00A0 2203∃ 2200∀ 2286⊆ 2713x 27FA⟺ 221A$\surd$ 221B$\sqrt[3]{}$ 2295⊕ 2297⊗ UTF8gbsn

zsl-tools

发布 *0.2.0*

zhaisilong

2022 年 06 月 10 日

Contents

1 Intro **1**
- 1.1 Contributors . . . 1
- 1.2 Goals . . . 1
- 1.3 Wanted . . . 1

2 Awesome **3**
- 2.1 2022 03 . . . 3

3 Basic **5**
- 3.1 配置 . . . 5
- 3.2 Basic . . . 6
- 3.3 Basic . . . 32
- 3.4 合作 . . . 37
- 3.5 文档规范 . . . 37
- 3.6 Python Style . . . 37

4 Shell **41**
- 4.1 tmux . . . 41
- 4.2 Git . . . 42
- 4.3 Aria2 . . . 47
- 4.4 SSH . . . 49
- 4.5 System Control . . . 50
- 4.6 终端 . . . 51
- 4.7 综合实战 . . . 51

5 Python **53**
- 5.1 Python . . . 53
- 5.2 Python DataFrame . . . 63
- 5.3 Python Applications . . . 75
- 5.4 Python 网络 . . . 75
- 5.5 Python 效率 . . . 79
- 5.6 Python Shell . . . 80

6 Chem **85**
- 6.1 Fingerprints . . . 85
- 6.2 Representations . . . 85

7 Bio: Life Language Processing **89**
7.1 多肽 . . . 89

8 AI **91**
8.1 Prerequisites . . . 91
8.2 AutoML . . . 95
8.3 Transformers . . . 97
8.4 Hugging Face . . . 98
8.5 OpenNMT . . . 98
8.6 主动学习 . . . 98

9 人工智能最佳实践 **101**
9.1 PyTorch 基础 . . . 101
9.2 PyTroch 数据 . . . 102
9.3 PyTorch 配置与日志 . . . 106
9.4 日志 . . . 107
9.5 可视化 . . . 108
9.6 Torchtext . . . 109
9.7 Pytorch Lightning . . . 110
9.8 Metrics . . . 111

10 论文 **115**
10.1 论文写作 . . . 115

11 Books **119**
11.1 人工智能药物设计 . . . 119

12 Indices and tables **121**

CHAPTER 1

Intro

This is a brief introduction to Zhaisilong’s repo of all tools.

It can be your handbook of deep learning tools to solve problems in your field.

1.1 Contributors

- Zhaisilong

1.2 Goals

知识、命令查询手册

深度学习，CS 入门，进阶学习资料

优秀博文，网站，资源分享

1.3 Wanted

欢迎各位加入

请联系 zhaisilong@outlook.com

CHAPTER 2

Awesome

2.1 2022 03

2.1.1 踩坑记

20220318

```
export https_proxy=https://127.0.0.1:10809 http_proxy=http://localhost:10809
```

错误原因：`https_proxy=http://127.0.0.1:10809`

两个代理都应该是 `http scheme`

思考：`pip` 不支持 `sock5` 所以不能直接用 `export ALL_PROXY=socks5://127.0.0.1:10808`

CHAPTER 3

Basic

3.1 配置

3.1.1 YAML

YAML 是专门用来写配置文件的语言，非常简洁和强大，远比 JSON 格式方便

发音 /ˈjæməl/

语法特点

- 大小写敏感
- 使用缩进表示层级关系
- 缩进时不允许使用 Tab 键，只允许使用空格。
- 缩进的空格数目不重要，只要相同层级的元素左侧对齐即可
- 注释：#
- 数据结构
  - 对象：键值对的集合，又称为映射（mapping）/ 哈希（hashes）/ 字典（dictionary）
  - 数组：一组按次序排列的值，又称为序列（sequence）/ 列表（list）
  - 纯量（scalars）：单个的、不可再分的值

```yaml
# 纯量
number: 12.30
isSet: true # 注意是小写
parent: ~ # null
date: 1976-07-31
e: !!str 123 # 强制类型转换
f: !!str true
```

(下页继续)

(续上页)

```
str: 这是一行字符串
# 换行符会被转为空格
str: 这是一段
  多行
  字符串
# 多行字符串可以使用|保留换行符，也可以使用>折叠换行
this: |
  Foo
  Bar
that: >
  Foo
  Bar
# +表示保留文字块末尾的换行，-表示删除字符串末尾的换行
s1: |
  Foo
s2: |+
  Foo
s3: |-
  Foo
# 表示分文
---
```

参考

- YAML 语言教程

3.2 Basic

3.2.1 Markdown

- Markdown 教程

GFM

GitHub Flavored Markdown

GitHub 在 Markdown 的基础上引入了一些新特性

```
删除
~~我是被删除的内容~~

自动链接
在 GFM 中只要是合法的 HTTP 网址就可以自动被解析成一个有效的链接，可以省略标准␣
↪Markdown 中的尖括号 (<>)
我的博客是 https://zhaisilong.com

任务列表
- [ ] 待办事项1
- [ ] 待办事项2
- [x] 待办事项3
- [x] 待办事项4
```

(下页继续)

(续上页)

```
表格
姓名 | 年龄 | 性别
----|-----|----
小明 | 18 | 男
小刚 | 29 | 女
李三 | 20 | 男

Emoji表情支持
:grinning:
:heart_eyes:
:speech_balloon:
:peach:
![](https://cdn.jsdelivr.net/gh/xxzhai123/img/imgv2-6c8c3b801b1e79cafa0a8642b430e271_
↪720w.jpg)
![](https://cdn.jsdelivr.net/gh/xxzhai123/img/imgv2-6d6b1fe9e1ba08a7191fbf3e64617ae8_
↪r.jpg)

禁止了一些特殊的原生HTML标签
GFM 中会对这些特殊标签的左尖号 `<` 进行转义成 `&lt;`，从而让这些标签不起作用。

单词内部的下划线
在 GFM 语法中会忽略掉单词内部的下划线
为了避免混淆，尽量不要去使用下划线尽量去使用`*`

+ 你好
- 你不好
```

3.2.2 Latex

展示

1. 空格
2. if n is odd 文本

3. & 错位的对齐制表符 4.

$$f(n) = \begin{cases} 2n, & \text{if } n \text{ is even} \\ 3n, & \text{if } n \text{ is odd} \end{cases}$$

5. $\sim$

插入公式

行内公式

$y = x + 1$

独立公式

$$y = x + 1$$

上下标

$$x^{y^z} = (1 + e_1)^{xy^w}$$

括号

(、)、[、] 和 | 表示符号本身，使用 \\{来表示 {，\\} 来表示 }

$$f'(x) = (\frac{df}{dx})$$

通过将 '(,:raw-latex:right 与 ') 结合使用，可以将括号大小随着其内容变化。其他括号同理。

$$f'(x) = \left(\frac{df}{dx}\right)$$

$$f'(0) = \left.\frac{df}{dx}\right|_x = 0$$

分段函数

$$f(n) = \begin{cases} 2n, & \text{if } n \text{ is even} \\ 3n, & \text{if } n \text{ is odd} \end{cases}$$

运算表达式

分数

$$\frac{a}{b}$$

根号

$$\sqrt[a]{b}$$

求和

$$\sum_{a}^{b}$$

$$\prod_{a}^{b}$$

省略号

$$a+\ldots+b$$

$$a+\cdots+b$$

极限

$$lim_{a\to\infty}$$

求导

$$f'(x)$$

积分

$$\int_{a}^{b} x\,dx$$

希腊字母

1. α
2. β
3. ϵ

运算符

二元运算符

1. $+$
2. $*$
3. $-$
4. $\times$
5. $\div$
6. $\pm$
7. $\mp$
8. $\setminus$
9. $\approx$
10. $\odot$
11. $\geq$
12. $\leq$
13. $\%$
14. $\neq$
15. 30°

对数运算符

1. $\log$
2. $\lg$
3. $\ln$

逻辑运算符

1. $\ddots$
2. $\therefore$
3. $\forall$
4. $\exists$
5. $\not\subset$

箭头

1. $\Leftarrow$
2. $\Lleftarrow$
3. $\Rightarrow$
4. $\Longrightarrow$
5. $\Longleftrightarrow$
6. $\leftarrow$
7. $\longleftarrow$
8. $\rightarrow$
9. $\longrightarrow$
10. $\longleftrightarrow$
11. $\Longleftrightarrow$

向量

1. $\boldsymbol{x}$
2. $\mathbf{x}$
3. $\vec{x}$
4. x
5. $\overrightarrow{AB}$

其他

1. 空格
2. 两个空格
3. ◇ 表重要
4. ...
5. $\emptyset$
6. $\perp$
7. $\top$

源码

```
1. $\quad$ 空格
2. $\text{if n is odd}$ 文本
3. `&` 错位的对齐制表符
4. $$
   f(n) =
   \begin{cases}
    2n,  & \text{if $n$ is even} \\\\
    3n, & \text{if $n$ is odd}
   \end{cases}
   $$
5. $\sim$

## 插入公式

### 行内公式

> $y=x+1$

### 独立公式

> $$y=x+1$$

## 上下标

$$ x^{y^z}=(1+e_1)^{xy^w} $$

## 括号

`(`、`)`、`[`、`]` 和 `|` 表示符号本身，使用 `\\{`来表示 `{`，`\\}` 来表示 `}`
$$ {f}'(x) = ( \frac{df}{dx} ) $$
通过将 \left与( ,\right与) 结合使用，可以将括号大小随着其内容变化。其他括号同理。
$$ {f}'(x) = \left( \frac{df}{dx} \right) $$

$$ {f}'(0)=\left.\frac{df}{dx}\right|\_{x=0} $$

## 分段函数

$$
f(n) =
\begin{cases}
2n,  & \text{if $n$ is even} \\\\
3n, & \text{if $n$ is odd}
\end{cases}
$$

## 运算表达式

### 分数

$$\frac{a}{b}$$

### 根号

$$\sqrt[a]{b}$$
```

(下页继续)

(续上页)

```
### 求和

$$\sum_{a}^{b}$$

$$\prod_{a}^{b}$$

### 省略号

$${a+}\ldots{+b}$$
$${a+}\cdots{+b}$$

### 极限

$$lim_{{a} \to {\infty}}$$

### 求导

$${f}'(x)$$

### 积分

$$\int_{a}^{b}{x}\,dx$$

## 希腊字母

1. $\alpha$
2. $\beta$
3. $\epsilon$

## 运算符

### 二元运算符

1. $+$
2. $\ast$
3. $-$
4. $\times$
5. $\div$
6. $\pm$
7. $\mp$
8. $\setminus$
9. $\approx$
10. $\odot$
11. $\geq$
12. $\leq$
13. $\%$
14. $\neq$
15. $30^\circ$

### 对数运算符

1. $\log$
2. $\lg$
3. $\ln$

### 逻辑运算符
```

(下页继续)

(续上页)

```
1. $\because$
2. $\therefore$
3. $\forall$
4. $\exists$
5. $\not\subset$

## 箭头

1. $\Leftarrow$
2. $\Longleftarrow$
3. $\Rightarrow$
4. $\Longrightarrow$
5. $\Longleftrightarrow$
6. $\leftarrow$
7. $\longleftarrow$
8. $\rightarrow$
9. $\longrightarrow$
10. $\longleftrightarrow$
11. $\iff$

## 向量

1. $\boldsymbol{x}$
2. $\mathbf{x}$
3. $\vec{x}$
4. $x$
5. $\overrightarrow{AB}$

## 其他

1. $\quad$ 空格
2. $\qquad$ 两个空格
3. $\diamondsuit$ 表重要
4. $\dots$
5. $\emptyset$
6. $\bot$
7. $\top$
```

3.2.3 Read The Docs

Sphinx

Sphinx 是一个基于 Python 的文档生成项目，最早只是用来生成 Python 官方文档，随着工具的完善，越来越多的知名的项目也用他来生成文档，甚至完全可以用他来写书采用 `reStructuredText` 作为文档写作语言, 不过也可以通过模块支持其他格式，待会我会介绍怎样支持 `MarkDown` 格式。

Sphinx 提供的 `API documentation` 生成器称为 `sphinx-apidoc`

```
pip install sphinx sphinx-autobuild sphinx_rtd_theme
cd path_to_your_repo
sphinx-quickstart
```

```
Separate source and build directories (y/n) [n]:y
Project name: scrapy-cookbook
Author name(s): Xiong Neng
Project version []: 0.2
Project release [1.0]: 0.2.2
Project language [en]: zh_CN
```

编译以及本地服务器托管

```
make html
cd build/html
python3 -m http.server 8000
# 清除编译输出，会删除 build 下的文件
make clean
# 重新编译
make html
```

使用主题修改 conf.py

```
import sphinx_rtd_theme
html_theme = "sphinx_rtd_theme"
html_theme_path = [sphinx_rtd_theme.get_html_theme_path()]
```

运行构建

-b 选择了生成器，Sphinx 将会生成 HTML 格式

```
sphinx-build -b html sourcedir builddir
```

因为 sphinx-quickstart 生成了 Makefile 和 make.bat 文件，这些文件能够减少不少工作：有了它们你就可以运行

```
make html
```

配置 Sphinx 项目以获得 Markdown 支持

recommonmark —Recommonmark 0.7.1 documentation

```
pip install recommonmark
# 然后修改 conf.py
extensions = ['recommonmark']
```

支持 markdown 的表格，表格在本地可行但是无法**托管到 Read the docs 上**。因此，这里推介大家学一下 reStructure

```
pip install sphinx-markdown-tables
# 然后修改 conf.py
extensions = [
'sphinx_markdown_tables',
]
```

目前不支持 markdown 的数学公式

Markdown to reStructuredText with Pandoc

如果你熟悉 Markdown 写作。你可以使用 Pandoc 将成稿的 Markdown 转化为 reStructuredText

```
sudo apt install pandoc
pandoc -s test.md -o test.rst
```

FAQ

用 Read The Docs build 时 pdf 生成错误

参考 https://blog.seisman.info/trash/sphinx-chinese-support/

See also

- 使用 ReadtheDocs 托管文档

3.2.4 reStructuredText

reStructuredText 标记语言，相对 Markdown 来说，在写书方面更有优势:

- 使用 sphnix 能够自动生成目录和索引文件，方便查询和检索；
- 有大量漂亮的 HTML 书籍主题模版，可为书籍轻松换肤（类似 Wordpress 的网站模版）；
- 对于参考手册类书籍的编写在语法上更为便利（python 官方帮助文档的使用者）；

```
# 下划线及上划线表示 部分
* 划线及上划线表示 章节
= 下划线表示 小章节
- 下划线表示 子章节
^ 下划线表示 子章节的子章节
" 下划线表示 段落
```

```
*emphasis*
**emphasis**
`interpreted text`
``inline literal``
:sub:`xxx`
:sup:`xxx`
:guilabel:`Action`
:kbd:`Ctrl+Shift`
:menuselection:`A-->B-->C`
```

列表

```
- 无序
1. 有序
#. 自动编号
```

定义列表

术语的定义

```
term
    术语定义必须缩进

    可以包含多个段落

next term
    术语描述
```

分块

```
| These lines are
| broken exactly like in
| the source file.
```

源代码

注意大多数时候，空格代替一个换行符。所有有些时候换行是必要的。比如 `.. code-block::` 下面接了一句普通的话。块和半句是不能放在一起的。

标记符号 `::` 紧接一空白行（这个空行是必要的），然后紧跟代码，整个代码文本块必须缩进这里的如:

```
::

    some codes
    some codes
    some codes

此行上面的空行也是必要的
```

显式标记块的第一行是以 `..` 开始，接着是紧随着空格，被结束于同样层级缩进的下一段落。（显式标记和正常的段落之间需要有一个空行。当你写它的时候，可能听起来有点复杂，但它是直观的。）

高级的代码高亮功能

```
此行下面的空行也是必要的

.. code-block:: python
   :caption: Code Blocks can have captions.
   :linenos:
   :emphasize-lines: 3,5

   def some_function():
       interesting = False
       print 'This line is highlighted.'
```

(下页继续)

(续上页)

```
        print 'This one is not...'
        print '...but this one is.'
```

快速定义代码块

```
.. highlight:: sh
        此指令后如下的“::”定义的块都会以 sh 语法进行高亮，直到遇到下一条 highlight␣
↪指令。

::
        # 此命令在主机执行
        sudo apt install python
        echo "helloworld,this is a script test!"
```

literalinclude 直接嵌入本地文件并高亮

```
.. literalinclude:: ../../base_code/hello.c
        :caption: ../../base_code/hello.c
        :language: c
        :emphasize-lines: 5,7-12
        :linenos:
        :name: hello.c
```

嵌入文件的某部分

```
.. literalinclude:: ../../base_code/hello.c
        :caption: ../../base_code/hello.c
        :language: c
        :linenos:
        :lines: 1,3,5-8,20-
```

python 模块的 include。他有选择性的选取 timer.start

```
.. literalinclude:: example.py
   :pyobject: Timer.start
```

文件对比

```
.. literalinclude:: ../../base_code/hello.c
        :diff: ../../base_code/hello_diff.c
```

侧边栏 (Sidebar)

```
.. sidebar:: 这是一个侧边栏

        这是一个侧边栏，可以放入代码，也可以放入图像代码等等，它下面可以是文字，图像,␣
↪代码等等，如本例中下面是一段文字.

冬日，在暖暖的午后，泡上一杯茶，随便拿起一本书，凑到阳光跟前，是何等的惬意与享受……

风虽然不大，但走在路上，鼻子冷的刺骨的疼；而阳光却那么地温热，温热地忍不住想和她亲吻。

我泡上一杯碧螺春，从书架上随便拿起一本书，走向靠窗的位置，凑到阳光面前，任由她吻着我的脸，就像吻着自

也许是身边暖气的缘故，空气的影子，映衬到桌子上、书纸上。影影绰绰如月下之花影，飘飘忽忽如山间之云气，
```

表格

```
.. csv-table:: Frozen Delights!
   :header: "Treat", "Quantity", "Description"
   :widths: 15, 10, 30

   "Albatross", 2.99, "On a stick!"
   "Crunchy Frog", 1.49, "If we took the bones out, it wouldn't be
   crunchy, now would it?"
   "Gannet Ripple", 1.99, "On a stick!"
```

```
=====  =====  =======
A      B      A and B
=====  =====  =======
False  False  False
True   False  False
False  True   False
True   True   True
=====  =====  =======
```

```
.. table:: Grid Table Demo
   :name: table-gridtable

   +------------------------+----------+----------+----------+
   | Header row, column 1   | Header 2 | Header 3 | Header 4 |
   | (header rows optional) |          |          |          |
   +========================+==========+==========+==========+
   | body row 1, column 1   | column 2 | column 3 | column 4 |
   +------------------------+----------+----------+----------+
   | body row 2             | ...      | ...      |          |
   +------------------------+----------+----------+----------+
```

直接标记 (Explicit Markup)

```
.. 这是一个注释, 你只能在源码中看到我, 我不会被渲染出来.
```

注意: .. 与评论块不能有空格

```
..
   这整个缩进块都是
   一个注释.
   你只能在源码中看到我们, 我们不会被渲染出来

   仍是一个评论.
```

```
.. |image_name| image:: picture.jpeg
   :height: 100px
   :width: 200 px
   :scale: 50 %
   :alt: 对于不能显示图片的时候, 显示这些文字
   :align: right
```

数学公式

注意：尽管 .. math:: 与公式之间可以不需要空行，但是仍然非常建议加上。

```
行内公式 :math:`\alpha > \beta` :

Display 公式:

.. math::

    n_{\mathrm{offset}} = \sum_{k=0}^{N-1} s_k n_k

带标签公式:

.. math::
   :label: This is a label

    n_{\mathrm{offset}} = \sum_{k=0}^{N-1} s_k n_k

多行公式:

.. math::

   (a + b)^2 = a^2 + 2ab + b^2

   (a - b)^2 = a^2 - 2ab + b^2

对齐多行公式:

.. math::

   (a + b)^2  &=  (a + b)(a + b) \\
              &=  a^2 + 2ab + b^2
```

提示警告类

注意：尽管 .. something:: 之间不需要空行。

```
.. tip:: This is a tip

.. note:: This is a note

.. hint:: This is a hint

.. danger:: This is a danger

.. error:: This is an error

.. warning:: This is a warning

.. caution:: This is a caution

.. attention:: This is an attention

.. important:: This is an important

.. seealso:: This is seealso
```

超链接

```
This is a paragraph that contains `a link`_.

.. _a link: http://example.com/
```

