CHAPTER 2

第 2 章

# NLP 前置技术解析

在本章中，你将学到 NLP 相关的一些前置技术。很多的数据科学库、框架、模块以及工具箱可以有效地实现 NLP 大部分常见的算法与技术，掌握与运用正则表达式，Numpy 是开始 NLP 工作的好方式。

本章的要点包括：

- 选择 Python 作为自然语言开发语言的理由
- 安装与使用 Anaconda
- 正则表达式
- Numpy

## 2.1 搭建 Python 开发环境

对于自然语言处理的学习，很多人会争论用什么样的编程语言实现最好？有些人认为是 Java 或者时下流行的 Scala，我认为 Python 才是最佳的选择！

对于学习和从事自然语言处理工作来说，Python 具有几大优势：

- 提供丰富的自然语言处理库。
- 编程语法相对简单（尤其易于理解）。

▼ 具有很多数据科学相关的库。

一般来说 Python 可以从 python.org（https://www.python.org）网站下载，但是对于没有任何 python 经验的读者来说，特别推荐安装 Anaconda（https://www.continuum.io/downloads）。对于初学者来说，Anaconda 使用起来特别方便，而且其涵盖了大部分我们需要的库。

### 2.1.1 Python 的科学计算发行版——Anaconda

Anaconda 是一个用于科学计算的 Python 发行版，支持 Linux、Mac、Windows 系统，它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安装问题。Anaconda 能让你在数据科学的工作中轻松安装经常使用的程序包。你还可以使用它创建虚拟环境，以便更轻松地处理多个项目。Anaconda 简化了工作流程，并且解决了多个包和 Python 版本之间遇到的大量问题。比如由于 Python 有 2.x 和 3.x 两个大的版本，有很多第三方库目前只支持 Python2.x 版本。

你可能已经安装了 Python，并且想知道为何还需要 Anaconda。首先，Anaconda 附带了一大批常用的数据科学包，因此你可以立即开始处理数据，而不需要使用 Python 自带的 pip 命令下载一大堆的数据科学包。其次，使用 Anaconda 的自带命令 conda 来管理包和环境能减少在处理数据过程中使用到的各种库与版本时遇到的问题。下面我们就来介绍下 conda。

#### conda

conda 与 pip 类似，只不过 conda 的可用包专注于数据科学，而 pip 应用广泛。然而，conda 并不像 pip 那样为 Python 量身打造，它也可以安装非 Python 包。它是任何软件堆栈的包管理器。话虽如此，不是所有的 Python 库都可以从 Anaconda 发行版和 conda 获得。你可以（并且今后仍然）使用 pip 和 conda 一起安装软件包。conda 安装预编译的软件包。例如，Anaconda 发行版带有使用 MKL 库编译的 Numpy、Scipy 和 Scikit-learn，它们加快了各种数学操作。这些软件包由供应商维护，这意味着它们通常落后于新版本。

但是对于需要为许多系统构建软件包的客户来说，这些软件包往往更稳定（并且更加方便使用）。

conda 的其中一个功能是包和环境管理器，用于在计算机上安装库和其他软件。conda 只能通过命令行来使用。因此，如果你觉得它很难用，可以参考面向 Windows 的命令提示符教程，或者学习面向 OSX/Linux 用户的 Linux 命令行基础知识课程。

安装了 Anaconda 之后，管理包是相当简单的。要安装包，请在终端中键入 conda install package_name。例如，要安装 Numpy，键入 conda install numpy：

```
conda install numpy
```

你可以同时安装多个包。类似 conda install numpy scipy pandas 的命令会同时安装所有这些包。你还可以通过添加版本号（例如 conda install numpy=1.10）来指定所需的包版本。

conda 还会自动为你安装依赖项。例如，scipy 依赖于 Numpy，因为它使用并需要 Numpy。如果你只安装 scipy（conda install scipy），则 conda 还会安装 Numpy（如果尚未安装的话）。

大多数命令都是很直观的。要卸载包，请使用 conda remove package_name。要更新包，请使用 conda update package_name。如果想更新环境中的所有包（这样做常常很有用），请使用 conda update --all。最后，要列出已安装的包，请使用前面提过的 conda list。

如果不知道要找的包的确切名称，可以尝试使用 conda search search_term 进行搜索。例如，我想安装 Beautiful Soup，但我不清楚确切的包名称。因此，我尝试执行 conda search beautifulsoup，如图 2-1 所示。

