin"
```

4. 修改设置文件

在 setting.py 设置文件中修改方法的执行优先级。数字从小到大，数字越小，优先级就越高，具体设置如下：

```
ITEM_PIPELINES = {
    # 'BoLeSpider.pipelines.BolespiderPipeline': 1,
    # 'BoLeSpider.pipelines.JsonExporterPipleline': 1,
    'BoLeSpider.pipelines.MysqlPipeline': 1,
}
```

5. 本地化 MySQL 存储

运行 main.py 文件，实现本地 MySQL 数据存储。执行完成后，打开 myarticles 数据表查看，结果如图 3-15 所示。

id	title	createdate	url	dianzan	soucang	comment
81	在 Linux 上使用 tarball	2019-01-07	http://blog.jobbole.com/114628/	1	1	0
82	计算机科学自学指南	2018-12-22	http://blog.jobbole.com/114573/	2	16	1
83	微软变了！招程序员的流	2019-01-05	http://blog.jobbole.com/114610/	2	5	1
84	5 款 Linux 街机游戏	2019-01-11	http://blog.jobbole.com/114636/	1	0	0
85	Linux 搜索文件和文件夹	2018-12-19	http://blog.jobbole.com/114561/	1	0	0
86	救命！我的电子邮件发不	2018-12-29	http://blog.jobbole.com/114589/	1	1	0
87	追思杰出的 Linux 内核开	2019-01-08	http://blog.jobbole.com/114630/	2	1	1
88	cat 命令的源码进化史	2019-01-03	http://blog.jobbole.com/114591/	1	0	0
89	Vim 命令合集	2019-01-18	http://blog.jobbole.com/114641/	1	2	2
90	关于 top 工具的 6 个替代	2018-12-11	http://blog.jobbole.com/114546/	1	1	0
91	Python 中星号的本质及其	2019-02-16	http://blog.jobbole.com/114655/	1	2	1
92	神奇的 Linux 命令行字符	2018-12-16	http://blog.jobbole.com/114549/	1	1	0
93	能从远程获得乐趣的 Linu	2019-01-13	http://blog.jobbole.com/114638/	1	3	1
94	克劳德·香农（信息论之父	2019-02-16	http://blog.jobbole.com/114648/	1	2	1
95	学会这两件事，让你成为	2018-12-18	http://blog.jobbole.com/114551/	1	0	2
96	5 个好用的开发者 Vim 插	2019-02-24	http://blog.jobbole.com/114666/	1	0	0
97	14 个依然很棒的 Linux A	2019-02-19	http://blog.jobbole.com/114663/	1	1	1
98	在 Linux 命令行上拥有一	2018-12-21	http://blog.jobbole.com/114570/	1	0	0
99	从软件工程的角度解读任	2019-01-08	http://blog.jobbole.com/114605/	3	4	0

图 3-15　MySQL 存储文章数据

3.6　网络爬虫技术扩展

数据采集是一项庞杂的工作，倘若是文档文件或者数据库文件，采用拷贝和文件导出的方法即可完成。面对海量的非结构化文件，尤其是网络数据不可避免地会

选择网络爬虫技术。网络爬虫作为一门单独的学科领域，其涉及的知识非常深，仅仅这一项技术足够一本书去阐述，故而本书只是管中窥豹地介绍了爬虫技术，更多的网络爬虫技术难点包括：

- 实现网站虚拟登录并爬取数据。
- 网站反爬策略。
- 网站模板定期变动。
- 网站 URL 抓取失败。
- 网站频繁抓取 IP 被封。

3.7 本章小结

本章介绍了结构化、半结构化和非结构化数据及其数据的采集策略。面对非结构化网页信息，带领读者实现了页面分析和数据爬取，并把抓取的数据进行本地化存储。由于网络爬虫技术内容较多，本书篇幅有限，只是管中窥豹地介绍了网络爬虫技术与方法。下一章介绍文本信息抽取，即对采集的数据（包括 DOC、PDF、HTML、Excel 等）抽取文本信息。