内部链接

```
.. _figure-datangfurongyuan:

.. figure:: ../_static/figs/mkdocs/insertfigure.png
...
```

图片

Sphinx 将会自动将图像文件拷贝到输出目录中（例如 HTML 格式输出，会拷贝到 _static 目录中。）

- Sphinx 扩展了标准的 docutils 的功能，允许文件扩展名为星号:

```
.. image:: gnu.png
   (options)

.. image:: gnu.*
   (options)
```

引用

引用文档

```
自定义引用文字
:doc:`引用本地的其它 rst 文档,rst 后缀要省略，如 about_us <../about_us>`

使用标题文字
:doc:`../about_us`
```

使用标签引用文档

```
:ref:`about_embedfire <about_embedfire>`

:ref:`about_embedfire`
```

引文

所有的文件可以使用所有的引文。

引文用法是类似的脚注的用法，但带标签不是数字，或以 # 开始。

```
Lorem ipsum [Ref]_ dolor sit amet.
.. [Ref] Book or article reference, URL or whatever.
```

脚注

```
Lorem ipsum [#f1]_ dolor sit amet ... [#f2]_

.. rubric:: Footnotes

.. [#f1] Text of the first footnote.
.. [#f2] Text of the second footnote.
```

下载

```
:download:`引用非 rst 的本地文档 <../requirements.txt>`.
```

html

```
.. raw:: html
<iframe src="//player.bilibili.com/player.html?aid=70961112&cid=122951107&page=1"␣
↪crolling="no" border="0" frameborder="no" framespacing="0" allowfullscreen="true">
↪</iframe>
```

Sphinx 标记结构

toctree

由于 reST 不便于多个文件相互关联或者分割文件到多个输出文件中，Sphinx 通过使用自定义的指令（标识符）来处理构成文档的单个文件的关系，这同样使用与内容表。toctree 指令（标识符）就是核心要素。

推荐阅读：

- Toctree 的 Sphinx 使用手册
- TOC 树

几个重要的附加选项

```
:maxdepth:2             设置最大深度
:numbered:              自动编号
:name:                  名字
:titlesonly:            仅显示在树中的文件的标题，而不是其他的同级别的标题
:glob:                  通配符，这样写文件条目简单写
:reversed:              反向编号
:hidden:                隐藏，如果想要从toctree生成”sitemap”的话，这是非常有用的
```

特殊的文件名

```
genindex 总索引
modindex Python模块索引
search 搜索页
```

案例

- 所有这些文件的内容表被加入，最大的深度为 2，这意味着一个嵌套标题。在这些文件中的 toctree 指令（标识符）也会被考虑到（识别）。
- All about strings <strings>，index.rst 中显示 All about strings，点击时索引到 string.rst（可以换成<https://www,baidu.com>）
- glob 使得 docdir2/* 和 * 的星号能被替换

```
.. toctree::
    :glob:
    :reversed:
    :numbered:
    :maxdepth:2
    :caption: test

    intro
    docdir2/*
    *
    All about strings <strings>
    Go to Baidu <https://www,baidu.com>
```

内联标记 Inline markup

Sphinx 使用经过解释的 text role 将语义标记插入文档。

交叉引用语法

```
:role:`target`

使用下面的可以链接
:role:`title <target>`

下面不会创建链接
:role:`!target`

下面创建的是路径
:role:`~target`
```

交叉引用 objects

python 的交叉引用

```
:py:mod:
:py:func:
:py:data:
:py:const:
:py:class:
:py:meth:
:py:attr:
:py:exc:
:py:obj:
```

替换

```
|release|

|version|

|today|
```

段落级标记

这些指令（标识符）创建简短的段落，可用于内部信息的单位以及普通的文本

```
.. note:: 这是note

.. note::
    这也是 note，推荐这么写。

.. warning:: 这是warning

.. versionadded:: 2.5
        The *spam* parameter.

.. versionchanged:: 2.6
        The reason why you changed the version.

.. deprecated:: 1.0
        说明了什么场合功能不推荐使用。
        Use :func:`spam` instead.

.. seealso:: 这是seealso

.. rubric:: title

该指令（标识符）创建一个段落，标题，不使用节点创建一个表的内容。

.. centered:: LICENSE AGREEMENT

创建一个居中粗体显示的文本行。
```

(下页继续)

(续上页)

```
.. hlist::
   :columns: 3

   * A list of
   * short items
   * that should be
   * displayed
   * horizontally
```

3.2.5 Latex 教程

Latex 的学习并不简单，想要入门的朋友们要有心理准备。你可以看那些半小时入门 Latex 的书，但一定要仔仔细细阅读一本正统的书。他能帮助你解答很多疑惑，让你少走弯路。下面推介一个简单的入门路径。

1. 阅读各类关于 Latex 学习的帖子：初步了解 Latex。
2. 学习使用模板，一般官方正规的模板会有详细的模板使用指南，这就涉及到一些边边角角的知识了。推荐几个模板
    1. 清华和浙大的毕业论文模板，可以在 GitHub 上找到
    2. ACS 期刊的模板
3. 最后必须要回归正统，也就是读一些专业的，系统的书，这里推介一下刘海洋的《Latex 入门》

Installation

安装 texlive

```
sudo apt-get install texlive-latex-base
sudo apt-get install texlive-fonts-recommended
sudo apt-get install texlive-fonts-extra
sudo apt-get install texlive-latex-extra

# 或者简单一点
sudo apt-get install texlive-full

# 中文支持
sudo apt-get install texlive-lang-chinese
sudo apt-get install texlive-xetex  # 编译引擎
```

编译

这里只展示 xetex 编译含有中文字符的文档。

```
xelatex example.tex
bibtex example.aux
xelatex example.tex
xelatex example.tex
```

Tutorial

空行不是必要的，但它可以让长的文档更易读。

简单的例子

```
\documentclass{article}

\begin{document}

\title{AI in Drug}
\author{Zhai Silong}
\date{\today}
\maketitle

\pagenumbering{roman}
\tableofcontents
\newpage
\pagenumbering{arabic}

hello, world
\end{document}
```

文档以及导言

```
\documentclass[a4paper, 12pt]{article}
```

article 文档类型适合较短的文章，比如期刊文章和短篇报告。report（适用于更长的多章节的文档，比如博士生论文）。proc（会议论文集）book 和 beamer。

日期

```
\data{November 2013}
\date{\today}
```

章节与段落

```
\documentclass{article}
\title{Hello World}
\begin{document}
\maketitle

\section{Hello China} China is in East Asia.
\subsection{Hello Beijing} Beijing is the capital of China.
\subsubsection{Hello Dongcheng District}

\paragraph{Tian'anmen Square}is in the center of Beijing
\subparagraph{Chairman Mao} is in the center of Tian'anmen Square
```

(下页继续)

(续上页)

```
\subsection{Hello Guangzhou}
\paragraph{Sun Yat-sen University} is the best university in Guangzhou.
\end{document}
```

对于 report 和 book 类型的文档我们还支持 \chapter{...} 的命令

段落缩进

LaTeX 默认每个章节第一段首行顶格，之后的段落首行缩进。如果想要段落顶格，在要顶格的段落前加 \noindent 命令即可。如果希望全局所有段落都顶格，在文档的某一位置使用 \setlength{\parindent}{0pt} 命令，之后的所有段落都会顶格。

标签

标签只能给章节创建，只能显示且无法链接

```
\section{Methods}

\subsection{Stage 1}
\label{sec1}
The first part of the methods.

\ref{sec1}  % show labelname
\pageref{sec1}  % show page number
```

目录

```
\documentclass{article}
\begin{document}
\tableofcontents  % show words of Contents
\section{Hello China} China is in East Asia.
\subsection{Hello Beijing} Beijing is the capital of China.
\subsubsection{Hello Dongcheng District}
\paragraph{Hello Tian'anmen Square}is in the center of Beijing
\subparagraph{Hello Chairman Mao} is in the center of Tian'anmen Square
\end{document}
```

参考文献

注意：参考文献有两个拓展的宏包，一个是 natbib 一个是 biblatex，他们是不兼容的。

BibTeX 文件类型

文献类型（reference type)

- @article
- @book
- @incollection: 表示一个章节
- @inproceedings: 会议论文

不引包，简单地插入文献列表。

```
\bibliographystyle{plain}
\bibliography{references}
```

参考文献标注

使用 `\cite{citationkey}` 来在你想要引用文献的地方插入一个标注。如果你不希望在正文中插入一个引用标注，但仍想要在文献列表中显示这次引用，使用 `\nocite{citationkey}` 命令。想要在引用中插入页码信息，使用方括号：`\cite[p. 215]{citationkay}` 要引用多个文献，使用逗号分隔：`\cite{citation01,citation02,citation03}`

引用格式

Plain 方括号包裹数字的形式，如 `[1]`。文献列表按照第一作者的字母表顺序排列。每一个作者的名字是全称。Abbrv 与 plain 是相同的，但作者的名字是缩写。Unsrt 与 plain 是相同的，但文献列表的排序按照在文中引用的先后顺序排列。Alpha 与 plain 一样，但引用的标注是作者的名字与年份组合在一起，不是数字，如 [Kop10]。

注释和空格

% 代表评论，之后同一行的字都不会被输出通常来说，LaTeX 忽略空行和其他空白字符，两个反斜杠 (`\\`) 可以被用来换行。如果你想要在你的文档中添加空格，你可以使用 `\vaspace{...}` 的命令。如 `\vspace{12pt}` 会产生一个空格，高度等于 12pt 的文字的高度。

特殊字符

下列字符在 LaTeX 中属于特殊字符，可以用 \ 进行转义：

```
# $ % ^ & _ { } ~ \
```

反斜杠不能通过反斜杠转义（换行），使用 \textbackslash 命令代替。

字体

中文支持

推荐方法：改变布局和文中字

```
% UTF8 可带可不带
\documentclass[UTF8]{ctexart}
```

或者

```
\usepackage[UTF8]{ctex}
```

其他方法：不改变布局，只显示中文文字

```
\documentclass{article}
\usepackage{xeCJK}

\begin{document}
hello,你好
\end{document}
```

字体效果

```
\textit{words in italics}
\textsl{words slanted}
\textsc{words in smallcaps}
\textbf{words in bold}
\texttt{words in teletype}
\textsf{sans serif words}
\textrm{roman words}
\underline{underlined words}
```

彩色字体

```
\usepackage{color}
% green red blue cyan magenta yellow white
{\color{colorname}text}
\colorbox{colorname}{text}
```

字体大小

```
normal size words
{\tiny tiny words}
{\scriptsize scriptsize words}
{\footnotesize footnotesize words}
{\small small words}
{\large large words}
{\Large Large words}
{\LARGE LARGE words}
{\huge huge words}
```

列表与表格

LaTeX 支持两种类型的列表：有序列表（enumerate）和无序列表（itemize）。

```
\begin{enumerate}
\item First thing
\item Second thing
\begin{itemize}
\item A sub-thing
\item Another sub-thing
\end{itemize}
\item Third thing
\end{enumerate}

% 修改头
\begin{itemize}
\item[+] A sub-thing
\item[-] Another sub-thing
\end{itemize}
```

```
\begin{table}
\caption{The Number of Iterations}
\centering
% Each element in the tabular is left aligned
\begin{tabular}{l l}  % fist and second line aligned to left
\hline
iter1 & iter2  \\
\hline
31 & 25  \\
20 & 17  \\
45 & 37  \\
23 & 19  \\
\hline
\end{tabular}
\end{table}
```

表格（tabular）命令用于排版表格。LaTeX 默认表格是没有横向和竖向的分割线的——如果你需要，你得手动设定。LaTeX 会根据内容自动设置表格的宽度。l 表示一个左对齐的列.r 表示一个右对齐的列.c 表示一个向中对齐的列.l 表示一个列的竖线. & 用于分割列.\\ 用于换行.\hline 表示插入一个贯穿所有列的横着的分割线.\cline{1-2} 会在第一列和第二列插入一个横着的分割线

```
\begin{tabular}{|l|l|}
Apples & Green \\
Strawberries & Red \\
Orange & Orange\\
\end{tabular}

\begin{tabular}{rc}
Apples & Green\\
\hline
Strawberries & Red \\
\cline{1-1}
Oranges & Orange \\
\end{tabular}

\begin{tabular}{|r|l|}
\hline
```

(下页继续)

(续上页)

```
8 & here's \\
\cline{2-2}
86 & stuff\\
\hline \hline
2008 & now \\
\hline
\end{tabular}
```

符号和公式

数学公式, 希腊字母，以及特殊符号都需要在数学环境中使用。如果是在同一行，使用 `$ 表达式 $`。如果是另起一行, 使用`\[表达式\]` 或者 `$$ 表达式 $$`。

矩阵

矩阵需要使用 package amsmath。`$$` 之内不能有空行。

```
\documentclass{article}
\usepackage{amsmath}

\begin{document}
$$
\begin{bmatrix}  % bracket []
0 & 0 & 1 \\
1 & 0 & 0 \\
\end{bmatrix}
\begin{pmatrix}  % parentheses ()
1 & 4 & 0 \\
2 & 5 & 8 \\
\end{pmatrix}
\begin{vmatrix}  $ vertical ||
1 & 4 & 0 \\
2 & 5 & 8 \\
\end{vmatrix}
$$
\end{document}
```

公式对齐

```
\begin{align}
53(4+x)+2x &= 212 + 53x + 2x \\
&= 212 + 55x
\end{align}
```

插图

```
\usepackage{graphicx}
\begin{figure}
\includegraphics[scale=.5]{tmp.png}
\caption{fig: tmp}
\end{figure}
```

```
% others
\includegraphics[width=4.00in,height=3.00in]{figure1.eps}
```

Markdown 与 Latex

- 以 Markdown 撰写文稿，以 LaTeX 排版

使用 markdown + pycharm 命令要带 `--shell-escape`

latex 模板推荐

- 优雅：书论文还有笔记
- ElegantPaper
- 清华论文 Latex 模板

Links

- LATEX 教程
- LaTeX 入门

3.3 Basic

3.3.1 Pycharm

PyCharm 是由 JetBrains 公司开发的提供给 Python 专业的开发者的一个集成开发环境，它最大的优点是能够大大提升 Python 开发者的工作效率，为开发者集成了很多用起来非常顺手的功能，包括代码调试、高亮语法、代码跳转、智能提示、自动补全、单元测试、版本控制等等。此外，PyCharm 还提供了对一些高级功能的支持，包括支持基于 Django 框架的 Web 开发、。

配置

字体大小

File->Setting->Editor->Font

Size: 16

Ctrl+ 鼠标滚动调整大小

File->Setting->Editor->General->Mouse-> 第二个复选框打勾

快捷键

```
ctrl+shift+A        # 万能命令行
shift两次           # 查看资源文件

Ctrl + 左击         # 跳转到（变量、方法、类）声明

Ctrl + /            # 行注释或取消注释

Ctrl + Shift + ]/[ # 选定代码块结束、开始
Alt + /             # 自动完成（联想）

Ctrl + D            # 复制选定的区域或行
Ctrl + Y            # 删除选定的行

Tab                 # 缩进
Shift + Tab         # 反缩进

Shift + Enter       # 下一行

Ctrl + Alt + I      # 自动缩进
Ctrl + Alt + L      # 代码美化

Shift + F9          # 调试
Shift + F10         # 运行

Ctrl + F            # 替换
Ctrl + R            # 查找
```

PyCharm 自动添加作者注释

https://blog.csdn.net/lly1122334/article/details/103181579

插件推荐

1. Tabnine

3.3.2 ipython

IPython 是一种基于 Python 的交互式解释器，提供了强大的编辑和交互功能。

IPython 的优点：

- 满足你各种需求的交互式 shell
- 火爆数据科学社区的 Jupyter 内核（供 Jupyter Notebook 使用）
- 对交互式数据可视化和 GUI 工具的完美支持
- 简单易用的高性能并行计算工具

Usage

```
? : 内省的帮助命令
object ?: 对象属性内审
object ??: 对象源码内省
history: 查看历史输入
hist
! shell_command: 执行shell命令
```

Tab 自动补全

魔法命令

```
%: 魔法命令 单行有效
%%: 魔法命令 单元 cell 有效
%run script.py: 执行脚本, 相当于 cell 运行该脚本
%run -d: 交互式执行脚本
%timeit
%%timeit
%pwd
%matplotlib inline: 将图表直接嵌入到 notebook 中, 方便查看
%conda install pkgs: 在 IPython 中安装 python 第三方库
%pylab: 使 numpy 和 matplotlib␣
↪中的科学计算功能生效, 这些功能被称为基于向量和矩阵的高效操作, 交互可视化特性。它能够让我们在控制台
%quickref: 查看 IPython 的特定语法和魔法命令参考
%ls: 显示目录内容
pd.*Da*?
%cd
_: 打印前一个输出的结果
f();: 抑制输出结果
%debug: 报错之后的一个 cell 启动
%pdb: 报错之前需要开启
%pycat: 高亮脚本
%env:  显示环境变量
%load script.py: 载入代码到下一个 cell
%macro taskname n1 n2 n3-n4 ..: 用来定义宏, 并给宏命名, 执行指定的代码行。
%notebook mynotebook.ipynb: 导出当前 notebook 内容到指定 ipynb 文件中。
%pdef: 打印构造信息
%pdoc: 命令用来打印对象的文档字符串。
```

(下页继续)

(续上页)

```
%who: 显示当前变量
%who int: 显示当前 int 类型变量
%whos: 详细显示当前变量
%save sample.py n1-n7: 保存 cell 到 python 文件
%reset -f: 命令用于删除定义的所有变量
%%html: 渲染 HTML
%%javascript: 运行 JavaScript
%%latex: 渲染 LaTeX
%%markdown: 渲染 markdown
%%writefile: 写入 cell 到文件 文件格式可为 txt、py 等
%magic: 获取魔法命令列表
```

links

- 50 个关于 IPython 的使用技巧，get 起来！

3.3.3 jupyter notebook

快捷键

Ctrl+Enter 执行单元格

Shift-Enter 执行单元格进一格

DD 删除

B（Below）键，在单元格下方新建单元格。可以按 ESC 退出编辑

A 在上方插入

M 转化为 Markdown

Y 转化为代码

Tab 代码补全或缩进

Shift-Tab 代码提示

魔法命令

magic 函数主要包含两大类

行魔法（Line magic）前缀为% 单元魔法 (Cell magic) 前缀为%%；

Jupyter

```
%lsmagic # 查看所有魔法
%lsmagic? # 查看魔法的帮助
%matplotlib inline #使用matplotlib画图时，图片嵌入在jupyter␣
↪notebook里面，不以单独窗口显示
%pwd # 和linux一样，查找当前目录
%cd ../
%cp test_peace.py test_load.py
%whos # 查看当前变量,类型，信息
```

(下页继续)

(续上页)

```
%reset # 清除变量
%load test_peace.py # 加载一个文件里面的内容

%timeit %%timeit #为代码执行计时
import numpy as np
%timeit np.sin(24)

%%timeit
x=np.sin(20)
np.cos(-x)
```

%%writefile # 后面紧接着一个 file_name.py, 表示在 jupyter notebook 里面创建一个 py 文件，后面 cell 里面的内容为 py 文件内容

```
%%writefile test_peace.py
import numpy as np
print(np.random.randint(1,5))
```

%run # 后面紧接着一个相对地址的 file_name.py，表示运行一个 py 文件

```
%run test_peace.py
```

Jupyter 修改主题

```
# 安装
pip install --upgrade jupyterthemes
# 加载可用主题列表
jt -l
# selecting a particular theme
jt -t <name of the theme>
# 恢复到最初主题
jt -r
# 其中 -f(字体) -fs(字体大小) -cellw(占屏比或宽度) -ofs(输出段的字号) -T(显示工具栏) -
↪N(显示自己主机名)
jt -t grade3 -f fira -fs 13 -cellw 90% -ofs 11 -dfs 11 -T -N
```

- 如何优雅地使用 Jupyter?
- jupyter notebook 如何打开 md 文件

jupyter lab

```
# 激活已经创建好的虚拟环境
conda activate myenv

# 在虚拟环境中安装 ipykernel
conda install ipykernel
pip install ipywidgets

# 安装 kernel, --name 自定义名称
ipython kernel install --user --name myenv

# 不需要该 kernel 时可以删除
jupyter kernelspec remove myenv
```