**提示：**conda 将几乎所有的工具、第三方包都当做 package 对待，因此 conda 可以打破包管理与环境管理的约束，能更高效地安装各种版本 Python 以及各种 package，并且切换起来很方便。

```
Fetching package metadata .............
beautifulsoup4                4.4.0                    py27_0  defaults
                              4.4.0                    py34_0  defaults
                              4.4.1                    py27_0  defaults
                              4.4.1                    py34_0  defaults
                              4.4.1                    py35_0  defaults
                              4.5.1                    py27_0  defaults
                              4.5.1                    py34_0  defaults
                              4.5.1                    py35_0  defaults
                              4.5.1                    py36_0  defaults
                              4.5.3                    py27_0  defaults
                              4.5.3                    py34_0  defaults
                              4.5.3                    py35_0  defaults
                            * 4.5.3                    py36_0  defaults
                              4.6.0                    py27_0  defaults
                              4.6.0                    py34_0  defaults
                              4.6.0                    py35_0  defaults
                              4.6.0                    py36_0  defaults
```

图 2-1　通过 conda 搜索 beautifulsoup

除了管理包之外，conda 还是虚拟环境管理器。它类似于另外两个很流行的环境管理器，即 virtualenv 和 pyenv。

环境能让你分隔用于不同项目的包。你常常要使用依赖于某个库的不同版本的代码。例如，你的代码可能使用了 Numpy 中的新功能，或者使用了已删除的旧功能。实际上，不可能同时安装两个版本的 Numpy。你需要为每个 Numpy 版本创建一个环境，然后在对应的环境中工作，这里再补充一下，每一个环境都是相互独立，互不干预的。

在自然语言处理中，我们需要大量的安装包，使用 Anaconda 无疑大大简化了管理包流程。

### 2.1.2　Anaconda 的下载与安装

本书写作的时候，Anaconda 的版本是 4.4.0，所包含的 Python 版本是 3.6，对于不同的操作系统下载不同的环境，目前以 Windows 版本作为例子讲解，如图 2-2 所示。