Links

- 前置机器学习（二）：30 分钟掌握常用 Jupyter Notebook 用法

3.4 合作

3.5 文档规范

3.6 Python Style

3.6.1 Python Style

Python PEP-8 编码风格指南中文版

来自非英语国家的 Python 程序员们，请使用英文来写注释，除非你 120% 确定你的代码永远不会被不懂你所用语言的人阅读到。

英文写作指南

使用英文写作，参考 Strunk 和 White 的《The Elements of Style》

3.6.2 pprint

Python 标准库模块，全称 pretty printer，可以让各种数据结构更美观地输出

```
from pprint import pprint
pprint('hello world')
```

3.6.3 pydantic

pydantic 库是一种常用的用于数据接口 schema 定义与检查的库。

通过 pydantic 库，我们可以更为规范地定义和使用数据接口，这对于大型项目的开发将会更为友好。

当然，除了 pydantic 库之外，像是 valideer 库、marshmallow 库、trafaret 库以及 cerberus 库等都可以完成相似的功能，但是相较之下，pydantic 库的执行效率会更加优秀一些。

基本用法

```
# 数据通过 BaseModel 类来定义
from pydantic import BaseModel

class Person(BaseModel):
        name: str

# 直接传值
p: Person = Person(name="Tom")
```

(下页继续)

(续上页)

```
print(p.json())  # {"name": "Tom"}

# 通过字典传入
p = {"name": "Tom"}
p: Person = Person(**p)
print(p.json())  # {"name": "Tom"}

# 通过其他的实例化对象传入
p2: Person = Person.copy(p)
print(p2.json())  # {"name": "Tom"}
```

特色：

- 传入错误值：报错
- 传入未定义值：忽略

pydantic 基本数据类型

```
from pydantic import BaseModel
from typing import Dict, List, Sequence, Set, Tuple

class Demo(BaseModel):
    a: int # 整型
    b: float # 浮点型
    c: str # 字符串
    d: bool # 布尔型
    e: List[int] # 整型列表
    f: Dict[str, int] # 字典型，key 为 str，value 为 int
    g: Set[int] # 集合
    h: Tuple[str, int] # 元组
```

高级数据结构考察

```
from enum import Enum

class Gender(str, Enum):
        man: str = "man"
        women: str = "women"
```

See also

- https://blog.csdn.net/codename_cys/article/details/107675748

3.6.4 Flake8

Flake8 是由 Python 官方发布的一款辅助检测 Python 代码是否规范的工具，相对于目前热度比较高的 Pylint 来说，Flake8 检查规则灵活，支持集成额外插件，扩展性强。

Flake8 是对下面三个工具的封装：

1. PyFlakes：静态检查 Python 代码逻辑错误的工具。
2. Pep8：静态检查 PEP8 编码风格的工具。
3. NedBatchelder’s McCabe script：静态分析 Python 代码复杂度的工具。

安装

```
pip install flake8
```

3.6.5 Pylint

使用过，变态到发紫；不知道谁那么无聊，将规则定的那么死，我们 pythoner 能快乐吗？乃们不见 rubyer，Matz 倡导的是什么？Happy Coding 有木有？所以用过就仍了，因为我不需要这么变态的搞，无爱

CHAPTER 4

Shell

4.1 tmux

4.1.1 Common Commands

```
sudo apt-get install tmux  # Setup
tmux new -s <session-name>
tmux detach # Ctrl+b d
tmux ls #
tmux attach -t 0 # attach to session 0
tmux attach -t <session-name> # attach to session by name
tmux switch -t 0
tmux switch -t <session-name>
```

切换绘话：推荐使用 `Ctrl+b s` 然后上下键选择

4.1.2 Tmux Windows

```
Ctrl+b %  # 左右分

Ctrl+b "  # 上下分

Ctrl+b x # 关闭当前窗格

Ctrl+b Ctrl+<arrow key>  # 按箭头方向调整窗格大小。

Ctrl+b q  # 显示窗格编号。

ctrl+b ,  # rename the window

ctrl+b $   # rename the session
```

(下页继续)

(续上页)

```
Ctrl+b ;  # 光标切换到上一个窗格。

ctrl+b [  # 然后鼠标就可以移动了

ctrl+b :new -t new_session_name  # new a session and move into
```

4.2 Git

远程仓库和本地仓库名字的区别:

- origin 远程库名
- main 本地分支名
- origin/main 远程库中的远程分支

4.2.1 Git

Setup

```
sudo apt install git
```

配置

每个仓库的 Git 配置文件都放在当前目录下的 `.git/config` 文件中。

- `/etc/gitconfig` 文件: 系统中对所有用户都普遍适用的配置。若使用 `git config` 时用 `--system` 选项，读写的就是这个文件。
- `~/.gitconfig` 文件: 用户目录下的配置文件只适用于该用户。若使用 `git config` 时用 `--global` 选项，读写的就是这个文件。
- 当前文件夹`/.git/config` 文件: 这里的配置仅仅针对当前项目有效。使用 `git config`

```
# 查看当前配置信息
git config --list
git config --global --list

# 修改用户信息
git config --global user.email "zhaisilong@outlook.com"
git config --global user.name "zhaisilong"

# 添加 GPG 密钥 要先上传公钥
#E GPG 用于签名验证 注释
git config --global user.signingkey D482B92E13D3FCBA
# 自动验证签名 自动加上 -S
git config --global commit.gpgsign true

# 文本编辑器
git config --global core.editor emacs
# 差异分析工具
```

(下页继续)

(续上页)

```
git config --global merge.tool vimdiff
# 开启界面颜色
git config --global color.ui true
```

配置自动换基

自动换基，保持开发线的整洁。

```
git config --global pull.rebase true
git config --global rebase.autoStash true
```

配置 ssh key

配置文件的编辑

```
vim ~/.ssh/config
# github
Host github.com
HostName github.com
PreferredAuthentications publickey
IdentityFile ~/.ssh/git.rsa

# gitee
Host gitee.com
HostiName gitee.com
PreferredAuthentications publickey
IdentityFile ~/.ssh/git.rsa
```

常用命令

```
# 初始化仓库
git init

# 增加文件
git add [file1] [file2]
# 增加所有文件
git add -A

# 提交到本地仓库 并附加说明
# `-m`后面输入的是本次提交的说明
git commit -m "wrote a readme file"
# 签名验证，需要配置 gpg
git commit -S -m "..."

# -M, --move --force:
git branch -M main  #重命名分支名
# 列出所有分支
git branch -a
# 查看远程库的信息
git remote -v
```

(下页继续)

(续上页)

```
# 添加远程库
##␣
↪可以从这个仓库克隆出新的仓库，也可以把一个已有的本地仓库与之关联，然后，把本地仓库的内容推送到GitHu
git remote add {{remote_name}} {{remote_url}}
git remote add origin https://github.com/xxzhai123/PbootCMS.git
git remote remove {{remote_name}}
git remote rename {{old_name}} {{new_name}}

# 将本地库推到远程
## 当只有一个分支名时，简写成git push
git push {{remote_name}} {{local_branch}}
# 指定远程分支名将本地库上传
git push {{remote_name}} -u {{remote_branch}}
# 删除远程库中的一个分支
git push {{remote_name}} --delete {{remote_branch}}

# 拉取远程仓库
git fetch # 拉取
git merge # 合并
git pull # 拉取并合并
git pull origin master # 指定远程库的分支拉取

# 查看状态
git status
git diff # 命令查看文件是如何被修改
git log # 查看合并日志，以及 Head 信息
git log --oneline --decorate --graph
```

删除远程仓库的文件

```
git rm -r -n --cached data/
git rm -n --cached data/
git commit -m "delete"
git push
```

大容量

```
git config --global http.postBuffer 524288000
```

分支的创建与合并

注意 iss53 从未出现在远端

```
git checkout -b iss53 # 在当前分支，创建新分支
# 等价于下面两条
git branch iss53
git checkout iss53
# 然后修复 bug

# 修复好回到 main
```

(下页继续)

(续上页)

```
git checkout main
git merge iss53
# 最后删除 iss53
git branch -d iss53
```

如果不允许不想干分支合并请用

```
git merge iss53 --allow-unrelated-histories
```

如何更换 git 的远程库

ssh 地址：`git@github.com:zhaisilong/software.git` https 地址：`https://github.com/zhaisilong/software`

```
git remote set-url origin <ssh地址> | <https地址>
```

配置命令别名

```
# 告诉Git，以后st就表示status：
$ git config --global alias.st status
```

4.2.2 忽略特殊文件

不需要从头写`.gitignore`文件，GitHub 已经为我们准备了各种配置文件，只需要组合一下就可以使用了。所有配置文件可以直接在线浏览：https://github.com/github/gitignore

`.gitignore`文件的格式规范如下：

1. #注释
   - 不允许行内注释
2. 可以使用标准的 glob 模式匹配
3. ！开头表示不忽略
4. / 开头表示根目录,/结尾表示目录 `/*.js` 可以匹配 `app.js`，但无法匹配 `js/app.js`。
5. ** 表示匹配零到多级目录

4.2.3 子模块

子模块允许你将一个 Git 仓库作为另一个 Git 仓库的子目录。默认情况下，子模块会将子项目放到一个与仓库同名的目录中。

创建名为 `.gitmodules` 的文件。该配置文件保存了项目 URL 与已经拉取的本地目录之间的映射

```
git submodule add
```

克隆带有子模块的仓库

正常 clone 包含子模块的函数之后，由于.submodule 文件的存在 someSubmodule 已经自动生成。但是里面是空的。还需要执行 2 个命令。

```
# clone 父仓库的时候加上 --recursive，会自动初始化并更新仓库中的每一个子模块
git clone --recursive https://github.com/chaconinc/MainProject


# 或者分步拉取
## 用来初始化本地配置文件
git submodule init
## 从该项目中抓取所有数据并检出父项目中列出的合适的提交(指定的提交)。
git submodule update

# 更新子模块
## Git 将会进入子模块然后抓取并更新，默认更新 master 分支
git submodule update --remote
```

4.2.4 标签管理

如果你达到一个重要的阶段，并希望永远记住那个特别的提交快照，你可以使用 git tag 给它打上标签。

```
git tag # 查看标签信息
git tag -a v1.0
git tag -a v0.9 85fc7e7 # 追加标签，忘记了的话
git log --oneline --decorate --graph
# 指定标签信息
git tag -a v1.0 -m "runoob.com标签"
# 用 gpg 指定标签信息
```

4.2.5 Gitee

使用 GitHub 时，国内的用户经常遇到的问题是访问速度太慢，有时候还会出现无法连接的情况（原因你懂的）。

如果我们希望体验 Git 飞一般的速度，可以使用国内的 Git 托管服务——Gitee

4.2.6 git-lfs

```
# 安装
sudo apt-get install git-lfs
# 关联 git，这一步的目的是改变 git 的全局配置，查看 .gitconfig 文件
git lfs install
```

用 LFS 追踪的文件，这一步的目的是下面的文件变成 LFS 文件，支持通配符。操作完成后会生成或更新 *.gitattributes* 配置文件

```
git lfs track "*.sql"
git lfs untrack "文档.doc"
git lfs ls-files
git add .gitattributes
git commit -m "add .gitattributes"
```

4.2.7 Links

- Git 各指令的本质，绝对通俗易懂！| 强烈推介

4.3 Aria2

4.3.1 Setup

```
conda install aria2 -c biconda # Recommended
sudo apt-get install aria2 # Debian
yay -S aria2 # Arch
```

WebUI can be found with google chrome plugins, search by *aria2 for chrome*.

Most useful commands

```
# --continue, --split, max, j 同时下载数, --out
aria2c {{URL}} -c -s2 -x8 -j 10  -o out.zip
# download to pwd
aria2c <URL>
#
aria2c -s2 -o out.type <URL>
#
aria2c -c <URL>
# use proxy
aria2c all-proxy=127.0.0.1:10809 <URL>

# 种子下载
aria2c xxx.torrnet
aria2c -S target.torrent # 列出种子内容
aria2c --select-file=1,4-7 target.torrent # 选择下载种子
```

4.3.2 Configuration

这里需要注意，`aria2.session` 与 `aria2.conf` 一定要在 `etc` 中，否则无法以 daemon 形式启动

```
sudo mkdir -p /etc/aria2 && cd /etc/aria2 #新建文件夹
sudo touch aria2.session #新建 session 文件，用来缓存下载，可断点续传
sudo chmod 777 aria2.session #设置aria2.session可写
sudo vim aria2.conf #创建配置文件
```

```
# aria2.conf
# 注意，配置不支持行内注释
dir=/home/seeyou/下载
disable-ipv6=true

#打开 rpc 的目的是为了给 web 管理端用
enable-rpc=true
rpc-allow-origin-all=true
rpc-listen-all=true

#rpc-listen-port=6800
#断点续传
```

(下页继续)

(续上页)

```
continue=true
input-file=/etc/aria2/aria2.session
save-session=/etc/aria2/aria2.session

#最大同时下载任务数
max-concurrent-downloads=20
save-session-interval=120

# Http/FTP 相关
connect-timeout=120
#lowest-speed-limit=10K
#同服务器连接数
max-connection-per-server=10
#max-file-not-found=2
#最小文件分片大小，下载线程数上限取决于能分出多少片，对于小文件重要
min-split-size=10M

#单文件最大线程数，路由建议值：5
split=10
check-certificate=false
#http-no-cache=true

# 使用代理
all-proxy=127.0.0.1:10809
```

4.3.3 启动

```
sudo aria2c --conf-path=/etc/aria2/aria2.conf -D
```

开机启动，可以以脚本的形式加入 KDE 桌面，tips：`alt` + `space` -> `autostart`。

在 Debian 系电脑自己编写开机启动脚本

```
sudo vim /etc/init.d/aria2c # 增加开机启动脚本
sudo chmod 755 /etc/init.d/aria2c # 修改文件权限为755 (a+x)
sudo update-rc.d aria2c defaults # 添加 aria2c 服务到开机启动
sudo service aria2c start # 启动服务
sudo systemctl status aria2c # 查看服务状态
```

```
#!/bin/sh
### BEGIN INIT INFO
# Provides: aria2
# Required-Start: $remote_fs $network
# Required-Stop: $remote_fs $network
# Default-Start: 2 3 4 5
# Default-Stop: 0 1 6
# Short-Description: Aria2 Downloader
### END INIT INFO

case "$1" in
start)

 echo -n "已开启Aria2c"
 aria2c --conf-path=/etc/aria2/aria2.conf -D
```

(下页继续)

(续上页)

```
;;
stop)

 echo -n "已关闭Aria2c"
 killall aria2c
;;
restart)

 killall aria2c
 aria2c --conf-path=/etc/aria2/aria2.conf -D
;;
esac
exit
```

4.4 SSH

SSH-Key 与一般的 gpg 密钥有所区别

1. 私钥只有命名为 id_rsa 才有效否则需要指定。
2. 公钥必须写入 `~/.ssh/authorized_keys` 才有效。
3. 一个密钥对可重复使用。
4. 密钥的权限不能太高，否则会报错。`700`

ssh 提供两种级别的安全认证：

1. 基于口令的安全认证
2. 基于密钥的安全认证

4.4.1 RSA 密钥

passphrase 是证书口令，以加强安全性，避免证书被恶意复制。

会在 `~/.ssh` 下生成 `id_rsa`, `id_rsa.pub` 两个文件，分别是私钥/公钥。

```
# 生成 ssh-key，选加密算法（rsa、dsa），给秘钥命名（可选）：
ssh-keygen -t rsa -b 2048 -C "957574373@qq.com" -f ./zhaisilong_rsa

# 将公钥传输到远程服务器
ssh-copy-id -i ~/.ssh/id_rsa.pub root@172.xx.yy.zzz
```

判断公钥与私钥是否配对

```
ssh-keygen -y -e -f <private key>
```

下面手动演示具体流程

```
scp ~/.ssh/id_rsa.pub root@47.111.225.3:/root/.ssh/
# 要保证 .ssh 和 authorized_keys 都只有用户自己有写权限。否则验证无效
chmod -R 700 ~/.ssh/
chmod 600 ~/.ssh/authorized_keys
```

(下页继续)

(续上页)

```
# 使用 ssh-copy-id 则不需要此步
cat id_rsa.pub >> authorized_keys
```

登录远程服务器

简单情况下，通过手动指定私钥文件登录

```
ssh -p 22 username@hostname -i "~/.ssh/my_id_rsa"
```

配置客户端，可自动指定私钥文件

```
vim ~/.ssh/config

Host alias
    HostName hostname
    User root
    Port 22
    IdentityFile ~/.ssh/id_rsa
```

4.4.2 配置远程主机 sshd 的配置

一般不需要配置。下面以 Ubuntu 为例

```
sudo vim /etc/ssh/sshd_config

PasswordAuthentication yes
PubkeyAuthentication yes
# 有了证书登录了，可以禁用密码登录
PasswordAuthentication no
# 禁用root账户登录，非必要，但为了安全性，请配置
PermitRootLogin no

# 重启 sshd 使改动生效
systemctl reload sshd
```

4.4.3 See also

- Linux 下使用 SSH 远程执行命令方法收集

4.5 System Control

4.5.1 Proc

```
/proc/cpuinfo cpu的信息
/proc/mounts 系统中使用的所有挂载
/proc/uptime 系统已经运行了多久
/proc/version Linux内核版本和gcc版本
/proc/swaps 交换空间的使用情况
```

(下页继续)

(续上页)

```
/proc/stat 所有的CPU活动信息
/proc/net 网卡设备信息
/proc/tty tty设备信息
/proc/vmstat 虚拟内存统计信息
/proc/diskstats 取得磁盘信息
/proc/zoneinfo 显示内存空间的统计信息，对分析虚拟内存行为很有用
```

以下是 /proc 目录中进程 N 的信息

```
/proc/N pid为N的进程信息
/proc/N/cmdline 进程启动命令
/proc/N/cwd 链接到进程当前工作目录
/proc/N/environ 进程环境变量列表
/proc/N/exe 链接到进程的执行命令文件
/proc/N/fd 包含进程相关的所有的文件描述符
/proc/N/maps 与进程相关的内存映射信息
/proc/N/mem 指代进程持有的内存，不可读
/proc/N/root 链接到进程的根目录
/proc/N/stat 进程的状态
/proc/N/statm 进程使用的内存的状态
/proc/N/status 进程状态信息，比stat/statm更具可读性
/proc/self 链接到当前正在运行的进程
```

4.6 终端

4.7 综合实战

CHAPTER 5

Python

5.1 Python

5.1.1 Python 安装

安装 Python

- 如何在 Ubuntu 20.04 上安装 Python 3.9

安装 pip

```
# Ubuntu 自带安装 3.8
sudo apt install python3
sudo apt install python3-pip
```

5.1.2 Python OS

```
import os
os.chdir(path)
os.getcwd()
os.listdir(path)
os.makedirs('/home/seeyou/a/b/c', exist_ok=True)
os.mkdir('./book', exist_ok=True)
os.remove(path)
os.removedirs(path)
os.rename(src, dst)
os.chmod(path, mode)
os.chown(path, uid, gid)
os.close(fd)
```

(下页继续)

(续上页)

```
os.getppid() # 父进程的 id
os.getpid() # 当前进程的 id
import os.path
# 返回绝对路径
os.path.abspath(path)
# 返回文件名
os.path.basename(path)
# 返回文件路径
os.path.dirname(path)
# 是否存在
os.path.exists(path)
# 文件大小
os.path.getsize(path)
# 判断
os.path.isabs(path)
os.path.isfile(path)
os.path.isdir(path)
os.path.islink(path)
os.path.ismount(path)
# 目录和文件合成
os.path.join(path1[, path2[, ...]])
# 将目录和文件名分割，返回一个元组
os.path.split(path)
```

实战设置临时环境变量

```
os.environ['WORKON_HOME'] = "value"

# 获取环境变量，推荐第二种
os.environ.get('WORKON_HOME')
os.getenv('path')
# 删除变量
del os.environ['WORKON_HOME']
```

比如 hugging face 的数据获取不了，可以如此

```
import os
proxy='http://127.0.0.1:10809'
os.environ['https_proxy'] = proxy
os.environ['http_proxy'] = proxy
from datasets import Dataset
import datasets
train_test_ds: Dataset = datasets.load_dataset('bookcorpus', split='train+test')
```

5.1.3 Python Debug

Python 的调试方法有三种，一种是执行时调试，一种是交互调试，一种是程序里埋点调试。

其中，最常用的是执行时调试，也就是 pdb 调试。

pdb 有 2 种用法：

```
# 1. 非侵入式方法（不用额外修改源代码，在命令行下直接运行就能调试）
python3 -m pdb filename.py

# 2. 侵入式方法（需要在被调试的代码中添加一行代码然后再正常运行代码）
import pdb;pdb.set_trace()
```

pdb 常用命令

```
l 查看当前位置前后11行源代码（多次会翻页）
ll 查看当前函数或框架的所有源代码

添加断点
b
b lineno
b filename:lineno
b functionname

添加临时断点
tbreak
tbreak lineno
tbreak filename:lineno
tbreak functionname

清除断点
cl
cl filename:lineno
cl bpnumber [bpnumber ...]

打印变量值
p expression, expression 为 Python 表达式

逐行调试命令
s 执行下一行（能够进入函数体）
n 执行下一行（不会进入函数体）
r 执行下一行（在函数中时会直接执行到函数返回处）

非逐行调试命令
c 持续执行下去，直到遇到一个断点
unt lineno 持续执行直到运行到指定行（或遇到断点）
j lineno 直接跳转到指定行（注意，被跳过的代码不执行）

查看函数参数
a 在函数中时打印函数的参数和参数的值

打印变量类型
whatis expression

启动交互式解释器
```

(下页继续)

(续上页)

```
interact
启动一个 python 的交互式解释器，使用当前代码的全局命名空间（使用ctrl+d返回pdb）

打印堆栈信息
w

退出 pdb
q
```

See also

- 10 分钟教程掌握 Python 调试器 pdb

5.1.4 Python 环境

关于 Python 虚拟环境管理，很多人一开始不以为意，把编程时所有依赖的库全安装在一起，要用的时候直接导入，看似非常方便，但是会造成很多隐患：

- 当电脑/服务器里面的项目越来越大，更新迭代次数多了，会造成很多以前用到现在不需要用到的库，把这些库都写在 requirements 里面会造成冗余，而且 docker 化的时候安装 requirements 特别慢；
- 当电脑/服务器里面的项目越来越多，会造成很多项目的依赖库的版本相互冲突，而且也不敢随便删掉/更新某些库，因为有可能会造成某个项目的依赖无法找到或者版本不兼容的故障；所以，Python 在实际项目应用当中，虚拟环境管理是非常重要的

Python 虚拟环境管理工具的发展

最常用的虚拟环境管理工具是 virtualenv，virtualenvwrapper，pipenv，事实上，这几种虚拟环境的管理都是基于 virtualenv，只是做了不同的封装，达到了更好的效果。

virtualenv -> virtualenvwrapper -> pipenv 推介

pipenv 的优势

1. 跟踪包的关系变化，pipenv 会在项目目录下创建 Pipfile 和 Pipfile.lock 文件
2. 安装卸载包无需激活虚拟环境，直接在项目文件夹下即可操作
3. 卸载的时候，可以自动检查依赖库是否被其他包依赖，来选择是否彻底删除。也可以通过 pipenv graph 来查看各个包的依赖关系图；
4. 当代码需要在虚拟环境执行时，通过 pipenv run python xx.py，即可在虚拟环境下执行 python 文件。
5. 便于 docker 容器化管理，Pipfile 文件支持生成 requirements 文件，便于项目代码 docker 化管理，另外，
6. pipfile 还支持–dev 环境，可以在调试阶段安装许多调试工具等，而不影响生产环境的环境。

使用

```
pip install pipenv  # 安装 pipenv
pip uninstall pipenv  # 卸载 pipenv
pipenv --python 3.6  # 初始化特定版本的环境

# 第一次的话 初始化虚拟环境
# 根据 Pipfile 的记录，安装所有依赖包
pipenv install


pipenv --rm  # 删除当前虚拟环境，注意 Pipfile 不会被删掉

# 进入虚拟环境
cd path/to/env
pipenv shell


exit  # 退出虚拟环境
pipenv install xxx  # 安装相关依赖包
pipenv install --dev xxx  # 安装包到开发组
pipenv uninstall xx:  # 卸载包
pipenv graph  # 查看目前按照的依赖包
pipenv --venv  # 显示虚拟环境安装路径

pipenv install requests==2.13.0
pipenv install "requests~=2.2"  # 锁定包的主版本(这相当于使用==2.*)
pipenv update --outdated  # 检查更新
pipenv update  # 更新
pipenv update <包名>  # 指定更新

pipenv lock -r > requirements.txt  # 将 Pipfile 里的全部包导出 requirements.txt
pipenv lock -r --dev-only > requirements.txt  # 只导出开发环境的包
pipenv lock -r --dev > requirements.txt  # 导出开发环境 + 默认的环境的包
pipenv install -r path/to/requirements.txt  # 安装 requirements.txt
pipenv run python --version  # 如果想进入 shell 运行 python

# 从 setup.py 安装依赖包
pipenv install -e .
```

```
pipenv --python 3.7 创建3.7版本Python环境
```

Pipfile

第一次在项目中运行 pipenv 命令的话，会在项目中创建一个名为 Pipfile 的文件

```
[[source]]
url = "https://pypi.org/simple"
verify_ssl = true
name = "pypi"

[packages]
requests-html = "*"

[dev-packages]
```

(下页继续)

(续上页)

```
[requires]
python_version = "3.7"
```

Pipfile.lock 文件

Pipfile.lock 利用了 pip 中一些新的安全改进。默认情况下，Pipfile.lock 包含每个下载的包的 sha256 哈希值。这可以使 pip 能够保证从不信任的 PyPI 源安装包时或者在不稳定的网络环境下安装包时都能保证包的正确性与完整性。

Pipfile 和 Pipfile.lock 不一致原因：当 pipenv install 安装一些比较大的包，比如 TensorFlow 等科学包，在安装完包，正在解析包依赖的时候，可能会花费很长时间，可能就会 ctrl+C 停止掉了，此时包已经安装完成，Pipfile 已经写入完成，但是没有解析完依赖，所以 Pipfile.lock 没有写入完成。方法：删除 Pipfile.lock；使用 `pipenv lock` 重新生成

利器之环境变量：（.env）

`.env` 文件可以设置一些环境变量，在程序开发的时候模拟环境变量。

```
echo -e 'HELLO==WORLD \n CONFIG_PATH=${HOME}/.config/foo' >> .env
```

如果你的.env 文件位于不同的路径下或者有其他名称，你可以设置一个 `PIPENV_DOTENV_LOCATION` 环境变量：

```
PIPENV_DOTENV_LOCATION=/path/to/.env pipenv shell
```

要禁止 pipenv 加载.env 文件，可以设置 PIPENV_DONT_LOAD_ENV 环境变量：

```
PIPENV_DONT_LOAD_ENV=1 pipenv shell
```

其他

```
alias prp="pipenv run python"  # 少打几个字
```

修改镜像源

```
[[source]]
url = "https://mirrors.aliyun.com/pypi/simple"
verify_ssl = true
name = "pypi"
```

环境的迁移

复制 Pipfile 和 Pipfile.lock 文件到新电脑 `pipenv install`

修改虚拟环境目录位置

有三种方法：

```
# 设置这个环境变量，pipenv 会在当前目录下创建 .venv 的目录，以后都会把模块装到这个 .
↪venv 下。
export PIPENV_VENV_IN_PROJECT=1

# 自己在项目目录下手动创建 .venv 的目录，然后运行 pipenv run 或者 pipenv shell pipenv␣
↪都会在 .venv 下创建虚拟环境。
mkdir .venv
pipenv shell


# 设置 WORKON_HOME 到其他的地方 （如果当前目录下已经有.venv,此项设置失效）。
export WORKON_HOME=$HOME/.virtualenvs
```

See also

- Python—pipenv 精心整理教程
- Python 多环境管理神器（pipenv）

5.1.5 Poetry

poetry 是一个 Python 虚拟环境和依赖管理的工具。poetry 和 pipenv 类似，另外还提供了打包和发布的功能。

如果您不熟悉 pipenv 请阅读

“应该有一种明显的方法，最好只有一种。”

请阅读 Poetry 的历史

相比 Pipenv，Poetry 是一个更好的选择

- Github
- 官网
- 官方文档

poetry 安装

方式一：（推荐）

```
curl -sSL https://install.python-poetry.org | python3 -
```

方式二：（pip）为了防止依赖冲突不推荐使用 pip 的方式直接安装

```
pip install --user poetry
```

使用

工程初始化

```
poetry new poetry-demo
```

创建结构如下

```
poetry-demo
├── pyproject.toml
├── README.rst
├── poetry_demo
│   └── __init__.py
└── tests
    ├── __init__.py
    └── test_poetry_demo.py
```

除了新建工程，还可以在已有工程的基础上进行创建，

```
poetry init  # 生成 pyproject.toml
```

常用命令

```
poetry install  # 解析并安装 pyproject.toml 的依赖包
poetry add numpy requests  # 安装包
poetry add pytest --dev  # 指定为开发依赖
poetry add flask=2.22.0  # 指定具体的版本
poetry install --no-dev  # 一般部署时使用
poetry update  # 更新所有锁定版本的依赖包
poetry self update
poetry update numpy  # 更新指定依赖包
poetry remove numpy  #
poetry show --outdated  # 查看可以更新的依赖
poetry show  # 查看项目安装的依赖
poetry show  numpy # 查看项目安装的依赖
poetry show -t  # 树形结构查看项目安装的依赖
```

add

```
poetry add pendulum@^2.0.5
poetry add "pendulum>=2.0.5"

poetry add pendulum@latest

poetry add git+https://github.com/sdispater/pendulum.git
poetry add git+ssh://git@github.com/sdispater/pendulum.git
poetry add git+https://github.com/sdispater/pendulum.git#develop
poetry add git+https://github.com/sdispater/pendulum.git#2.0.5

poetry add ./my-package/
poetry add ../my-package/dist/my-package-0.1.0.tar.gz
poetry add ../my-package/dist/my_package-0.1.0.whl
```

cache

```
poetry cache list

poetry cache clear --all
poetry cache clear <cache>
```

虚拟环境管理

```
poetry lock  # 锁定 （不安装） 中指定的依赖项

# 指定解释器
poetry env use python3.7
poetry env use /full/path/to/python
poetry env use system

poetry shell
poetry run python -V
poetry env info
poetry env list
poetry env list --full-path
poetry env remove python3.7  # 删除现有的虚拟环境
poetry run python -V

# 导出
poetry export -f requirements.txt --output requirements.txt
# --dev
# --extras (-E): 额外的依赖
# --without-hashes: 忽略哈希
```

包的分发

```
poetry build

# pypi
poetry config http-basic.pypi username password
poetry publish

# 公司有自己的私有仓库
poetry config repositories.foo https://foo.bar/simple/
poetry config http-basic.foo username password
poetry publish -r my-repository
```

自定义镜像源

```
[[tool.poetry.source]]
name = "aliyun"
url = "https://mirrors.aliyun.com/pypi/simple/"
secondary = true

[[tool.poetry.source]]
name = "tsinghua"
url = "https://pypi.tuna.tsinghua.edu.cn/simple/"
default = true
```

也可以用命令行

```
poetry config repositories.tuna https://pypi.tuna.tsinghua.edu.cn/simple
poetry config repositories.tuna --unset
```

进阶

构建软件包并且指定可执行的脚本

指定可执行程序的函数入口：

```
[tool.poetry.scripts]
abc = "bca.abc:main"
```

依赖配置

版本限制：

- 尖括号：^1.2 代表 >=1.2.0 <2.0.0
- 波浪号：~1.2.3 代表 >=1.2.3 <1.3.0
- 星号：1.* 代表 >=1.0.0 <2.0.0

poetry 的全局配置

Tips: poetry config 之后接 `--unset`、`--local`

- unset : 重置某项的配置
- local：配置只对本地的项目生效，不影响全局配置

```
poetry config --list

# 可以设置虚拟环境默认安装到项目的 .venv 目录里
poetry config virtualenvs.in-project true

# 在部署时先使用这个命令可以使所有的包安装到系统中，而不是虚拟环境里
poetry config virtualenvs.create false --local

poetry config virtualenvs.path .virtualenvs  --local
```

注意: python3.8(包括) 以下，不能 poetry config virtualenvs.in-project true

Python Poetry 管理包安装速度慢的解决办法

因为 Poetry 是依靠 pip 来进行安装的，所以我们可以通过更改 pip 镜像源来加快速度。

```
mkdir -p ~/.pip && vim ~/.pip/pip.conf

[global]
index-url = http://mirrors.aliyun.com/pypi/simple/
[install]
trusted-host = mirrors.aliyun.com
```

See also

- Python 包管理之 poetry 的使用
- Poetry 科学管理 Python 虚拟环境

5.1.6 Pyenv

```
curl https://pyenv.run | bash

export PATH="$HOME/.pyenv/bin:$PATH"
eval "$(pyenv init -)"
eval "$(pyenv virtualenv-init -)"


pyenv install --list | grep ' 3.9'

pyenv install 3.9.4

pyenv versions

pyenv global 3.8.0

pyenv local 2.7.17
```

5.2 Python DataFrame

5.2.1 Python Apache Arrow

Python 官方文档

Arrow manages data in arrays (pyarrow.Array), which can be grouped in tables (pyarrow.Table) to represent columns of data in tabular data.

Arrow also provides support for various formats to get those tabular data in and out of disk and networks. Most commonly used formats are Parquet (Reading and Writing the Apache Parquet Format) and the IPC format (Streaming, Serialization, and IPC).

安装

```
conda install -c conda-forge pyarrow
pip install pyarrow
```

简单使用

创建数据表

```
import pyarrow as pa
from pyarrow.lib import Table, Int8Array, Int16Array
days: Int8Array = pa.array([1, 12, 17, 23, 28], type=pa.int8())
months: Int8Array = pa.array([1, 3, 5, 7, 1], type=pa.int8())
years: Int16Array = pa.array([1990, 2000, 1995, 2000, 1995], type=pa.int16())
birthdays_table: Table = pa.table([days, months, years],
                            names=["days", "months", "years"])
```

保存和加载数据表

```
import pyarrow.parquet as pq
pq.write_table(birthdays_table, 'birthdays.parquet')
reloaded_birthdays = pq.read_table('birthdays.parquet')
```

计算

```
import pyarrow.compute as pc
pc.value_counts(birthdays_table["years"])
```

处理大数据

```
import pyarrow.dataset as ds
ds.write_dataset(birthdays_table, "savedir", format="parquet",
                 partitioning=ds.partitioning(
                    pa.schema([birthdays_table.schema.field("years")])
                ))
birthdays_dataset = ds.dataset("savedir", format="parquet", partitioning=["years"])
```

5.2.2 Python CSV

```
import csv
with open('eggs.csv', newline='') as csvfile:
    spamreader = csv.reader(csvfile, delimiter=' ', quotechar='|')
    for row in spamreader:
        print(', '.join(row))
# Spam, Spam, Spam, Spam, Spam, Baked Beans
# Spam, Lovely Spam, Wonderful Spam
```

待续…

5.2.3 Jsonlines

Jsonlines 手册

```
pip install jsonlines
```

```
import jsonlines

with jsonlines.open('input.jsonl') as reader:
    for obj in reader:
```

(下页继续)

(续上页)

```
with jsonlines.open('output.jsonl', mode='w') as writer:
    writer.write(...)
```

5.2.4 Pandas

创建数据对象

```
# 标准导入
import pandas as pd

pd.Series([1, 3, 5, np.nan, 8])
# 指定 series 的 index, index 可重复
pd.Series([1, 3, 5, np.nan, 6, 8], index=['c', 'a', 'i', 'yong', 'j', 'i'])
# 创建 DataFrame
pd.DataFrame(np.array([
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]), index=['i', 'ii', 'iii'], columns=['A', 'B', 'C'])
```

访问

```
df.head(2)
df.tail(3)
df.describe()  # 描述统计信息
```

索引

```
df['A']  # 按列名取
df[0:3]  # 按行数取
df.loc['2021-01-01':'2021-01-02', ['A', 'B']]  # 指定具体的标签
df.loc[['2021-01-01', '2021-01-02'], ['A', 'B']]  # 指定具体的标签
df.loc[['std'], ...]
df.at[int(index), 'column_nam']  # 取点
df.iloc[3:5, 0: 3]  # 指定标签的索引位置
```

函数

```
# 对列排序
df.sort_values(by='C')
# 选择某列最大的 n 行数据
df.nlargest(2, 'A')
# 采样
df.sample(5)
df.sample(frac=0.01)  # 采样 1%
```

赋值

```
# 坐标赋值
df.iloc[2, 2] = 1111
# 轴名赋值
df.loc['20130101', 'B'] = 2222
# 根据条件赋值
df.B[df.A > 4] = 0
# 整列赋值
df['F'] = np.nan
df['E'] = pd.Series([1, 2, 3, 4, 5, 6])  # 长度要一致
# 多列赋值
a = [['a', '1.2', '4.2'], ['a', '70', '0.03'], ['b', '5', '0']]
df = pd.DataFrame(a, columns=['col1', 'col2', 'col3'])
df.col1[df.col1 == 'a'] = 'm'
```

```
data['type'].str.contains('red')
data['type'].isin(['red', 'yellow'])
```

布尔索引取反

python 内置 sum 对布尔索引求和，即计算其总数

```
abnormal = (data[areas[1]] < 10) | (data[areas[1]] > 1e4)
data = data[~abnormal]
sum(abnormal)
```

统计某一列中不同种类的个数：

```
data['Type'].value_counts()[0:20]
```

df 转换为 array

```
df.values
```

处理缺失值

```
df2.dropna(how='any')  # 非原位操作
df2.fillna(df2.mean())
```

绘图

```
np.random.seed(999)
df = pd.DataFrame(np.random.rand(10,4), columns = ['a', 'b', 'c', 'd'])
df.plot()
df.plot.bar()
df.plot.bar(stacked=True)
df.T  # 转置
```

拼接

```
pd.concat(objs, axis=0, join='outer', join_axes=None, ignore_index=False,
          keys=None, levels=None, names=None, verify_integrity=False)

# 用 a 作为键内交，是默认的
df1.merge(df2, how='inner', on='a')
# how: left, right, cross
```

序列化

使用 pandas 直接序列化

```
import pandas as pd
df = pd.DataFrame([range(11), range(100,110)], columns=list('abcdefghijk'))
df.to_pickle('my_df.pickle')
df2 = pd.read_pickle('my_df.pickle')
```

统计

- Python 数据分析之 pandas 统计分析

5.2.5 Numpy

NumPy 数据类型

```
numpy.array(object, dtype = None, copy = True, order = None, subok = False, ndmin =␣
↪0)  # -> ndarray

ndarray.ndim # 数组维度
ndarray.dtype
ndarray.itemsize # 数组元素的内存大小
ndarray.size # 数组大小，flatten 后长度 shape: (3, 2) -> size: 6
b.size * b.itemsize  # 数组占内存
ndarray.shape # 数组形状

# 重构数组对象
ndarray.reshape(2,3)
```

数据类型对象：dtype

创建 dtype 对象，用于指定数组的数据类型

```
dt = np.dtype(np.int32)
```

创建结构化数据类型

```
dt = np.dtype([('age', np.int8)]) # 格式是：[('字段名', 字段类型)]
a = np.array([(10,),(20,),(30,)], dtype = dt) # -> [(10,) (20,) (30,)]
a['age'] # 字段名可用于访问列的内容
```

更好的例子

```
student = np.dtype([('name','S20'), ('age', 'i1'), ('marks', 'f4')])
a = np.array([('abc', 21, 50),('xyz', 18, 75)], dtype = student)
```

修改数据类型

```
arr.astype()
```

结构化数据类型 dtype

读者应该优先使用 dtype 对象来表示数据类型，而不是这些字符编码。‘b’ – boolean ‘i’ – (signed) integer ‘u’ – unsigned integer ‘f’ – floating-point ‘c’ – complex-floating point ‘m’ – timedelta ‘M’ – datetime ‘O’ – (Python) objects ‘S’, ‘a’ – (byte-)string ‘U’ – Unicode ‘V’ – raw data (void)

dtype 对象

```
bool
inti  # 平台决定精度的整数
int8, int16, int32, int64,
uint8, uint16, uint32, uint64
float16, float32
float64  # or float
complex64
complex128  # complex
```

创建 NumPy 数组

由已有对象创建

```
list = [1,2,3,4]
arr = np.array(list)
arr = np.asarray(list) # 只要是序列即可
# 使用可迭代对象创建ndarray数组
it = iter(list)
x = np.fromiter(it, dtype = float, count = 2)
```