下载之后一般按照默认提示图形化安装即可，安装完毕之后，可以通过两种方式启动 Anaconda 的 Notebook：

▼ 在 Windows 开始菜单中找到 Anaconda，然后点击 Anaconda Prompt，输入 Jupyter Notebook 启动。

▼ 在 Windows 开始菜单中找到 Anaconda，然后点击 Jupyter Notebook 运行。在浏览器中会出现如图 2-3 所示的画面。

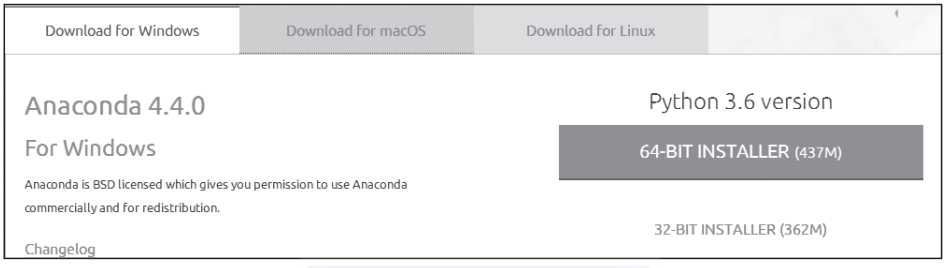


图 2-2　Anaconda 下载

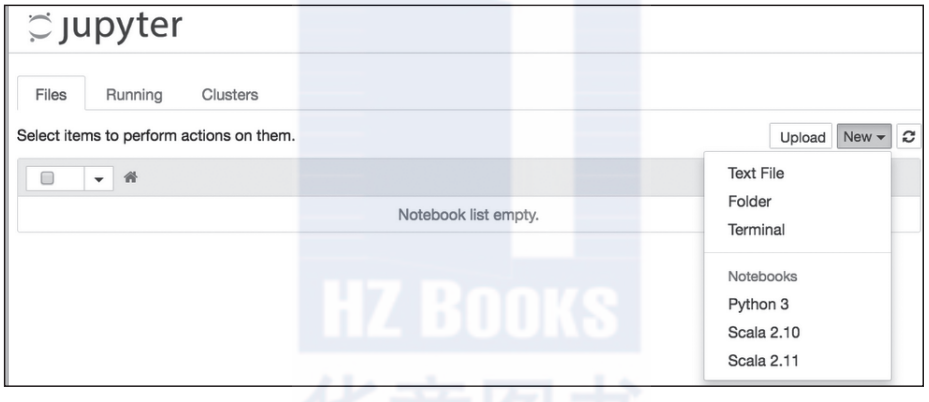


图 2-3　Jupyter Notebook 界面

通过右上角菜单 New—>Python3 新建一个编写代码的页面。然后我们在网页窗口中的“In”区域输入“1+1”，最后按“Shift”+“Enter”键，我们会看到 Out 区域显示为 2，这个就说明我们的 Anaconda 环境部署成功了，如图 2-4 所示。


图 2-4　环境测试界面

Jupyter Notebook 提供的功能之一就是可以让我们多次编辑 Cell（代码单元格）。在

实际开发当中，为了得到最好的效果，我们往往会对测试数据（文本）使用不同的技术进行解析与探索，因此 Cell 的迭代分析数据功能变得特别有用。

**延伸学习**

本节我们主要介绍了 Anaconda 的基本概念和使用方法，如果读者需要对 Anaconda jupyter notebook 有更深入的了解，可以访问官方文档：https://jupyter.readthedocs.io/en/latest/install.html。

## 2.2 正则表达式在 NLP 的基本应用

正则表达式是一种定义了搜索模式的特征序列，主要是用于字符串的模式匹配，或是字符的匹配。随着计算机的普及以及互联网的发展，大量的信息以电子文档方式呈现在人们的面前。NLP 通常所需要处理的语料一部分来自于 web 网页的信息抽取，一部分来自于文本格式的文档。Web 网页具有很强的开发价值，具有时效性强，信息量大，结构稳定，价值高等特点，文本格式的文档多来源于人为编写或系统生成，其中包含了非结构化文本、半结构化文本以及结构化文本。正则表达式的作用之一是将这些文档内容从非结构化转为结构化以便后续的文本挖掘。

正则表达式的另一个作用就是去除“噪声”。在处理大量文本片段的时候，有非常多的文字信息与最终输出的文本无关，这些无关的片段称之为“噪声”（比如 URL 或链接、语气助词、标点符号等）。

正则表达式是处理 NLP 的最基本的手段之一，学习与掌握正则表达式在 Python 中的应用，可以帮助我们在格式复杂的文本中抽取所需的文本信息。比如说抽取以下文本中的年份，每一行的格式不同，因此没有办法通过 Python 提供的字符串方法来抽取，这个时候我们往往考虑使用正则表达式。

```
- "July 16, 2017"
- "16/07/2009"
- "Summer 2008"
```

### 2.2.1 匹配字符串

在 Python 中，我们会使用 re 模块来实现正则表达式。为了让大家更好地理解正则表达式在 Python 中的应用，我们会通过一系列的例子来阐述。

案例中，我们会提到 re 的一个方法——re.search。

通过使用 re.search（regex，string）这个方法，我们可以检查这个 string 字符串是否匹配正则表达式 regex。如果匹配到，这个表达式会返回一个 match 对象，如果没有匹配到则返回 None。

我们先看下准备的有关爬虫介绍的文字信息。句子和句子之间是以句号分隔。具体的文本如下所示：

文本最重要的来源无疑是网络。我们要把网络中的文本获取形成一个文本数据库。利用一个爬虫抓取到网络中的信息。爬取的策略有广度爬取和深度爬取。根据用户的需求，爬虫可以有主题爬虫和通用爬虫之分。

**例 1　获取包含“爬虫”这个关键字的句子**

查找哪些语句包含“爬虫”这个关键字。Python 的代码实现如下：

```
import re
text_string = '文本最重要的来源无疑是网络。我们要把网络中的文本获取形成一个文本数据库。利用一个爬虫抓取到网络中的信息。爬取的策略有广度爬取和深度爬取。根据用户的需求，爬虫可以有主题爬虫和通用爬虫之分。'
regex = '爬虫'
p_string = text_string.split('。')# 以句号为分隔符通过 split 切分
for line in p_string:
    if re.search(regex,line) is not None: #search 方法是用来查找匹配当前行是否匹配这个 regex，返回的是一个 match 对象
        print(line) # 如果匹配到，打印这行信息
```