使用 NumPy 函数创建

```
np.empty((3, 2), dtype = int)
np.zeros((2, 3), order = 'C') # C 是按行排列, F 是按列排列
np.eye(2)
np.ones((2, 3))
np.arange(7) # [0 1 2 3 4 5 6]
np.repeat([1, 2], 2) # [1, 1, 2, 2]
np.tile([1, 2], 2) # [1, 2, 1, 2]
mat = np.matrix('1 2; 3 4')
mat.T
mat.H # 共轭转置
mat.I # 逆矩阵
np.matlib.identity(5)
```

基于数值区间创建数组

```
np.arange(start, stop, step, dtype)
np.linspace(start, stop, num, dtype)
np.logspace(start, stop, num, base, dtype)
```

NumPy 数组操作

数组间的算术运算

```
a = np.array([[1,2,30],[10,15,4]])
b = np.array([[1,2,3],[12, 19, 29]])
a+b
a*b
a/b
```

数组拼接

```
np.vstack((a,b))  # 上下
np.hstack((a,b))  # 水平
```

数组切片

```
arr_name[start: end: step]
# 均匀分布与linspace
np.linspace(5,15,10)
# 通过内置slice函数创建一个切片
a = np.arange(10)
s = slice(2,7,2)
a[s]
```

数组索引与变换

，区分不同的维度多个冒号可以用一个省略号 `...` 来代替 `b[0, :, :] = b[0, ...]`

```
a[:,1] # 取列
a[:,[1,2]] # 取多列
a[1,:] # 取行
a[[1,2],:] # 取多行
a[1:1] # 取元素
a[a[:,1]>2,] # 单条件过滤
a[(a[:,1]>2) & (a[:,1]<4),] # 多条件过滤
a.T # 转置
```

```
np.transpose(a) # 转置
a.flatten() # 展平，返回拷贝
a.ravel() # 展平，返回视图，变则原变
a.shape = (6,4)  # 直接设置形状
```

(下页继续)

(续上页)

```
a.reshape((6,4))  # 返回拷贝
a.resize((6,4))  # 返回视图
```

数组的组合

```
# 水平组合
hstack((a, b))
concatenate((a, b), axis=1)
# 垂直组合
vstack((a, b))
concatenatel((a, b), axis=0)
dstack((a, b))  # 深度堆积
```

数组的分隔

```
hsplit(a, 3)  # 三个子数组
split(a, 3, axis=1)
vsplit(a, 3)
split(a, 3, axis=0)
dplit(a, 3)
```

数组的转换

```
b.tolist()
a.astype(int)  # 修改数组类型
```

数组的排序

```
np.sort(a)
a.sort()
a.sort(axis=0) # 按列排序
np.argsort(a) # 返回排序后的下标
a[np.argsort(-a)] # 数组的降序
```

NumPy 数组迭代

```
a = np.array([[1,2,3,4],[2,4,5,6],[10,20,39,3]])
# 可通过修改 order 来指定迭代方向
for x in np.nditer(at):
    print(x, end= ' ')
# 二重枚举迭代与广播
for x,y in np.nditer([a,b]):
    print ("%d:%d" % (x,y))
```

NumPy 数组函数

```
ndarray.max(axis = 0) # 对列进行操作，输出单行
ndarray.min()
ndarray.sum()
```

NumPy 字符串操作

```
np.char.add(['hello'],[' qikegu.com'])
np.char.add(['hello', 'hi'],[' qikegu.com', ' kevin'])
np.char.multiply('qikegu ', 3)
np.char.center('qikegu', 20, fillchar = '*') # -> ******qikegu********
np.char.capitalize()
np.char.title()
np.char.lower()
np.char.upper()
np.char.split()
np.char.splitlines() # 返回字符串中的行列表，在行边界处断开。
np.char.strip()
np.char.join(':','dmy')
np.char.join([':','-'],['dmy','ymd'])
np.char.replace()

# 字符解码与编码
a = np.char.encode('qikegu', 'gb2312')
print (a)
print (np.char.decode(a,'gb2312'))
```

NumPy 排序、查找、计数

| 种类 | 速度 | 最差情况 | 工作区 | 稳定性 | | :———| :—| :———-| :—| :—| | quicksort | 1 | O(n^2) | 0 | no | | mergesort | 2 | O(n*log(n)) | ~*n/2* | *yes* | | *heapsort* | *3* | *O(n*log(n)) | 0 | no |

```
np.sort(a, axis, kind, order) # kind='quicksort', e.g. order = 'name'
np.argsort() # 返回数组排序后的索引，用于重构数组
np.lexsort() # 多序列排序，返回数组排序后的索引，用于重构数组
np.argmax() # -> Index
np.argmin() # -> Index
np.nonzero() # -> Index
np.where() # 查找数组中符合条件的元素，返回其索引
numpy.extract() # # -> 符合条件的元素
```

NumPy 函数

NumPy 数学函数

```
np.sqrt()
np.std()

# 三角函数
sin()
```

(下页继续)

(续上页)

```
cos()
tan()
arcsin()
arccos()
arctan()
degrees() # 查看度数结果
around(num, decimals)
floor()
ceil()
```

NumPy 统计函数

```
a = np.array([[2,10,20],[80,43,31],[22,43,10]])
# 查找指定轴上，数组元素的最小值和最大值
np.amin(a,0) # 对列进行操作
np.amax()

# 返回数组某个轴方向的峰间值，即最大值最小值之差
np.ptp(a,1)

# 百分位数是统计中使用的度量，表示小于这个值的观察值占总数的百分比。
np.percentile(input, q, axis) # q: 要计算的百分位数，在 0 ~ 100 之间

# 计算数组项的中值、平均值、加权平均值
np.median() # 中值
np.mean() # 平均值

## 加权平均值
wt = np.array([0, 0, 10])
np.average(a, axis=1, weights = wt))

# 标准差与方差
np.std([1,2,3,4])
np.var([1,2,3,4])
```

其他

随机

```
np.random.rand(10, 1)  # 二维数据
np.random.rand(10)  # 一维数据
# 指定集合和概率的随机
np.random.choice([3, 5, 7, 9],
                         p=[0.1, 0.3, 0.6, 0.0],
                         size=(100))
```

```
np.random.seed(99)
arr = np.array([1, 2, 3, 4, 5])
# 随机排列
new_arr = np.random.permutation(arr)
# shuffle
np.random.shuffle(arr)  # 原位操作
```

随机分布

```
# 正态分布
x = np.random.normal(loc=1, scale=2, size=(2, 3))
# 二项式分布
np.random.binomial(n=10, p=0.5, size=10)
# 多项式分布
np.random.multinomial(n=6, pvals=[1/6, 1/6, 1/6, 1/6, 1/6, 1/6])
```

NumPy 副本和视图

NumPy 数组赋值，传的是指针。

```
ndarray.view() # shallow copy, 类似指针传值
ndarray.copy() # deep copy, 内存独立
```

NumPy 矩阵库函数

numpy.matlib 模块

```
numpy.matlib.empty(shape, dtype, order)
numpy.matlib.zeros()
numpy.matlib.ones()
numpy.matlib.eye() # mn不等单位矩阵
numpy.matlib.identity()
```

NumPy 线性代数

numpy.linalg 模块

| SN | 函数 | 描述 | | :—| :————————- | :————————| | 1 | dot() 两个数组的点积 | | 2 | vdot() | 两个向量的点积 | | 3 | inner() | 两个数组的内积 | | 4 | matmul() | 两个数组的矩阵乘积 | | 5 | det() | 计算矩阵的行列式 | | 6 | solve() | 解线性矩阵方程 | | 7 | inv() | 求矩阵的乘法逆矩阵 |

5.2.6 ZipFile

使用

```
import zipfile

zipFile = zipfile.ZipFile(file_dir, mode="r")

zipFile.infolist()
zipFile.namelist()
zipFile.printdir()

# 解压文件
zipFile = zipfile.ZipFile(file_dir)
for file in zipFile.namelist():
    zipFile.extract(file, 'to_dir')
```

(下页继续)

(续上页)

```
zipFile.close()

## 解压文件 简单版
zipFile.extractall('to_dir')
```

```
import zipfile
import os

def make_zip(source_dir, output_filename):
    """压缩目录中的文件
    source_dir 中不能包含目录
    example:
        make_zip('data', 'data.zip')
    """
    zipf = zipfile.ZipFile(output_filename, 'w')
    pre_len = len(os.path.dirname(source_dir))
    for parent, _, filenames in os.walk(source_dir):
        for filename in filenames:
            pathfile = os.path.join(parent, filename)
            arcname = pathfile[pre_len:].strip(os.path.sep)
            zipf.write(pathfile, arcname)
    zipf.close()
```

5.2.7 Pickle

Pickling 允许您将 python 对象保存为硬盘驱动器上的二进制文件。

一句警告：不要加载你不信任的 pkl 文件。恶意的人可以制作恶意的 pkl 文件，可能会在您的计算机上执行意外的代码（SQL 注入，密码暴力强制等）。

Pickle 用于序列化和反序列化 Python 对象结构，也称为 marshalling 或 flattening。Pickle 不要与压缩相混淆！前者是将对象从一种表示（随机存取存储器（RAM）中的数据）转换为另一种表示（磁盘上的文本），而后者是使用较少位编码数据的过程，以节省磁盘空间。

代码示例

```
import pickle
# make an example object to pickle
some_obj = {'x':[4,2,1.5,1], 'y':[32,[101],17], 'foo':True, 'spam':False}

# 写入 pkl
with open('mypickle.pickle', 'wb') as f:
    pickle.dump(some_obj, f)
# note that this will overwrite any existing file
# in the current working directory called 'mypickle.pickle'

# 读取 pkl
with open('mypickle.pickle') as f:
    loaded_obj = pickle.load(f)
print('loaded_obj is', loaded_obj)
```

使用 pandas 直接序列化

```
import pandas as pd
df = pd.DataFrame([range(11), range(100,110)], columns=list('abcdefghijk'))
df.to_pickle('my_df.pickle')
df2 = pd.read_pickle('my_df.pickle')
```

5.2.8 dill

```
pip install dill
```

```
import dill as pickle

with open('kilgariff_ngram_model.pkl', 'wb') as fout:
    pickle.dump(model, fout)

with open('kilgariff_ngram_model.pkl', 'rb') as fin:
    model_loaded = pickle.load(fin)
```

5.3 Python Applications

5.3.1 NLP

5.4 Python 网络

5.4.1 Python 协程与异步 IO

python 一直在进行并发编程的优化，比较熟知的是使用 thread 模块多线程和 multiprocessing 多进程，后来慢慢引入基于 yield 关键字的协程。

对于 IO 密集型任务我们有一种选择就是协程。协程，又称微线程，英文名 Coroutine，是运行在单线程中的“并发”，协程相比多线程的一大优势就是省去了多线程之间的切换开销，获得了更高的运行效率。Python 中的异步 IO 模块 asyncio 就是基本的协程模块。

Python 中的协程经历了很长的一段发展历程。最初的生成器 yield 和 send() 语法，然后在 Python3.4 中加入了 asyncio 模块，引入 @asyncio.coroutine 装饰器和 yield from 语法，在 Python3.5 上又提供了 async/await 语法，目前正式发布的 Python3.6 中 asynico 也由临时版改为了稳定版。

yield 是迭代器，yield + send 是协程。此时 yield 语句不再只是 yield xxxx 的形式，还可以是 var = yield xxxx 的赋值形式。它同时具备两个功能，一是暂停并返回函数，二是用 var 接收外部 send() 方法发送过来的值，重新激活函数。

协程

协程的切换不同于线程切换，是由程序自身控制的，没有切换的开销。协程不需要多线程的锁机制，因为都是在同一个线程中运行，所以没有同时访问数据的问题，执行效率比多线程高很多。

因为协程是单线程执行，那怎么利用多核 CPU 呢？最简单的方法是多进程 + 协程，既充分利用多核，又充分发挥协程的高效率，可获得极高的性能。

进程/线程：操作系统提供的一种并发处理任务的能力。

协程：程序员通过高超的代码能力，在代码执行流程中人为的实现多任务并发，是单个线程内的任务调度技巧。

多进程和多线程体现的是操作系统的能力，而协程体现的是程序员的流程控制能力。

因为 send() 方法的参数会成为暂停的 yield 表达式的值，所以，仅当协程处于暂停状态时才能调用 send() 方法，例如 my_coro.send(10)。不过，如果协程还没激活（状态是’GEN_CREATED’），就立即把 None 之外的值发给它，会出现 TypeError。因此，始终要先调用 next(my_coro) 激活协程（也可以调用 my_coro.send(None)），这一过程被称作预激活。

@asyncio.coroutine 与 yield from

`yield from range(10)` 等价于 `for i in range(10): yield i`

`yield from` 其实就是等待另外一个协程的返回

```
import asyncio
import datetime

@asyncio.coroutine  # 声明一个协程
def display_date(num, loop):
    end_time = loop.time() + 10.0
    while True:
        print("Loop: {} Time: {}".format(num, datetime.datetime.now()))
        if (loop.time() + 1.0) >= end_time:
            break
        yield from asyncio.sleep(2)  # 阻塞直到协程 sleep(2) 返回结果

loop = asyncio.get_event_loop()  # 获取一个event_loop
tasks = [display_date(1, loop), display_date(2, loop)]

loop.run_until_complete(asyncio.gather(*tasks))  # "阻塞"直到所有的 tasks 完成
loop.close()
```

async 和 await

推荐使用 Python3.5 中对协程提供了更直接的支持，引入了 async/await 关键字。上面的代码可以这样改写：使用 async 代替 @asyncio.coroutine，使用 await 代替 yield from，代码变得更加简洁可读。从 Python 设计的角度来说，async/await 让协程独立于生成器而存在，不再使用 yield 语法。

```
import asyncio
import datetime

async def display_date(num, loop):       # 注意这一行的写法
    end_time = loop.time() + 10.0
    while True:
```

(下页继续)

(续上页)

```
        print("Loop: {} Time: {}".format(num, datetime.datetime.now()))
        if (loop.time() + 1.0) >= end_time:
            break
        await asyncio.sleep(2)  # 阻塞直到协程 sleep(2) 返回结果

loop = asyncio.get_event_loop()  # 获取一个 event_loop
tasks = [display_date(1, loop), display_date(2, loop)]

loop.run_until_complete(asyncio.gather(*tasks))  # "阻塞"直到所有的 tasks 完成
```

asyncio 模块

asyncio 的使用可分三步走

- 创建事件循环
- 指定循环模式并运行
- 关闭循环

通常我们使用 asyncio.get_event_loop() 方法创建一个循环。

运行循环有两种方法：一是调用 run_until_complete() 方法，二是调用 run_forever() 方法。run_until_complete() 内置 add_done_callback 回调函数，run_forever() 则可以自定义 add_done_callback()，具体差异请看下面两个例子。

使用 run_until_complete() 方法：

```
import asyncio

async def func(future):
    await asyncio.sleep(1)
    future.set_result('Future is done!')

if __name__ == '__main__':

    loop = asyncio.get_event_loop()
    future = asyncio.Future()
    asyncio.ensure_future(func(future))
    print(loop.is_running())    # 查看当前状态时循环是否已经启动
    loop.run_until_complete(future)
    print(future.result())
    loop.close()
```

使用 run_forever() 方法：

```
import asyncio

async def func(future):
    await asyncio.sleep(1)
    future.set_result('Future is done!')

def call_result(future):
    print(future.result())
    loop.stop()

if __name__ == '__main__':
```

(下页继续)

(续上页)

```
    loop = asyncio.get_event_loop()
    future = asyncio.Future()
    asyncio.ensure_future(func(future))
    future.add_done_callback(call_result)        # 注意这行
    try:
        loop.run_forever()
    finally:
        loop.close()
```

参考

协程与异步 IO

5.4.2 Socket

Python 的 socket 模块提供了与 Berkeley 套接字 API 的接口。

`socket.SOCK_STREAM` 用于为 TCP 创建套接字，而 `socket.SOCK_DGRAM` 为 UDP 创建套接字

创建套接字时，必须指定其地址族。然后，我们只能在套接字中使用该类型的地址。

- AF_UNIX，AF_LOCAL-本地通讯
- AF_INET-IPv4 Internet 协议
- AF_INET6-IPv6 Internet 协议
- AF_IPX-IPX-Novell 协议
- AF_BLUETOOTH-无线蓝牙协议
- AF_PACKET-底层数据包接口

对于 `AF_INET` 地址族，指定了一对（主机，端口）。`host` 是一个字符串，表示互联网域表示法中的主机名（如 `example.com`）或 IPv4 地址（如 `93.184.216.34` ），并且 port 是整数。

```
import socket

# 获取 IP 地址
ip = socket.gethostbyname('example.com')
print(ip)

# UDP 套接字示例
with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as s:

    message = b''
    addr = ("djxmmx.net", 17)

    s.sendto(message, addr)

    data, address = s.recvfrom(1024)
    print(data.decode())

# TCP 套接字示例
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:

    host = "time.nist.gov"
```

(下页继续)

(续上页)

```
    port = 13

    s.connect((host, port))
    s.sendall(b'')
    print(str(s.recv(4096), 'utf-8'))
```

See also

- Python 套接字教程
- Python 网络编程

5.5 Python 效率

5.5.1 Joblib

对于大多数问题，并行计算确实可以提高计算速度。随着 PC 计算能力的提高，我们可以通过在 PC 中运行并行代码来简单地提升计算速度。Joblib 就是这样一个可以简单地将 Python 代码转换为并行计算模式的软件包，它可非常简单并行我们的程序，从而提高计算速度。

Joblib 是一组用于在 Python 中提供轻量级流水线的工具。它具有以下功能：

- 透明的磁盘缓存功能和“懒惰”执行模式，简单的并行计算
- Joblib 对 numpy 大型数组进行了特定的优化，简单，快速。

使用

输出值的透明快速磁盘缓存

```
from joblib import Memory
cachedir = 'your_cache_dir_goes_here'
mem = Memory(cachedir)
import numpy as np
a = np.vander(np.arange(3)).astype(np.float)
square = mem.cache(np.square)
b = square(a)
c = square(a)
```

并发

```
from joblib import Parallel, delayed
# n_jobs is the number of parallel jobs
Parallel(n_jobs=2)(delayed(my_fun)(i, j ) for i in range(num))
```

快速压缩持久化

替代 pickle，有效地处理包含大数据的 Python 对象 (joblib)。

```
from joblib import dump, load

pickle_file = './pickle_data.joblib'
with open(pickle_file, 'wb') as f:
    dump(data, f)
    dump(data, f, compress='zlib')

with open(pickle_file, 'rb') as f:
    data_stored = load(f)
```

参考

5.5.2 Linecache

linecache 模块的作用是将文件内容读取到内存中，进行缓存，而不是每次都要从硬盘中读取，这样效率提高很多，又省去了对硬盘 IO 控制器的频繁操作。

linecache 里面最常用到的就是 getline 方法，简单实用可以直接从内容中读到指定的行，日常编程中如果涉及读取大文件，一定要使用首选 linecache 模块，相比 open() 那种方法要快 N 倍，它是你读取文件的效率之源。

```
linecache.getlines('poetry.lock')
linecache.getline('poetry.lock', 2)  # 注意 2 是行号，返回的字符串包含换行符
```

实战

```
import linecache

line: str = linecache.getline(self._file_path, idx + 1).strip()
```

5.6 Python Shell

5.6.1 shutil

shutil 可以简单地理解为 sh + util，shell 工具的意思。shutil 模块是对 os 模块的补充，主要针对文件的拷贝、删除、移动、压缩和解压操作。

copyfileobj 是 shutil 最基础的函数 copy 文件内容到另一个文件，可以 copy 指定大小的内容

```
import shutil
s = open('fsrc.txt','r')
d = open('fdst.txt','w')
shutil.copyfileobj(s,d,length=16*1024)
shutil.copyfile(src, dst) # 拷贝文件
shutil.copymode(src, dst) # 仅拷贝权限
shutil.copystat(src, dst) # 仅复制所有的状态信息，包括权限，组，用户，时间等。
shutil.copy(src,dst) # 同时复制文件的内容以及权限，也就是先 copyfile() 然后 copymode()
shutil.copy2(src,dst) # 复制所有信息和内容，先 copyfile() 后 copystat()
```

文件夹操作

```
# 递归地复制目录及其子目录的文件和状态信息
shutil.copytree(src, dst)
# 忽略文件地复制
copytree(source, destination, ignore=ignore_patterns('*.pyc', 'tmp*'))
# 递归删除文件
shutil.rmtree(path[, ignore_errors[, onerror]])
# 递归地移动文件，类似mv命令
shutil.move(src, dst)
```

其他

```
shutil.wich('echo')
```

创建归档或压缩文件，或其逆操作

```
shutil.make_archive("d:\\3", "zip",  base_dir="d:\\1.txt")
shutil.unpack_archive("d:\\3.zip", "f:\\3", 'zip')
```

```
import shutil
source = "/home/User/Documents/file.txt"
destination = "/home/User/Documents/file.txt"

try:
    shutil.copy(source, destination)
    print("File copied successfully.")

# If source and destination are same
except shutil.SameFileError:
    print("Source and destination represents the same file.")

# If there is any permission issue
except PermissionError:
    print("Permission denied.")

# For other errors
except:
    print("Error occurred while copying file.")
```

参考

shutil | 刘江的博客教程

5.6.2 Python Argparse

通过 `argparse` 模块，可以轻松编写用户友好的命令行界面。它解析 `sys.argv` 中定义的参数。

`argparse` 模块还会自动生成帮助和使用消息，并在用户为程序提供无效参数时发出错误。

`argparse` 是标准模块；我们不需要安装它。

使用 `ArgumentParser` 创建一个解析器，并使用 `add_argument()` 添加一个新参数。参数可以是可选的，必需的或定位的。

建议阅读一下参考指定的教程，能帮助你理解

add_argument

```
defaut=2
default='hello world'
metavar='value' # 为错误的期望值命名，并提供帮助输出
required=True # 设置为必须参数
type=int # 指定类型
help="shows output" # 显示帮助信息
nargs=2 # 该选项需要两个参数
nargs=* # 该选项需要好多个参数
choices=['std', 'iso', 'unix', 'tz']

action='append' # 这个选项可以重复出现多次
action='store_true' # 保存为真假
```

简单的例子

```
#!python3

import argparse
parser = argparse.ArgumentParser()

# 可选参数
## dest 指定参数的在 python 中的名称，不指定 argparse 会自动推导
## action 的 store_true 会将参数存储为 True
parser.add_argument('-o', '--output', dest='output_dest', action='store_true',
                    help="shows output")



# 必须参数
parser.add_argument('--name', required=True)
## type 指定参数的类型
parser.add_argument('-n', type=int, required=True,
                    help="define the number of random integers")

# 位置参数
parser.add_argument('age')

# append 操作
parser.add_argument('-n', '--name', dest='names', action='append',
                    help="provides names to greet")
names = args.names
for name in names:
    print(f'Hello {name}!')

args = parser.parse_args()

if args.output:
    print("This is some output")


print(f'Hello {args.name}')
```

子命令

子命令的例子：`pip install` 和 `pip uninstall`

一个用子命令的实例：

```
# sub-command functions
def foo(args):
        print(args.x * args.y)
def bar(args):
        print('((%s))' % args.z)

# create the top-level parser
parser = argparse.ArgumentParser()
subparsers = parser.add_subparsers()

# create the parser for the "foo" command
parser_foo = subparsers.add_parser('foo')
parser_foo.add_argument('-x', type=int, default=1)
parser_foo.add_argument('y', type=float)
parser_foo.set_defaults(func=foo)

# create the parser for the "bar" command
parser_bar = subparsers.add_parser('bar')
parser_bar.add_argument('z')
parser_bar.set_defaults(func=bar)

# parse the args and call whatever function was selected
args = parser.parse_args()
args.func(args)
```

参考

Python argparse 教程

5.6.3 Subprocess

Python 运行 Shell，推荐使用 `subprocess`。

```
#只返回结果
import subprocess
subprocess.call("echo hello world", shell=True)
```

`os.system` 方法会创建子进程运行外部程序，方法只返回外部程序的运行结果

```
import os
# #只返回结果
print(os.system("echo hello world")) # 还会多输出一行返回状态
```

`os.popen` 方不仅仅返回结果，还返回一个类文件对象，通过调用该对象的 `read()` 或 `readlines()` 方法可以读取输出内容

```
output = os.popen('echo hello world', 'r')
print(output.read())
```

实战

```
self._len_source: int = int(subprocess.check_output("wc -l " + self._source_path,␣
↪shell=True).split()[0])
```

CHAPTER 6

Chem

6.1 Fingerprints

6.1.1 MolVS

MolVS is a molecule validation and standardization tool, written in Python using the RDKit chemistry framework.

```
conda install molvs
pip install MolVS
```

```
from molvs import standardize_smiles
standardize_smiles('[Na]OC(=O)c1ccc(C[S+2]([O-])([O-]))cc1')
'[Na+].O=C([O-])c1ccc(CS(=O)=O)cc1'
```

6.2 Representations

6.2.1 selfies

为什么 selfies 比 smiles 更适合生成

The SELFIES grammar and bond constraints enforce chemical valency rules, guaranteeing that generated SELFIES are syntactically and semantically valid, without requiring post-hoc corrections or complex model architectures that are difficult to train.

使用

```
pip install selfies
# check if the correct version
pip show selfies
# if not
pip install selfies --upgrade
```

```
selfies.encoder
selfies.decoder
selfies.set_semantic_constraints
selfies.len_selfies
selfies.split_selfies
selfies.get_alphabet_from_selfie
selfies.selfies_to_encoding
selfies.encoding_to_selfies
```

使用

```
import selfies
smi = 'CC[N+](C)(C)Cc1ccccc1Br'
sel = selfies.encoder(smi)
print(f'SELFIES string: {sel}')
>>> SELFIES string: [C][C][N+][Branch1_2][epsilon][C][Branch1_
↪3][epsilon][C][C][c][c][c][c][c][c][Ring1][Branch1_1][Br]
toks = atomwise_tokenizer(sel)
print(toks)
>>> ['[C]', '[C]', '[N+]', '[Branch1_2]', '[epsilon]', '[C]', '[Branch1_3]',
↪'[epsilon]', '[C]', '[C]', '[c]', '[c]', '[c]', '[c]', '[c]', '[c]', '[Ring1]',
↪'[Branch1_1]', '[Br]']

toks = kmer_tokenizer(sel, ngram=4)
print(toks)

>>> ['[C][C][N+][Branch1_2]', '[C][N+][Branch1_2][epsilon]', '[N+][Branch1_
↪2][epsilon][C]', '[Branch1_2][epsilon][C][Branch1_3]', '[epsilon][C][Branch1_
↪3][epsilon]', '[C][Branch1_3][epsilon][C]', '[Branch1_3][epsilon][C][C]',
↪'[epsilon][C][C][c]', '[C][C][c][c]', '[C][c][c][c]', '[c][c][c][c]', '[c][c][c][c]
↪', '[c][c][c][c]', '[c][c][c][Ring1]', '[c][c][Ring1][Branch1_1]',
↪'[c][Ring1][Branch1_1][Br]']
```

See also

- GitHub repo

6.2.2 SMILES

SmilesPE

```
pip install SmilesPE
```

```
from SmilesPE.pretokenizer import atomwise_tokenizer

smi = 'CC[N+](C)(C)Cc1ccccc1Br'
toks = atomwise_tokenizer(smi)
print(toks)
```

1. K-mer Tokenzier

```
from SmilesPE.pretokenizer import kmer_tokenizer

smi = 'CC[N+](C)(C)Cc1ccccc1Br'
toks = kmer_tokenizer(smi, ngram=4)
print(toks)
```

6.2.3 deepsmiles

```
import deepsmiles
converter = deepsmiles.Converter(rings=True, branches=True)
smi = 'CC[N+](C)(C)Cc1ccccc1Br'
deepsmi = converter.encode(smi)
print(f'DeepSMILES string: {deepsmi}')> >> DeepSMILES string: CC[N+]C)C)Ccccccc6Br
toks = atomwise_tokenizer(deepsmi)
print(toks)

>>> ['C', 'C', '[N+]', 'C', ')', 'C', ')', 'C', 'c', 'c', 'c', 'c', 'c', 'c', '6', 'Br
↪']

toks = kmer_tokenizer(deepsmi, ngram=4)
print(toks)

>>> ['CC[N+]C', 'C[N+]C)', '[N+]C)C', 'C)C)', ')C)C', 'C)Cc', ')Ccc', 'Cccc', 'cccc',
↪'cccc', 'cccc', 'ccc6', 'cc6Br']
```

CHAPTER 7

Bio: Life Language Processing

7.1 多肽

7.1.1 多肽的结构分析方法

质谱

核磁共振

红外光谱

CHAPTER 8

AI

8.1 Prerequisites

8.1.1 Multimodal

特征融合

多模态特征融合的方法大体分为三种：前端融合、中间融合和后端融合。

前端融合

将多个独立的数据集融合成一个单一的特征向量，然后输入到机器学习分类器中。

多模态前端融合方法常常与特征提取方法相结合以剔除冗余信息，如主成分分析（PCA）、最大相关最小冗余算法（mRMR）、自动解码器（Autoencoders）等。

本人研究的是使用深层联合自编码模型，将三种模态的特征使用三层线性层将维度转化为同一维度，然后相加，最后将三者进行还原回去。

中间融合

指的是将不同的模态数据先转化为高维特征表达，再于模型的中间层进行融合。以神经网络为例，中间融合首先利用神经网络将原始数据转化成高维特征表达，然后获取不同模态数据在高维空间上的共性。在问答对话中有 MFB 方法，它针对文本和图像两种模态，先将每个模态特征转化为相同维度的高维向量，然后进行逐元素相乘，最后进行 sum pooling 操作。

后端融合

指的是将不同模态数据分别训练好的分类器输出打分 (决策) 进行融合。常见的后端融合方式包括最大值融合 (max-fusion)、平均值融合 (averaged-fusion)、贝叶斯规则融合 (Bayes’ rule based) 以及集成学习 (ensemble learning) 等。

下面介绍比较实用的专门总结多模态融合文章的网址（里面都是关于多模态的高水平论文）：

里面会注明是否含有开源代码，文章出处，很齐全：网址：https://github.com/pliang279/awesome-multimodal-ml#multimodal-fusion

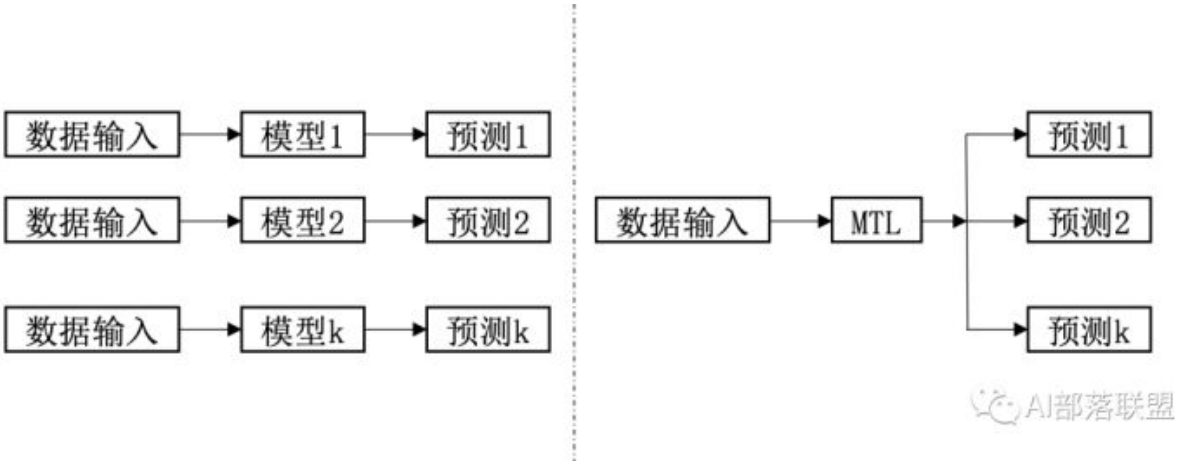
综述文章：

深度多模态表征学习：一项调查，该文章通过对深度学习中多模态数据方法进行总结和讨论，分析方法种类和各自优缺点。网址：https://ieeexplore.ieee.org/abstract/document/8715409

8.1.2 Multi-Task

多任务学习（multi task learning）简称为 MTL

单任务学习 VS 多任务学习



单任务学习（single task learning）：一个 loss，一个任务，例如 NLP 里的情感分类、NER 任务一般都是可以叫单任务学习。大部分的机器学习任务都属于单任务学习。

多任务学习：简单来说有多个目标函数 loss 同时学习的就算多任务学习。

多任务学习（Multitask Learning）是一种推导迁移学习方法，主任务（main tasks）使用相关任务（related tasks）的训练信号（training signal）所拥有的领域相关信息（domain-specific information），做为一直推导偏差（inductive bias）来提升主任务（main tasks）泛化效果（generalization performance）的一种机器学习方法。

多任务学习（multitask learning）产生的原因?

现在大多数机器学习任务都是单任务学习。对于复杂的问题，也可以分解为简单且相互独立的子问题来单独解决，然后再合并结果，得到最初复杂问题的结果。这样做看似合理，其实是不正确的，因为现实世界中很多问题不能分解为一个一个独立的子问题，即使可以分解，各个子问题之间也是相互关联的，通过一些共享因素或共享表示（share representation）联系在一起。把现实问题当做一个个独立的单任务处理，忽略了问题之间所富含的丰富的关联信息。多任务学习就是为了解决这个问题而诞生的。把多个相关（related）的任务（task）放在一起学习。这样做真的有效吗？答案是肯定的。多个任务之间共享一些因素，它们可以在学习过程中，共享它们所学到的信息，这是单任务学习所具备的。相关联的多任务学习比单任务学习能去的更好的泛化（generalization）效果。

共享表示

共享表示的目的是为了提高泛化（improving generalization），图 2 中给出了多任务学习最简单的共享方式，多个任务在浅层共享参数。MTL 中共享表示有两种方式：

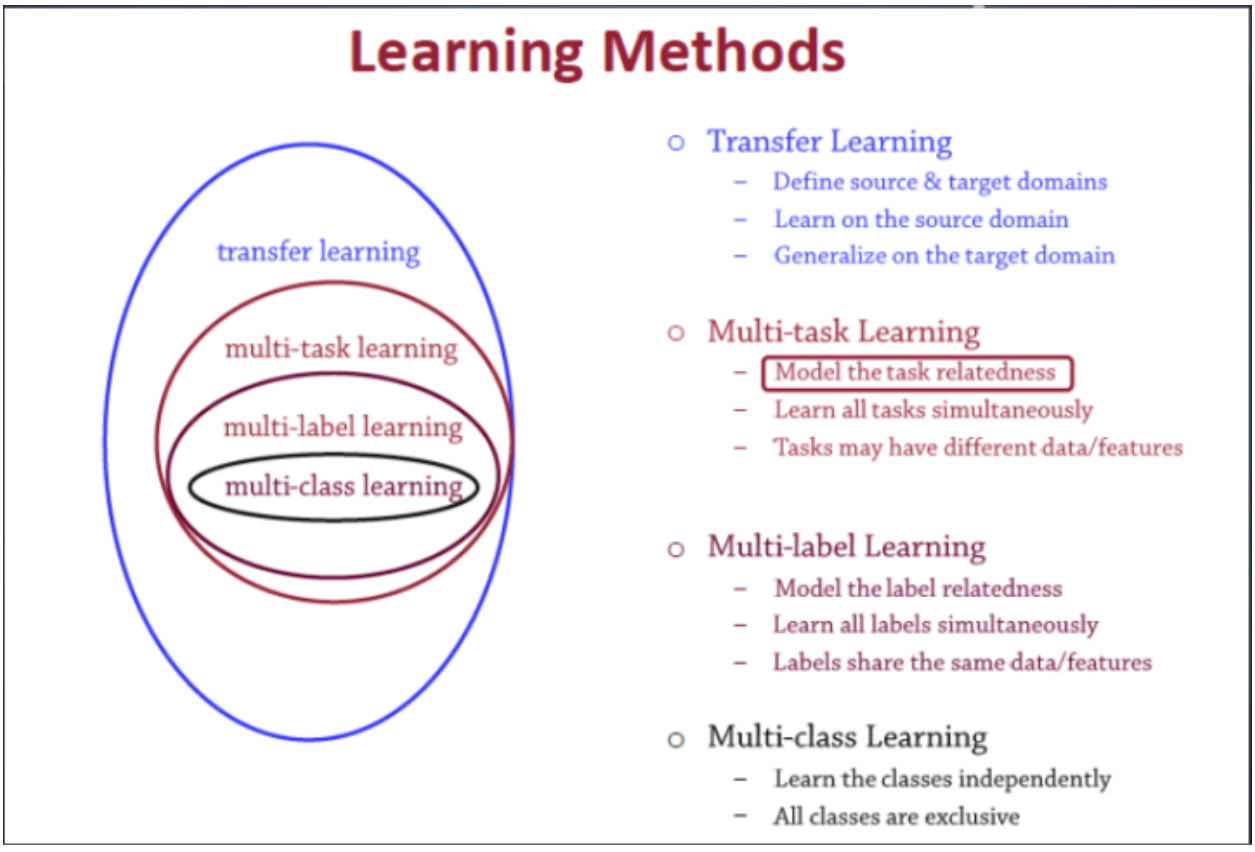
基于参数的共享（Parameter based）：比如基于神经网络的 MTL，高斯处理过程。

基于约束的共享（regularization based）：比如均值，联合特征（Joint feature）学习（创建一个常见的特征集合）。

多任务学习与其他学习算法之间的关系

多任务学习（Multitask learning）是迁移学习算法的一种，迁移学习之前介绍过。定义一个一个源领域 source domain 和一个目标领域（target domain），在 source domain 学习，并把学习到的知识迁移到 target domain，提升 target domain 的学习效果（performance）。

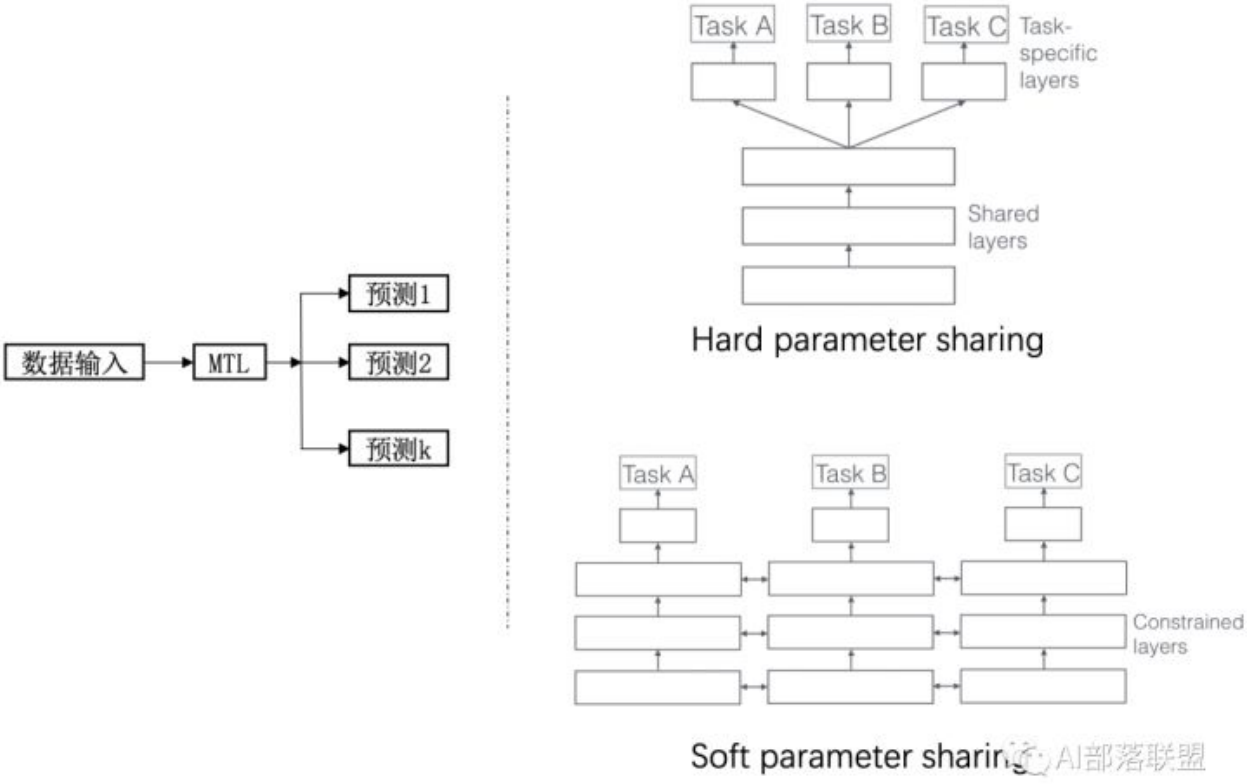
多标签学习（Multilabel learning）是多任务学习中的一种，建模多个 label 之间的相关性，同时对多个 label 进行建模，多个类别之间共享相同的数据/特征。