运行上面的程序，我们可以看到输出结果为：

利用一个爬虫抓取到网络中的信息

根据用户的需求，爬虫可以有主题爬虫和通用爬虫之分

**轮到你来：**尝试模仿上述代码，打印包含“文本”这个字符串的行内容。

### 例 2　匹配任意一个字符

正则表达式中，有一些保留的特殊符号可以帮助我们处理一些常用逻辑。如表 2-1 所示。

表 2-1　匹配任意一个字符

| 符号 | 含义 |
| --- | --- |
| . | 匹配任意一个字符 |

我们来举几个例子：

| 正则表达式 | 可以匹配的例子 | 不能匹配的例子 |
| --- | --- | --- |
| “a.c” | “abc”, “branch” | “add”, “crash” |
| “..t” | “bat”, “oat” | “it”, “table” |

**提示：**“.”代替任何单个字符（换行除外）

我们现在来演示下如何查找包含“爬”+任意一个字的句子。代码如下：

```
import re
text_string = '文本最重要的来源无疑是网络。我们要把网络中的文本获取形成一个文本数据库。利用一个爬虫抓取到网络中的信息。爬取的策略有广度爬取和深度爬取。根据用户的需求，爬虫可以有主题爬虫和通用爬虫之分。'
regex = '爬.'
p_string = text_string.split('。') # 以句号为分隔符通过 split 切分
for line in p_string:
    if re.search(regex,line) is not None: #search 方法是用来查找匹配当前行是否匹配这个 regex，返回的是一个 match 对象
        print(line) # 如果匹配到，打印这行信息
```

上述代码基本不变，只需要将 regex 中的“爬”之后加一个“.”即可以满足需求。我们来看下输出会多一行。因为不仅是匹配到了“爬取”也匹配到了“爬虫”。

利用一个爬虫抓取到网络中的信息

爬取的策略有广度爬取和深度爬取

根据用户的需求，爬虫可以有主题爬虫和通用爬虫之分

**轮到你来：**模仿上述程序，尝试设计一个案例匹配包含“用户”+任意一个字

### 例 3 匹配起始和结尾字符串

现在介绍另一个特殊符号，具体功能如表 2-2 所示。

表 2-2 匹配开始与结尾的字符串

| 符号 | 含义 |
| --- | --- |
| ^ | 匹配开始的字符串 |
| $ | 匹配结尾的字符串 |

举个例子：

▼“^a”代表的是匹配所有以字母 a 开头的字符串。

▼“a$”代表的是所有以字母 a 结尾的字符串。

我们现在来演示下如何查找以“**文本**”这两个字起始的句子。代码如下：

```
import re
text_string = '文本最重要的来源无疑是网络。我们要把网络中的文本获取形成一个文本数据库。利用一个爬虫抓取到网络中的信息。爬取的策略有广度爬取和深度爬取。根据用户的需求，爬虫可以有主题爬虫和通用爬虫之分。'
regex = '^文本'
p_string = text_string.split('。')
for line in p_string:
if re.search(regex,line) is not None:
print(line)
```

我们可以看到输出为：

文本最重要的来源无疑是网络

**轮到你来：**模仿上述程序，尝试设计一个案例匹配以“信息”这个字符串结尾的行。

### 例 4 使用中括号匹配多个字符

现在介绍另一个特殊符号，具体功能如表 2-3 所示。

表 2-3 匹配多个字符串

| 符号 | 含义 |
| --- | --- |
| [ ] | 匹配多个字符 |

举个例子：

“[bcr]at”代表的是匹配“bat”“cat”以及“rat”。

我们先看下文字信息。句子和句子之间以句号分隔。

▼ [ 重要的 ] 今年第七号台风 23 日登陆广东东部沿海地区。
▼ 上海发布车库销售监管通知：违规者暂停网签资格。
▼ [ 紧要的 ] 中国对印连发强硬信息，印度急切需要结束对峙。

我们希望提取以 [ 重要的 ] 或者 [ 紧要的 ] 为起始的新闻标题。代码如下：

```
import re
text_string = ['[ 重要的 ] 今年第七号台风 23 日登陆广东东部沿海地区 ',' 上海发布车库销售监管通知：违规者暂停网签资格 ','[ 紧要的 ] 中国对印连发强硬信息，印度急切需要结束对峙 ']
regex = '^\[[ 重紧 ]..\]'
for line in text_string:
    if re.search(regex,line) is not None:
        print(line)
    else:
        print('not match')
```

观测下数据集，我们发现一些新闻标题是以“[ 重要的 ]”“[ 紧要的 ]”为起始，所以我们需要添加“^”特殊符号代表起始，之后因为存在“重”或者“紧”，所以我们使用“[ ]”匹配多个字符，然后以“.”“.”代表之后的任意两个字符。运行以上代码，我们看到结果正确提取了所需的新闻标题。

```
[ 重要的 ] 今年第七号台风 23 日登陆广东东部沿海地区
not match
[ 紧要的 ] 中国对印连发强硬信息，印度急切需要结束对峙
```

### 2.2.2 使用转义符

上述代码中，我们看到使用了“\”为转义符，因为“[ ]”在正则表达式中是特殊符号。

与大多数编程语言相同，正则表达式里使用“\”作为转义字符，这就可能造成反斜杠困扰。假如你需要匹配文本中的字符“\”，那么使用编程语言表示的正则表达式里将需要 4 个反斜杠“\\\\”：前两个和后两个分别用于在编程语言里转义成反斜杠，转换成两个反斜杠后再在正则表达式里转义成一个反斜杠。Python 里的原生字符串很好地解决了这个问题，这个例子中的正则表达式可以使用 r“\\”表示。同样，匹配一个数字的“\\d”可以写成 r“\d”。有了原生字符串，你再也不用担心是不是漏写了反斜杠，写出来的表达式也更直观。

为了方便理解我们来举个例子：

```
import re
if re.search("\\\\","I have one nee\dle") is not None:
    print("match it")
else:
    print("not match")
```

通过上述例子，我们就可以匹配到字符串中的那个反斜杠”nee\dle”。为了简洁，我们可以换一个写法：

```
import re
if re.search(r"\\", "I have one nee\dle") is not None:
    print("match it")
else:
    print("not match")
```

通过加一个 r，我们就不用担心是否漏写了反斜杠。

### 2.2.3 抽取文本中的数字

#### 1. 通过正则表达式匹配年份

“[0-9]”代表的是从 0 到 9 的所有数字，那相对的”[a-z]”代表的是从 a 到 z 的所

有小写字母。我们通过一个小例子来讲解下如何使用。首先我们定义一个 list 分配于一个变量 strings，匹配年份是在 1000 年～ 2999 年之间。代码如下：

```
import re
strings = ['War of 1812', 'There are 5280 feet to a mile', 'Happy New Year 2016!']
for string in strings:
    if re.search('[1-2][0-9]{3}', string):# 字符串有英文有数字，匹配其中的数字部分，并且是在 1000 ~ 2999 之间，{3} 代表的是重复之前的 [0-9] 三次，是 [0-9] [0-9] [0-9] 的简化写法。
        year_strings.append(string)
print(year_strings)
```

### 2. 抽取所有的年份

我们使用 Python 中的 re 模块的另一个方法 findall() 来返回匹配带正则表达式的那部分字符串。re.findall（“[a-z]”，“abc1234”）得到的结果是 [ “a”，“b”，“c” ]。

我们定义一个字符串 years_string，其中的内容是‘2015 was a good year, but 2016 will be better!’。现在我们来抽取一下所有的年份。代码如下：

```
import re
years_string = '2016 was a good year, but 2017 will be better!'
years = re.findall('[2][0-9]{3}',years_string)
```

在 Anaconda 中执行这段语句，我们能看到输出 [‘2016’，‘2017’]。

**延伸学习**

关于 Python 的教程比比皆是，官方的教程（https://docs.python.org/3/tutorial/）是不错的入门选择。

## 2.3 Numpy 使用详解

Numpy（Numerical Python 的简称）是高性能科学计算和数据分析的基础包，提供了矩阵运算的功能。Numpy 提供了以下几个主要功能：

- ndarray——一个具有向量算术运算和复杂广播能力的多维数组对象。
- 用于对数组数据进行快速运算的标准数学函数。
- 用于读写磁盘数据的工具以及用于操作内存映射文件的工具。
- 非常有用的线性代数，傅里叶变换和随机数操作。
- 用于集成 C /C++ 和 Fortran 代码的工具。

除明显的科学用途之外，Numpy 也可以用作通用数据的高效多维容器，可以定义任意的数据类型。这些使得 Numpy 能无缝、快速地与各种数据库集成。

**提示**

这里提到的“广播”可以这么理解：当有两个维度不同的数组（array）运算的时候，可以用低维的数组复制成高维数组参与运算（因为 Numpy 运算的时候需要结构相同）。