多类别学习（Multiclass learning）是多标签学习任务中的一种，对多个相互独立的类别（classes）进行建模。这几个学习之间的关系如图 5 所示：

多任务学习的基本模型框架

通常将多任务学习方法分为：hard parameter sharing 和 soft parameter sharing。



一个老当益壮的方法：hard parameter sharing

即便是 2021 年，hard parameter sharing 依旧是很好用的 baseline 系统。

现代研究重点倾向的方法：soft parameter sharing

多任务学习应用概述

基于神经网络的多任务学习，尤其是基于深度神经网络的多任务学习（DL based Multitask Learning），适用于解决很多 NLP 领域的问题，比如把词性标注、句子句法成分划分、命名实体识别、语义角色标注等任务，都可以采用 MTL 任务来解决。

Links

- 收藏 | 浅谈多任务学习（Multi-task Learning）

8.2 AutoML

8.2.1 AutoML

AutoML 介绍

AutoML 框架执行的任务可以被总结成以下几点:

1. 预处理和清理数据。
2. 选择并构建适当的特征。
3. 选择合适的模型。
4. 优化模型超参数。
5. 设计神经网络的拓扑结构（如果使用深度学习）。
6. 机器学习模型后处理。
7. 结果的可视化和展示。

AutoML 的优势

Gartner 在之前的 AI 平台报告中也指出，对于专业数据科学家，AutoML 能够提高他们的工作效率，减少在手动调参方面的时间投入。对于平民数据科学家，或者说更擅长业务领域知识的数据分析人员，AutoML 能帮助他们快速应用机器学习的能力，不需要再掌握机器学习的复杂技术细节，就能达到业界数据科学家的平均水平。在此基础上，很多之前没有机会应用机器学习的项目也开始可能出现正向的 ROI 回报，推动各行各业的 AI 应用落地，而不只是集中在头部场景和 high tech 公司中。

AutoML 框架

AutoGluon

一句话点评：目前最好的 domain specific autoML 框架方案。

Autokeras

AutoKeras 是一个基于 Keras 的 AutoML 系统，只需几行代码就可以实现神经架构搜索（NAS）的强大功能。它由德克萨斯 A&M 大学的 DATA 实验室开发，以 TensorFlow 的 tf.keras API 和 Keras 为基础进行实现。

AutoKeras 可以支持不同的任务，例如图像分类、结构化数据分类或回归等。

Auto-Sklearn

本着入门一般选 SKLearn 的原则，先聊聊 AutoSKLearn 开箱即用的自动化机器学习库。以 scikit-learn 为基础，自动搜索正确的学习算法并优化其超参数。通过元学习、贝叶斯优化和集成学习等搜索可以获得最佳的数据处理管道和模型。

FLAML

A Fast and Lightweight AutoML Library

AutoML 在近年来的各类机器学习和 Kaggle 比赛中获得了不少的成功。而 FLAML 是今年由微软主推的一个全新的高效轻量级自动化机器学习框架。

AutoML 的框架很多，与目前 SOTA 的一些 automl 框架不同的是，FLAML 重点在于 lightweight，将 hyperparameter, learner, sample size 都考虑到 cost 里，而 cost 除了每一个 trial 的 CPU 时间还包括做 cross-validation 或者 hold-out 的时间。

- 微软 AutoML 框架之 FLAML

NNI

从项目的名字可以看出这个框架主要的重心还是在优化神经网络类的模型方面，而且整体的定位是偏通用框架方向。**一句话点评：NN 模型的最佳 autoML 通用框架**

Optuna

Optuna 是一个特别为机器学习设计的自动超参数优化软件框架。传统搜参工具 hyperopt 或者 BayesianOptimization 的进阶版本。轻量级，经验固化的调优 pipeline，设计简单，效率还不错

一句话点评：如果只想要一个快速灵活的调参工具，就选 Optuna。

主要特点

1. 轻量级、多功能和跨平台架构
2. 并行的分布式优化
3. 对不理想实验 (trial) 的剪枝 (pruning)
4. 超参数重要性
5. 集成新的 CMA-ES 采样
6. 集成 MLflow

与第三方框架的集成 (Integration):

1. LightGBM, XGBoost, PyTorch, TensorFlow
2. MLflow
3. Redis

TPOT

TPOT（Tree-based Pipeline Optimization Tool）是一个 Python 自动化机器学习工具，跟 hyperopt 等属于比较早期的 autoML 框架它使用遗传算法（非贝叶斯优化）优化对机器学习的流程进行优化。它也是基于 Scikit-Learn 提供的方法进行数据转换和机器学习模型的构建，但是它使用遗传算法编程进行随机和全局搜索。另外 TPOT 的整体代码风格上也比较随意，不像是工业界的成熟作品，理解起来会有点困难。**一句话点评：可变 pipeline 的构建和代码生成值得学习**。

- TPOT 原理 | 深度解析 AutoML 框架——TPOT：一键生成 ML 代码，释放双手

H2O AutoML

H2O 的 AutoML 可用于在用户指定的时间限制内自动训练和调整许多模型。H2O 提供了许多适用于 AutoML 对象（模型组）以及单个模型的可解释性方法。可以自动生成解释，并提供一个简单的界面来探索和解释 AutoML 模型。

Links

- AutoML 框架概览
- 走马观花 AutoML
- awesome AutoML

8.2.2 AutoGluon

AutoGluon: AutoML for Text, Image, and Tabular Data

```
pip3 install -U pip
pip3 install -U setuptools wheel

pip3 install autogluon
```

8.2.3 MLflow

Links

- 基础概念 | MLflow 教程文档翻译
- mlflow 官方快速教程
- mlflow GitHub

8.3 Transformers

8.3.1 Hugging Face Transformers

AutoClass

`BERT` 是一个 `architecture`

`bert-base-uncased` 是 `checkpoint`

`Model` 是上面的两者

AutoProcessor = AutoTokenizer + AutoFeatureExtractor

AutoProcessor = AutoProcessor + model

8.4 Hugging Face

8.4.1 Tokenizers

常用

直接转成 tensor

```
# Returns PyTorch tensors
model_inputs = tokenizer(sequences, padding=True, return_tensors="pt")
```

```
import torch
from transformers import AutoTokenizer, AutoModelForSequenceClassification

checkpoint = "distilbert-base-uncased-finetuned-sst-2-english"
tokenizer = AutoTokenizer.from_pretrained(checkpoint)
model = AutoModelForSequenceClassification.from_pretrained(checkpoint)
sequences = [
  "I've been waiting for a HuggingFace course my whole life.",
  "So have I!"
]

tokens = tokenizer(sequences, padding=True, truncation=True, return_tensors="pt")
output = model(**tokens)
```

multi 的两个句子长短不一，需要 padding，有三种方法

```
inputs_multi = tokenizer(sequence_multi, padding="longest") # 将序列填充到最大序列长度
inputs_multi = tokenizer(sequence_multi, padding="max_length") #␣
↪将序列填充到模型最大长度 bert 512
inputs_multi = tokenizer(sequence_multi, padding="max_length", max_length=8) #␣
↪将序列填充到指定最大长度
```

See also

- Transformers Tokenizer API 的使用

实战

- PyTorch 的 BERT 微调教程

8.5 OpenNMT

8.6 主动学习

主动学习属于“半监督学习”。用更少的数据做更多的事情。学者们通过一些技术手段或者数学方法来降低人们标注的成本，学者们把这个方向称之为主动学习（Active Learning）。主动学习是机器学习的一个子领域，在统计学领域也叫查询学习或最优实验设计。主动学习方法尝试解决样本的标注瓶颈，通过主动优先选择最有价值的未标注样本进行标注，以尽可能少的标注样本达到模型的预期性能。

8.6.1 基础

原理和思路

思路：通过机器学习的方法获取到那些比较“难”分类的样本数据，让人工再次确认和审核，然后将人工标注得到的数据再次使用有监督学习模型或者半监督学习模型进行训练，逐步提升模型的效果，将人工经验融入机器学习的模型中。

在主动学习中，有三种典型场景。知名度最高的一种场景称为基于池的采样（Pool-based Sampling），它遵循以下五个步骤：

1. 人员（在此过程中称为 Oracle）标注数据集的一小部分，并将标注数据提供给模型。
2. 模型（称为主动学习者）处理这些数据，并以一定的置信度预测未标注数据点的类别。
3. 假设初始预测低于所需精度和置信度，则会使用采样技术确定下一个需要标注的数据子集。
4. 人员标注选定的数据子集并将标注的数据子集发送回模型进行处理。
5. 该过程将继续，直至模型的预测达到所需的置信度和精度水平。

![al](https://cdn.jsdelivr.net/gh/xxzhai123/img/img776dab0021964e148edbb2a6c4b8dbee~tplv-k3u1fbpfcp-zoom-in-crop-mark 1304 0 0 0.awebp.webp)

另一个主动学习场景即基于流的选择采样（Stream-based Selective Sampling）。

主动学习的分类

根据输入数据的方式，主动学习可以分为：

基于流的主动学习，它将未标记的数据一次性全部呈现给一个预测模型，该模型将预测结果（实例的概率值），根据某些评价指标（比如 margin）计算评估实例的价值，随后应用主动学习决定是否应该花费一些预算来收集此数据的类标签，以进行后续的训练；

基于池的主动学习，这个通常是离线、反复的过程。这里向主动学习系统提供了大量未标记的数据，在此过程的每个迭代周期，主动学习系统都会选择一个或者多个未标记数据进行标记并用于随后的模型训练，直到预算用尽或者满足某些停止条件为止。此时，如果预测性能足够，就可以将模型合并到最终系统中，该最终系统为模型提供未标记的数据并进行预测。

根据数据选择的角度，又可以分为具有渐进关系的两类：

一是仅基于独立同分布（IID）数据的不确定性进行主动学习，其中选择标准仅取决于针对每个数据自身信息计算的不确定性值；

二是通过进一步考虑实例相关性来进行主动学习，基于数据相关性的不确定性度量标准，利用一些相似性度量来区分数据之间的差异。

基于不确定性的方法基于信息量的策略由于实用性强，因此被广泛使用。不确定性越大，蕴含的信息量越大，越有训练价值。用已打标的数据子集训练模型，用该模型预测剩余未打标样本，根据预测结果使用不确定性衡量标准找出最不确定的样本，交给打标人员标注，加入训练集训练模型，再用该模型进行数据挑选，反复迭代。代表方法

1. least confident(LC): 关注模型预测时置信度值很大，“可信度”依旧很低的样本。缺点是没关注易混淆的样本。
2. smallest margin（SM）: 关注置信度最大的两个值的差（margin）最小的样本，即易混淆的样本，该方案是针对 LC 的缺点进行的改进。
3. entropy（ENT）: 关注综合信息量最大的样本。

基于委员会查询的方法（Query-By-Committee，QBC）将优化 ML 模型看成是版本空间搜索，QBC 通过压缩版本空间的搜索范围，找到最优秀 ML 模型。相同训练集训练多个同结构的模型，模型投票选出争议样本，将争议样本打标后训练模型，反复迭代。

主动学习 vs. 被动学习

被动学习（passive learning）被认为是从数据集中随机选择（randomly select）数据进行标注。而主动学习选择要标注的样本时，有一些 criteria 进行指导，这就是主动学习和被动学习的区别。不过被动学习似乎叫的不多，一般用 random selection 与主动学习的 criteria 比较就好。

主动学习与监督学习、弱监督学习、半监督学习、无监督学习之间的关系

监督学习（Supervised learning）任务中，数据集的标签都是完整而精确的。无监督学习（Unsupervised learning）任务中，数据集是不含标签的。弱监督学习（Weakly-supervised learning）任务中，数据集的标签分为三种情况：（这三种情况可能同时出现）

- 部分数据有标签，部分数据没有标签。一般有标签的数据占少数，大部分数据没有标签。(Incompelet supervison）
- 数据都有标签，但是标签的粒度不够。例如，在图像语义分割中，细粒度的标签应该是 pixel-level 的，但给出的标签仅仅是 image-level 的，这就是标签的粒度不够。(Inexact supervison)
- 数据都有标签，但是标签有很多错误。(Inaccurate supervison)

8.6.2 问题与解决

类不平衡问题

主动学习无疑是有效的，但最近的一些研究表明，主动学习在应用于存在类不平衡问题时往往会失败：大类中的数据所占比例较大，可能会导致模型的训练和预测偏向一个类。之前的一些研究，试图通过使用不同的技术来解决这一问题。

8.6.3 模型

主动学习的模型分类包括两种

8.6.4 工具

- alipy
- alipy GitHub

8.6.5 Links

- Learning with not Enough Data Part 2: Active Learning | by Lilian Weng@OpenAI
- 主动学习方法实践：让模型变“主动”
- 主动学习 (Active Learning) 介绍
- 主动学习
- [Active Learning] 01 主动学习简介

CHAPTER 9

人工智能最佳实践

9.1 PyTorch 基础

9.1.1 张量

```
torch.stack([x1, x2], dim=-1)  # 最后一维堆叠
torch.FloatTensor(2, 3, 4)
torch.LongTensor
torch.Tensor(2, 3, 4)
torch.zeros
torch.ones(*(3,4)).fill_(1)
torch.rand(*size, )  # uniformly sampled between 0 and 1
torch.randn  # normal distribution with mean 0 and variance 1
torch.randint(low=0, high=2, size=(self.size, 2), dtype=torch.float32)
torch.arange  # from n to m torch.Tensor (input list)
# dim 不能超过 x 的维度
torch.cat([x1, x2], dim=1)
```

张量求导

```
from torch.autograd import Variable
Variable(x)
```

张量属性

```
x = torch.Tensor
x.data  # data 用索引下表访问
```

9.1.2 Basic

```
# 数据类型转换

torch.from_numpy(np_arr1)
x.numpy()
x.cpu().numpy()  # 对显卡上的数据

# 张量操作
x2.add_(x1)  # 原位
torch.matmul(x, W)
torch.mm
torch.bmm  # b,n,m @ b,m,p -> b,n,p
torch.einsum  # matrix multiplications
x.view(2, 3)  # reshape
x.requires_grad_(True)  # 默认的矩阵不求导

y.backward()
x.grad

# Size is same as Shape
x.shape
x.size()
```

9.1.3 拓展学习资料

- Pytorch 中的 masked_fill() 方法

9.2 PyTroch 数据

9.2.1 Tricks

1. 测试集合的 `shuffle` 和 `drop_last` 设置为 `False`

```
test_data_loader = data.DataLoader(
    test_dataset,
    batch_size=128,
    shuffle=False,
    drop_last=False,
    num_workers=conf.workers,  # 加快载入速度
)
```

9.2.2 使用现成的数据

```
from torch.utils.data import Dataset
from torchvision import datasets
from torchvision.transforms import ToTensor

training_data = datasets.FashionMNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor()
)

test_data = datasets.FashionMNIST(
    root="data",
    train=False,
    download=True,
    transform=ToTensor()
)
```

9.2.3 自定义数据

创建一个 Dataset 对象，实现 __getitem__() 和 __len__() 这两个方法 __getitem__() 返回一条数据的内容和标签

```
from torch.utils.data import DataLoader
# training_data 是 Dataset
train_dataloader: DataLoader = DataLoader(training_data, batch_size=64, shuffle=True)
```

多数据集的模型

用 Concatenated Dataset 实现

```
from pytorch_lightning import LightningModule
from torch.utils.data import ConcatDataset

class LitModel(LightningModule):
    def train_dataloader(self):
            concat_dataset = ConcatDataset(datasets.ImageFolder(traindir_A), datasets.
↪ImageFolder(traindir_B))

            loader = DataLoader(
                concat_dataset,
                batch_size=args.batch_size,
                shuffle=True,
                num_workers=args.workers,
                pin_memory=True  # 载入内存
            )
            return loader
    ...
```

使用数据加载器来训练模型

```
for epoch in range(100):
    for i, data in enumerate(train_loader):
        inputs, labels = data
```

数据分割

方法一

```
train_size = int(0.8 * len(full_dataset))
test_size = len(full_dataset) - train_size
train_dataset, test_dataset = torch.utils.data.random_split(full_dataset, [train_size,
↪ test_size])
```

方法二

```
dataset = MyCustomDataset(my_path)
batch_size = 16
validation_split = .2
shuffle_dataset = True
random_seed= 42

# Creating data indices for training and validation splits:
dataset_size = len(dataset)
indices = list(range(dataset_size))
split = int(np.floor(validation_split * dataset_size))
if shuffle_dataset :
    np.random.seed(random_seed)
    np.random.shuffle(indices)
train_indices, val_indices = indices[split:], indices[:split]

# Creating PT data samplers and loaders:
train_sampler = SubsetRandomSampler(train_indices)
valid_sampler = SubsetRandomSampler(val_indices)

train_loader = torch.utils.data.DataLoader(dataset, batch_size=batch_size,
                                           sampler=train_sampler)
validation_loader = torch.utils.data.DataLoader(dataset, batch_size=batch_size,
                                                sampler=valid_sampler)

# Usage Example:
num_epochs = 10
for epoch in range(num_epochs):
    # Train:
    for batch_index, (faces, labels) in enumerate(train_loader):
        # ...
```

torch 带的数据处理函数

```
class torch.utils.data.Dataset: 一个抽象类，
↪所有其他类的数据集类都应该是它的子类。而且其子类必须重载两个重要的函数：len(提供数据集的大小）、get
class torch.utils.data.TensorDataset: 封装成 tensor
↪的数据集，每一个样本都通过索引张量来获得。
class torch.utils.data.ConcatDataset: 连接不同的数据集以构成更大的新数据集。
class torch.utils.data.Subset(dataset, indices): 获取指定一个索引序列对应的子数据集。
class torch.utils.data.DataLoader(dataset, batch_size=1, shuffle=False, sampler=None,
↪batch_sampler=None, num_workers=0, collate_fn=<function default_collate>, pin_
↪memory=False, drop_last=False, timeout=0, worker_init_fn=None):
↪数据加载器。组合了一个数据集和采样器，并提供关于数据的迭代器。
torch.utils.data.random_split(dataset, lengths):
↪按照给定的长度将数据集划分成没有重叠的新数据集组合。
class torch.utils.data.Sampler(data_
↪source):所有采样的器的基类。每个采样器子类都需要提供 iter 方-法以方便迭代器进行索引
↪和一个 len方法 以方便返回迭代器的长度。
class torch.utils.data.SequentialSampler(data_
↪source):顺序采样样本，始终按照同一个顺序。
class torch.utils.data.RandomSampler(data_source):无放回地随机采样样本元素。
class torch.utils.data.
↪SubsetRandomSampler(indices): 无放回地按照给定的索引列表采样样本元素。
class torch.utils.data.WeightedRandomSampler(weights, num_samples, replacement=True):
↪按照给定的概率来采样样本。
class torch.utils.data.BatchSampler(sampler, batch_size, drop_last):
↪在一个batch中封装一个其他的采样器。
class torch.utils.data.distributed.DistributedSampler(dataset, num_replicas=None,
↪rank=None):采样器可以约束数据加载进数据集的子集。
```

9.2.4 数据保存与加载

numpy

```
# 保存字典对象
np.save(path, experimental_dict)
np.load(pwd + 'experimental_dataset_dict.npy', allow_pickle = True)
```

9.3 PyTorch 配置与日志

```
torch.__version__
```

9.3.1 环境变量

```
DATASET_PATH = os.environ.get("PATH_DATASETS", "data/")
CHECKPOINT_PATH = os.environ.get("PATH_CHECKPOINT", "saved_models/Activation_
↪Functions/")

os.makedirs(CHECKPOINT_PATH, exist_ok=True)
```

jupyter notebook

```
% matplotlib inline
```

9.3.2 随机数初始化

```
# Function for setting the seed
def set_seed(seed):
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    if torch.cuda.is_available():  # GPU operation have separate seed
        torch.cuda.manual_seed(seed)
        torch.cuda.manual_seed_all(seed)

set_seed(42)
# GPU 的随机锁定
torch.backends.cudnn.determinstic = True
torch.backends.cudnn.benchmark = False
```

9.3.3 设备

```
os.environ['CUDA_VISIBLE_DEVICES'] = '0'
device = torch.device("cpu") if not torch.cuda.is_available() else torch.device(
↪"cuda:0")
```