在处理自然语言过程中，需要将文字（中文或其他语言）转换为向量。即把对文本内容的处理简化为向量空间中的向量运算。基于向量运算，我们就可以实现文本语义相似度、特征提取、情感分析、文本分类等功能。

本节 Numpy 的要点包括：

- 创建 Numpy 数组
- 获取 Numpy 中数组的维度
- Numpy 数组索引与切片
- Numpy 数组比较
- 替代值
- Numpy 数据类型转换
- Numpy 的统计计算方法

### 2.3.1 创建数组

在 Numpy 中，最核心的数据结构是 ndarray，ndarray 代表的是多维数组，数组指的

是数据的集合。为了方便理解，我们来举一个小例子。

（1）一个班级里学生的学号可以通过一维数组来表示：数组名叫 a，在 a 中存储的是数值类型的数据，分别是 1,2,3,4。

| 索引 | 学号 |
|---|---|
| 0 | 1 |
| 1 | 2 |
| 2 | 3 |
| 3 | 4 |

其中 a[0] 代表的是第一个学生的学号 1，a[1] 代表的是第二个学生的学号 2，以此类推。

（2）一个班级里学生的学号和姓名，则可以用二维数组来表示：数组名叫 b

| | |
|---|---|
| 1 | Tim |
| 2 | Joey |
| 3 | Johnny |
| 4 | Frank |

类似的，其中 b[0,0] 代表的就是 1（学号），b[0,1] 代表的就是 Tim（学号为 1 的学生的名字），以此类推 b[1,0] 代表的是 2（学号）等。

借用线性代数的说法，一维数组通常称为向量（vector），二维数组通常称为矩阵（matrix）。

当我们安装完 Anaconda 之后，默认情况下 Numpy 已经在库中了，所以不需要额外安装。我们来写一些语句简单测试下 Numpy：

1）在 Anaconda 中输入，如果没有报错，那么说明 Numpy 是正常工作的。

```
In [1]: import numpy as np
```

稍微解释下这句语句：通过 import 关键字将 Numpy 库引入，然后通过 as 为其取一

个别名 np，别名的作用是为了之后写代码的时候方便引用。

2）通过 Numpy 中的 array()，可以将向量直接导入：

```
vector = np.array([1,2,3,4])
```

3）通过 numpy.array() 方法，也可以将矩阵导入：

```
matrix = np.array([[1,'Tim'],[2,'Joey'],[3,'Johnny'],[4,'Frank']])
```

**轮到你来：**定义一个向量，然后分配于变量名 vector，定义一个矩阵然后分配给变量 matrix，最后通过 Python 中的 print 方法在 Anaconda 中打印出结果。

### 2.3.2 获取 Numpy 中数组的维度

首先我们通过 Numpy 中的一个方法 arange(n)，生成 0 到 $n-1$ 的数组。比如我们输入 np.arange(15)，可以看到返回的结果是 array（[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]）。

之后再通过 Numpy 中的 reshape（row,column）方法，自动构架一个多行多列的 array 对象。

比如我们输入：

```
a = np.arange(15).reshape(3,5)，代表 3 行 5 列
```

可以看到结果：

```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
```

我们有了基本数据之后，可以通过 Numpy 提供的 shape 属性获取 Numpy 数组的维度。

```
print(a.shape)
```

可以看到返回的结果，这个是一个元组（tuple），第一个 3 代表的是 3 行，第二个 5 代表的是 5 列：

```
(3, 5)
```

**轮到你来：**通过 arange 和 reshape 方法自己定义一个 Numpy 数组，最后通过 Python 中的 print 方法打印出数组的 shape 值（返回的应该是一个元组类型）。

### 2.3.3 获取本地数据

我们可以通过 Numpy 中 genfromtxt() 方法来读取本地的数据集。需要使用的数据集，house-prices.csv 是由逗号（,）分隔的，在 Github 的 data 目录下能下载到。我们可以使用以下语句来读取这个数据集：

```
import numpy as np
nfl = np.genfromtxt("D:/numpy/data/price.csv", delimiter=",")
print(nfl)
```

上述代码从本地读取 price.csv 文件到 Numpy 数组对象中（ndarray），我们看一下数据集的前几行。

```
[[              nan              nan              nan              nan
                nan              nan]
    [  1.00000000e+00  1.14300000e+05  2.00000000e+00  2.00000000e+00
                nan              nan]
    [  2.00000000e+00  1.14200000e+05  4.00000000e+00  2.00000000e+00
                nan              nan]
    [  3.00000000e+00  1.14800000e+05  3.00000000e+00  2.00000000e+00
                nan              nan]
    [  4.00000000e+00  9.47000000e+04  3.00000000e+00  2.00000000e+00
                nan              nan]
```

暂时先不用考虑返回数据中出现的 nan。

每一行的数据代表了房间的地区，是否是砖瓦结构，有多少卧室、洗手间以及价格的描述。

每个列代表了：

▼ Home：房子的 id

▼ Price：房子的价格

- ▼ Bedrooms：有多少个卧室
- ▼ Bathroom：有多少个洗手间
- ▼ Brick：是否是砖房
- ▼ Neighborhood：地区

**轮到你来：**使用 Numpy 的 genfromtxt 方法，读取 price.csv 文件并且命名为 home_price，然后通过 print 方法打印其类型及内容。

**注意：**Numpy 数组中的数据必须是相同类型，比如布尔类型（bool）、整型（int），浮点型（float）以及字符串类型（string）。Numpy 可以自动判断数组内的对象类型，我们可以通过 Numpy 数组提供的 dtype 属性来获取类型。

### 2.3.4 正确读取数据

回到之前的话题，上文发现显示出来的数据里面有数据类型 na（not available）和 nan（not a number），前者表示读取的数值是空的、不存在的，后者是因为数据类型转换出错。对于 nan 的出错，我们可以用 genfromtxt() 来转化数据类型。

- ▼ dtype 关键字要设定为‘U75’. 表示每个值都是 75byte 的 unicode。
- ▼ skip_header 关键字可以设置为整数，这个参数可以跳过文件开头的对应的行数，然后再执行任何其他操作。

```
    import numpy as np
    nfl = np.genfromtxt("d:/numpy/data/price.csv", dtype='U75', skip_header =
1,delimiter=",")
    print(nfl)
```

### 2.3.5 Numpy 数组索引

Numpy 支持 list 一样的定位操作。举例来说：

```
    import numpy as np
    matrix = np.array([[1,2,3],[20,30,40]])
    print(matrix[0,1])
```

得到的结果是 2。

上述代码中的 matrix[0,1]，其中 0 代表的是行，在 Numpy 中 0 代表起始第一个，所以取的是第一行，之后的 1 代表的是列，所以取的是第二列。那么最后第一行第二列就是 2 这个值了。

### 2.3.6 切片

Numpy 支持 list 一样的切片操作。

```
import numpy as np
matrix = np.array([
[5, 10, 15],
    [20, 25, 30],
    [35, 40, 45]
    ])
print(matrix[:,1])
print(matrix[:,0:2])
print(matrix[1:3,:])
print(matrix[1:3,0:2])
```

上述的 print（matrix[:,1]）语法代表选择所有的行，但是列的索引是 1 的数据。那么就返回 10，25，40。

print（matrix[:,0:2]）代表的是选取所有的行，列的索引是 0 和 1。

print（matrix[1:3,:]）代表的是选取行的索引值 1 和 2 以及所有的列。

print（matrix[1:3,0:2]）代表的是选取行的索引 1 和 2 以及列的索引是 0 和 1 的所有数据。

### 2.3.7 数组比较

Numpy 强大的地方是数组或矩阵的比较，数据比较之后会产生 boolean 值。

举例来说：

```
import numpy as np
matrix = np.array([
    [5, 10, 15],
[20, 25, 30],
[35, 40, 45]
])
m = (matrix == 25)
print(m)
```

我们看到输出的结果为：

```
[[False False False]
    [False  True False]
    [False False False]]
```

我们再来看一个比较复杂的例子：

```
import numpy as np
matrix = np.array([
[5, 10, 15],
[20, 25, 30],
[35, 40, 45]
    ])
second_column_25 = (matrix[:,1] == 25)
print(second_column_25)
print(matrix[second_column_25, :])
```

上述代码中 print（second_column_25）输出的是 [False True False]，首先 matrix[:,1] 代表的是所有的行，以及索引为 1 的列 –>[10,25,40]，最后和 25 进行比较，得到的就是 false,true,false。print（matrix[second_column_25, :]）代表的是返回 true 值的那一行数据 –> [20，25，30]。