9.3.4 命令行参数

```
# 命令主题说明
parser = argparse.ArgumentParser(usage = 'peptide-HLA-I binding prediction')
parser.add_argument('--peptide_file', type = str, help = 'the path of the .fasta file␣
↪contains peptides')
parser.add_argument('--threshold', type = float, default = 0.5, help = 'the threshold␣
↪to define predicted binder, float from 0 - 1, the recommended value is 0.5')
parser.add_argument('--cut_peptide', type = bool, default = False, help = 'Whether to␣
↪split peptides larger than cut_length?')
parser.add_argument('--cut_length', type = int, default = 9, help = 'if there is a␣
↪peptide sequence length > 15, we will segment the peptide according the length you␣
↪choose, from 8 - 15')

# 此时可以查看默认设置的参数
args = parser.parse_args(args = [])
args
```

9.4 日志

9.4.1 lightning

rank_zero

```
# rank_zero_warn, rank_zero_debug, rank_zero_deprecation
from pytorch_lightning.utilities import rank_zero_info
rank_zero_info(
    'info 1'
    'info 2'
)
```

9.4.2 日志简单过滤

```
from log import warning

warnings.filterwarnings("ignore")
```

9.4.3 初始化日志

```
import logging

class Logger(object):
    level_relations = {
        'debug':logging.DEBUG,
        'info':logging.INFO,
        'warning':logging.WARNING,
        'error':logging.ERROR,
        'crit':logging.CRITICAL
    }  # 日志级别关系映射
```

(下页继续)

(续上页)

```
    def __init__(self,filename,level='info',fmt='%(asctime)s : %(message)s'):
        self.logger = logging.getLogger(filename)
        # 设置日志格式
        format_str = logging.Formatter(fmt)
        # 设置日志级别
        self.logger.setLevel(self.level_relations.get(level))
        # 往屏幕上输出
        sh = logging.FileHandler(filename, mode = 'w')
        # 设置屏幕上显示的格式
        sh.setFormatter(format_str)
        # 把对象加到logger里
        self.logger.addHandler(sh)

# 使用
from utils import Logger

## 紧急
    log = Logger('./error.log')
    log.logger.critical('The threshold invalid, please check whether it ranges from 0-
↪1.')
    sys.exit(1)
```

9.5 可视化

9.5.1 tensorboard

Tensorboard 原本是 TensorFlow 的可视化工具，而目前在 TensorboardX 的加持下，其他深度学习计算框架也可使用 TensorBoard 工具进行可视化操作了。

```
pip install tensorboardX
```

```
from torch.utils.tensorboard import SummaryWriter
```

- PyTorch 建模可视化工具 TensorBoard 的安装与使用

9.5.2 Others

```
from matplotlib.colors import to_rgba
import matplotlib.plot as plt

@torch.no_grad()   # Decorator, same effect as "with torch.no_grad(): ..." over the␣
↪whole function.
def visualize_classification(model, data, label):
    # Before plotting, fetch data into numpy from gpu onto cpu
    if isinstance(data, torch.Tensor):
        data = data.cpu().numpy()
    if isinstance(label, torch.Tensor):
        label = label.cpu().numpy()
    data_0 = data[label == 0]
    data_1 = data[label == 1]
```

(下页继续)

(续上页)

```
    plt.figure(figsize=(4, 4))
    plt.scatter(data_0[:, 0], data_0[:, 1], edgecolor="#333", label="Class 0")
    plt.scatter(data_1[:, 0], data_1[:, 1], edgecolor="#333", label="Class 1")
    plt.title("Dataset samples")
    plt.ylabel(r"$x_2$")
    plt.xlabel(r"$x_1$")
    plt.legend()

    model.to(device)
    c0 = torch.Tensor(to_rgba("C0")).to(device)
    c1 = torch.Tensor(to_rgba("C1")).to(device)
    x1 = torch.arange(-0.5, 1.5, step=0.01, device=device)
    x2 = torch.arange(-0.5, 1.5, step=0.01, device=device)
    xx1, xx2 = torch.meshgrid(x1, x2)  # Meshgrid function as in numpy
    model_inputs = torch.stack([xx1, xx2], dim=-1)
    preds = model(model_inputs)
    preds = torch.sigmoid(preds)
    # Specifying "None" in a dimension creates a new one
    output_image = (1 - preds) * c0[None, None] + preds * c1[None, None]
    output_image = (
        output_image.cpu().numpy()
    )  # Convert to numpy array. This only works for tensors on CPU, hence first push␣
↪to CPU
    plt.imshow(output_image, origin="lower", extent=(-0.5, 1.5, -0.5, 1.5))
    plt.grid(False)
```

9.6 Torchtext

先去 Github 了解个大概

- Github

这是官方文档 | 手册 这是官网的一个教程

```
pip install torchtext
pip install spacy
python -m spacy download en_core_web_sm
pip install sacremoses
```

```

```

9.6.1 torchtext 的组件

Field: 主要包含以下数据预处理的配置信息，比如指定分词方法，是否转成小写，起始字符，结束字符，补全字符以及词典等等

Dataset: 继承自 pytorch 的 Dataset，用于加载数据，提供了 TabularDataset 可以指点路径，格式，Field 信息就可以方便的完成数据加载。同时 torchtext 还提供预先构建的常用数据集的 Dataset 对象，可以直接加载使用，splits 方法可以同时加载训练集，验证集和测试集。

Iterator: 主要是数据输出的模型的迭代器，可以支持 batch 定制

Field

Field 包含一写文本处理的通用参数的设置，同时还包含一个词典对象，可以把文本数据表示成数字类型，进而可以把文本表示成需要的 tensor 类型

9.6.2 应用

- 构造词表

Field 包含一写文本处理的通用参数的设置，同时还包含一个词典对象，可以把文本数据表示成数字类型，进而可以把文本表示成需要的 tensor 类型

9.7 Pytorch Lightning

9.7.1 Debugging

有限步骤

```
# runs 1 train, val, test batch and program ends
trainer = Trainer(fast_dev_run=True)

# runs 7 train, val, test batches and program ends
trainer = Trainer(fast_dev_run=7)
```

小批量

```
# use only 10% of training data and 1% of val data
trainer = Trainer(limit_train_batches=0.1, limit_val_batches=0.01)

# use 10 batches of train and 5 batches of val
trainer = Trainer(limit_train_batches=10, limit_val_batches=5)
```

理智的验证步骤

```
# DEFAULT
trainer = Trainer(num_sanity_val_steps=2)
```

小数据过拟合

```
# use only 1% of training data (and turn off validation)
trainer = Trainer(overfit_batches=0.01)

# similar, but with a fixed 10 batches
trainer = Trainer(overfit_batches=10)
```

9.7.2 Eearly Stopping

```
from pytorch_lightning.callbacks.early_stopping import EarlyStopping

class LitModel(LightningModule):
    def validation_step(self, batch, batch_idx):
        loss = ...
        self.log("val_loss", loss)

model = LitModel()
early_stopping = EarlyStopping(monitor="val_acc", min_delta=0.1, patience=3,␣
↪verbose=False, mode="max")
trainer = Trainer(callbacks=[early_stopping])
trainer.fit(model)
```

9.7.3 Hyperparameters

ArgumentParser

```
from argparse import ArgumentParser

parser = ArgumentParser()
parser.add_argument("--layer_1_dim", type=int, default=128)
args = parser.parse_args()
```

9.8 Metrics

9.8.1 Top-k 实现代码

- 推荐阅读 | 月来客栈 | *Top-K* 准确率介绍与实现

Numpy 实现

```
def get_top_k_result(logits, k=3, sorted=True):
    indices = np.argsort(logits, axis=-1)[:, -k:]  # 取概率最大的前K个所对应的预测标签
    if sorted:  # np.argsort 默认返回的顺序是从小到大，sorted=True 可以返回从大到小
        tmp = []
        for item in indices:
            tmp.append(item[::-1])
        indices = np.array(tmp)
    values = []
    for idx, item in zip(indices, logits):  # 取所有预测值所对应的概率值
        p = item.reshape(1, -1)[:, idx].reshape(-1)
        values.append(p)
    values = np.array(values)
    return values, indices

logits = np.array([[0.1, 0.3, 0.2, 0.4],
                   [0.5, 0.01, 0.9, 0.4]])
y = np.array([3, 0])
```

(下页继续)

(续上页)

```
print(get_top_k_result(logits))

>>> (array([[0.4, 0.3, 0.2],
       [0.9, 0.5, 0.4]]), array([[3, 1, 2],
       [2, 0, 3]], dtype=int64))

def calculate_top_k_accuracy(logits, targets, k=2):
    values, indices = get_top_k_result(logits, k=k, sorted=False)
    y = np.reshape(targets, [-1, 1])
    correct = (y == indices) * 1.  # 对比预测的K个值中是否包含有正确标签中的结果
    top_k_accuracy = np.mean(correct) * k  # 计算最后的准确率
    return top_k_accuracy

print(calculate_top_k_accuracy(logits, y, k=2)) # 1.0
print(calculate_top_k_accuracy(logits, y, k=1)) # 0.5
```

Tensorflow

```
import tensorflow as tf

logits = tf.constant([[0.1, 0.3, 0.2, 0.4],
                      [0.5, 0.01, 0.9, 0.4]], shape=[2, 4], dtype=tf.float32)
y = tf.constant([3, 0], tf.int32)

def calculate_top_k_accuracy(logits, targets, k=2):
    values, indices = tf.math.top_k(logits, k=k, sorted=True)
    y = tf.reshape(targets, [-1, 1])
    correct = tf.cast(tf.equal(y, indices), tf.float32)
    top_k_accuracy = tf.reduce_mean(correct) * k
    return top_k_accuracy

sess = tf.Session()
print(sess.run(calculate_top_k_accuracy(logits, y, k=2)))# 1.0
print(sess.run(calculate_top_k_accuracy(logits, y, k=1)))# 0.5
```

Pytorch 中的实现

```

import torch
logits = torch.tensor([[0.1, 0.3, 0.2, 0.4],
                       [0.5, 0.01, 0.9, 0.4]])
y = torch.tensor([3, 0])
def calculate_top_k_accuracy(logits, targets, k=2):
    values, indices = torch.topk(logits, k=k, sorted=True)
    y = torch.reshape(targets, [-1, 1])
    correct = (y == indices) * 1.  # 对比预测的K个值中是否包含有正确标签中的结果
    top_k_accuracy = torch.mean(correct) * k  # 计算最后的准确率
    return top_k_accuracy

print(calculate_top_k_accuracy(logits, y, k=2).item())# 1.0
print(calculate_top_k_accuracy(logits, y, k=1).item())# 0.5
```

9.8.2 Links

- Top-K 准确率介绍与实现

CHAPTER 10

论文

10.1 论文写作

10.1.1 如何写综述

综述

天才第一步，始于纸尿裤，而科研第一步，则是来自于写综述。写综述是科研的基本功，它可以是让自己打入某个研究领域的第一道法门，也可以是在自己有了一定的研究基础之后，对所处领域进行宏观的概括，指点江山。文献综述不是简单的罗列现有的工作和堆砌前人的学术成果。一篇好的文献综述，应该至少能够让读者：

1. 很清晰的了解该问题发展的脉络，主流的方法都有哪些；
2. 掌握目前的研究现状，已有工作做到什么程度了、达到什么样的指标；
3. 还有哪些未解决的问题和研究空白；

综述写作的思路

综述写作思路也有各式花样，但又一脉相承。下面简单介绍三个综述写作的思路

专注当前，粗中取精

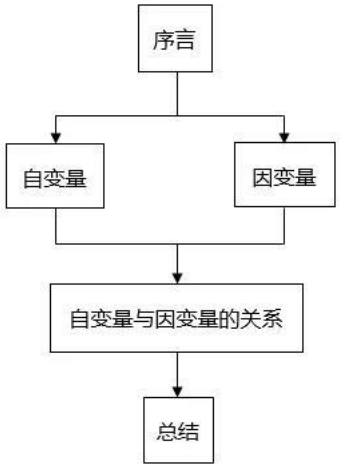
如果是刚要打入一个新的研究领域，写综述的目的主要是为了给自己的课题作铺垫，定义并限定自己要研究的问题，并把它置于历史脉络中来考察其价值，避免与现有研究重复。日常读文献、做笔记时都要考虑这几点，以便把读过的文献材料组织起来形成框架。

把材料组织好，其实一篇综述也差不多搭建起来了。美国内布拉斯加林肯大学的教育心理学教授约翰 W. 克雷斯威尔（John W. Creswell）曾经提出的五步文献综述法，在各个领域都可借鉴。说是“五步”，其实也可以看作材料组织和写作的思路结构。他认为，文献综述应由五部分组成：序言、主题 1（关于自变量）、主题 2（关于因变量

1. 序言告诉读者文献综述所涉及的几个部分，这一段是关于章节构成的陈述，也就相当于文献综述的总述。
2. **综述主题 1** 即关于“自变量”的文献。比如要讨论某药物对某蛋白表达的调控作用，那么该药物就是自变量，是由研究者主动操纵、能引起因变量发生变化的条件。你的课题中也许会有好几个自变量，那么再做个亚层分别论述；但总有那么一个比较关键，当然要重点阐述。
3. 综述主题 2 是关于“因变量”的文献。和自变量的情况相似，一个药物也许会引起多种生化反应，但仍然是落水三千取一瓢饮。

当然自变量和因变量的顺序不是固定的，看自己方便。只是要将它们分别阐述，能囊括更全面的信息，展示相关研究领域的大格局，便于把自己的课题置于其中，提炼其研究意义和价值。

1. 综述主题 3 要把自变量和因变量的关系建立起来，这是非常关键的一部分，是找到自己研究方向突破点的一部分。这里要包含与主题最为接近的研究，梳理两者千丝万绪的关系中哪些方面已经研究成熟，哪些方面稍薄弱但充满潜力，是提出假设的着力点。



献，那就要尽可能找到与主题相近的部分，或者综述在更广泛的层面上提及意不要回避那些不支持你观点的研究，不论多小的研究，提一下还是有必要其实更有可能漏掉一种潜在的问题解释方案，若将来你的实验未能证明自己持你的证据。

最重要的研究，抓住主题，指出为什么我们要对这个主题做更多的研究。其述进行总结，更重要的是强调你研究课题的基石（前人的肩膀），也就是你的

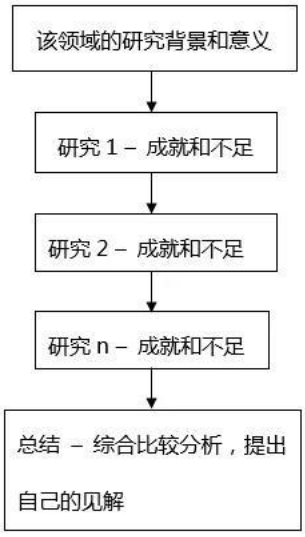
总揽大局，指点迷津

如果已经在 ** 本领域征战多年，眼界已经大了很多，尽管也是为了给自己的下一个研究作铺垫，但又不能仅限于此。** 这类综述就是那种期刊编辑特约、由本领域神级大牛所作，将来是有可能要进教科书的，当然也成为大家膜拜学习的对象。

** 这类综述，上述的“五步法”也是适用的。** 但由于大牛在本领域可能不止研究过 1 个问题，有可能是一种干预手段对应一种机制、却在不同的疾病模型中验证；或者对一种疾病提出了不同的病理机制假说，并各自找到对应的干预手段来验证，等等。

也就是说，他自己经历过的“自变量”和“因变量”下面的亚层更加复杂，更别说再综合他参考过的别人的研究，所以他的思考也不是决定要写综述才开始的，而是有岁月的积累，于是最后能发现的问题也更多，提出的设想、研究前景也更丰富。这时候有一种平行结构的思路可以借鉴。

当然如果你还不是大牛，只是遇到了一个比较复杂的课题，那么卯足劲做足功课，也可以尝试这种结构，毕竟不想当将军的士兵不是好士兵。但精华在于 Discussion 部分，关键还是提出现有的问题、分析可能的原因，并为以后的研究者提供一些研究方向的参考。



按时间顺序组织材料，疏理某个研究专题的历史发展脉络，着重展示它的“演论某种研究方法、工具的发展，或具有历史意义的课题，最后 Discussion 部分也的各种可能性。这类综述有时候被称为“描述性综述”，这名字其实有点黑，因，太过于注重史料的编排，有时候会忽略这点。“述”是指批判性思维（critical中，都是贯穿材料组织和写作始终的。这种编年法的综述就可以在对每一个研究斤，指出它的问题，同时引出解决它问题的下一个研究。总结部分可以对各个研也不要忘了强调历史上最有亮点的研究——最新的研究也许并不是最好的，那呢？也许也会为下一步发展提供一个旁路，打开新的研究方向。

几个误区

1. 注意几个误区，误区一：文献综述就是读书笔记，按照时间顺序来写就行，正解：文献综述，其实要按照逻辑层次顺序来写。
2. 误区二：文献综述就是总结概括一下前人写过的文献，正解：文献综述是对前人文章的论述，这里的论述就包括介绍、总结指出不足。
3. 误区三：文献综述随便选几个文献来写就可以了，正解：文献综述要选择和你文章最相关，最核心的文献来写。
4. 误区四：文献综述的主要目的就是向导师展示我已经读了很多文献，正解：是指出前人研究所存在的 research gap。

Links

- 如何有效率地写综述?
- 我整理了近 300 篇文献，写了一篇 SCI 文献综述 | 全面 | by 后山

CHAPTER 11

Books

11.1 人工智能药物设计

11.1.1 About

Author

Alexander Heifetz

Press

Humana Press

This Humana imprint is published by the registered company Springer Science+Business Media, LLC, part of Springer Nature.

11.1.2 Preface

The design of novel therapeutics is an inventive process which involves assimilation and analysis of available experimental data in conjunction with traditional and novel molecular modeling techniques. It can be an extremely complex, long, and expensive process. The application of artificial intelligence (AI) and machine learning (ML) approaches promises to revolutionize the design-make-test-analyze (DMTA) cycle, accelerating the drug design process and therefore reducing cost. Over the past few years, the field of AI/ML has moved from largely theoretical studies to real-world applications. Much of that explosive growth has been enabled by the availability of graphical processing units (GPUs) and advances in AI/ML algorithms, such as deep learning (DL). The use of neural networks in deep learning algorithms enables computers to imitate human intelligence by learning from data. The application of these approaches can be exploited across multiple aspects of drug design.

This book provides an overview of the state of the art in the development and applica- tion of AI/ML/DL methods in drug design. Topics covered include: how the application of these methods can be implemented to accelerate and revolutionise traditional drug design approaches such as structure- and ligand-based, augmented and multi-objective de

novo drug design, SAR and big data analysis, prediction of binding/activity, ADMET, pharma- cokinetics and drug-target residence time, precision medicine and favorable chemical syn- thetic routes prediction. Also included is discussion of how broadly these approaches are applied and where they maximally impact productivity both today and, potentially, in the near future. The review of these topics will allow a diverse audience, including computa- tional and medicinal chemists, pharmacologists and drug designers, to navigate through the existing techniques and challenges and gain an enhanced appreciation of the new directions under development.

Oxfordshire, UK

Alexander Heifetz

11.1.3 Applications of Artificial Intelligence in Drug Design: Opportunities and Challenges

1 Introduction: What Challenges Does Drug Design Face?

- cost has increased dramatically
- Around 80% of failures in clinical trials occur
  - insufficient efficacy or safety in patients.

Overview of the phases of drug design

Phase	Goal	Applications of AI
Target discovery and validation	Discover target biomolecules or cell-based/tissue-based assays which can be used to test compounds for potential therapeutic efficacy	Target discovery using ontologies, knowledge graphs to combine data types (genetic, clinical, etc...) [5]; ML for diagnostics & patient stratification [6]
Library design and hit discovery	Design, make, and test compounds for validated activity against the target which have appropriate properties for further development (ADMET, etc.)	Virtual screening, de novo molecule generation, computer-aided synthesis prediction, target prediction of phenotypic screen hits [7]
Hit-to-lead and lead optimization	Select the most promising hits; optimize chosen leads for potency against the intended target while minimizing off-target bioactivity and ensuring an appropriate ADMET profile	De novo molecule generation (requires multi-objective optimization), computer-aided synthesis prediction, active learning [8], ADMET modeling [9]
Preclinical and clinical studies	Choose dose, formulation, and method of administration; test for efficacy and safety in model organisms and then in patients	Estimation of treatment effects on animal models or patients using causal inference [10]

Many datasets have been gathered on the properties, reactions, and interactions of chemical compounds; however, this data is disproportionately focused on a small range of well-studied endpoints.

CHAPTER 12

Indices and tables

- genindex
- modindex
- search