---

**注意：**上述的例子是单个条件，Numpy 也允许我们使用条件符来拼接多个条件，其中“&”代表的是“且”，“|”代表的是“或”。比如 vector=np.array（[5,10,11,12]），equal_to_five_and_ten =（vector == 5）&（vector == 10）返回的都是 false，如果是 equal_to_five_or_ten =（vector == 5）|（vector == 10）返回的是 [True,True,False,False]

---

### 2.3.8 替代值

NumPy 可以运用布尔值来替换值。

在数组中：

```
vector = numpy.array([5, 10, 15, 20])
equal_to_ten_or_five = (vector == 10) | (vector == 5)
vector[equal_to_ten_or_five] = 50
print(vector)
[50, 50, 15, 20]
```

在矩阵中：

```
matrix = numpy.array([
[5, 10, 15],
[20, 25, 30],
[35, 40, 45]
])
second_column_25 = matrix[:,1] == 25
matrix[second_column_25, 1] = 10
print(matrix)
[[ 5 10 15]
    [20 10 30]
    [35 40 45]]
```

我们先创立数组 matrix。将 matrix 的第二列和 25 比较，得到一个布尔值数组。second_column_25 将 matrix 第二列值为 25 的替换为 10。

替换有一个很棒的应用之处，就是替换那些空值。之前提到过 NumPy 中只能有一个数据类型。我们现在读取一个字符矩阵，其中有一个值为空值。其中的空值我们很有必要把它替换成其他值，比如数据的平均值或者直接把他们删除。这在大数据处理中很有必要。这里，我们演示把空值替换为“0”的操作。

```
import numpy as np
matrix = np.array([
['5', '10', '15'],
['20', '25', '30'],
['35', '40','' ]
    ])
second_column_25 = (matrix[:,2] == '')
matrix[second_column_25, 2]='0'
print(matrix)
```

### 2.3.9 数据类型转换

Numpy ndarray 数据类型可以通过参数 dtype 设定，而且可以使用 astype 转换类型，在处理文件时这个会很实用，注意 astype 调用会返回一个新的数组，也就是原始数据的一份复制。

比如，把 String 转换成 float。如下：

```
vector = numpy.array(["1", "2", "3"])
vector = vector.astype(float)
```

**注意：**上述例子中，如果字符串中包含非数字类型的时候，从 string 转 float 就会报错。

### 2.3.10 Numpy 的统计计算方法

NumPy 内置很多计算方法。其中最重要的统计方法有：

- sum()：计算数组元素的和；对于矩阵计算结果为一个一维数组，需要指定行或者列。
- mean()：计算数组元素的平均值；对于矩阵计算结果为一个一维数组，需要指定行或者列。
- max()：计算数组元素的最大值；对于矩阵计算结果为一个一维数组，需要指定行或者列。

需要注意的是，用于这些统计方法计算的数值类型必须是 int 或者 float。

数组例子：

```
vector = numpy.array([5, 10, 15, 20])
vector.sum()
得到的结果是 50
```

矩阵例子：

```
matrix=
array([[ 5, 10, 15],
```

```
          [20, 10, 30],
          [35, 40, 45]])
matrix.sum(axis=1)
array([ 30,  60, 120])
matrix.sum(axis=0)
array([60, 60, 90])
```

如上述例子所示，axis = 1 计算的是行的和，结果以列的形式展示。axis = 0 计算的是列的和，结果以行的形式展示。

**延伸学习**

官方推荐教程（https://docs.scipy.org/doc/numpy-dev/user/quickstart.html）是不错的入门选择。

## 2.4 本章小结

工欲善其事，必先利其器。本章主要讲述了 NLP 工作者高效工作的一些“利器”：使用 Anaconda 快速构建开发环境，正则表达式快速进行字符串处理以及 Numpy 辅助进行科学计算。需要提醒读者的是，应重点关注正则表达式，因为在一些具体任务上，通常开端都是基于规则的方法最简单高效，而正则表达式正是实现这种规则最方便的方式，尤其是在以匹配为主的规则应用过程中。此外，章节篇幅有限，无法对一些诸如 pandas、SciPy 等常用 Python 库进行一一介绍，望读者自行查找相关资料，在入门 NLP 之前掌握一定的 Python 基础。