Table of Contents

Introduction 1.1

Python基础 1.2

- 文件读写 1.2.1
- OS模块 1.2.2
- 多进程和多线程 1.2.3
- 异常处理 1.2.4
- 发送邮件 1.2.5
- 面向对象编程 1.2.6

数据结构与算法 1.3

- 排序算法 1.3.1
- 递归算法 1.3.2
- Proximal Gradient Decent 1.3.3

统计学习 2.1

- 矩阵知识 2.1.1

数据库 2.2

- MySQL 2.2.1
- Mongodb 2.2.2

Linux基础 3.1

- 服务器配置 3.1.1
- 权限管理 3.1.2
- 文件操作 3.1.3
- 定时任务 3.1.4

其他软件介绍 3.2

- Gitbook使用 3.2.1
- git使用 3.2.2
- Pandoc使用 3.2.3

数据科学使用手册

> 写写我在数据科学道路上的一点点耕耘。主要为个人使用，所以很多引用没写清楚或者写的不规范。欢迎和我讨论一起学习.
> Email: chenliujun0556@163.com.

- Python基础
  - 文件读写
  - OS模块
  - 多进程和多线程
  - 异常处理
  - 发送邮件
  - 面向对象编程
- 数据结构与算法
  - 排序算法
  - 递归算法
  - Proximal Gradient Decent

- 统计学习
  - 矩阵知识
- 数据库
  - MySQL
  - Mongodb

- Linux基础
  - 服务器配置
  - 权限管理
  - 文件操作
  - 定时任务
- 其他软件介绍
  - Gitbook使用
  - git使用
  - Pandoc使用

Python基础

python文件处理

open 方法

Python open() 方法用于打开一个文件，并返回文件对象，在对文件进行处理过程都需要使用到这个函数，如果该文件无法被打开，会抛出 OSError。

注意：使用 open() 方法一定要保证关闭文件对象，即调用 close() 方法。

open() 函数常用形式是接收两个参数：file, mode, encoding。

```
open(file, mode='r',encoding=None)
```

mode 参数常用的有

- r 只读
- a+ 打开一个文件用于读写。如果该文件已存在，文件指针将会放在文件的结尾。文件打开时会是追加模式。如果该文件不存在，创建新文件用于读写。
- w+ 打开一个文件用于读写。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。如果该文件不存在，创建新文件。

file 对象

- file.close() 关闭文件
- file.seek() `0代表从头开始，1代表当前位置，2代表文件最末尾位置`
- file.readlines()
- file.writelines()

OS模块

目录操作-增删改查

- os.listdir() 列出指定目录下所有的
- os.mkdirs() 创建一个文件夹
- os.rmdir() 删除一个文件夹，但是必须是空文件夹

路径信息

- os.path.abspath(path) 显示当前绝对路径
- os.path.dirname(path) 返回该路径的父目录

```
os.path.dirname(os.path.abspath('hello.py'))
```

- os.path.isfile(path) 是文件则返回True,和os.path.isdir(path)相对应
- os.path.join(path,name) #连接目录与文件名或目录 结果为path/name
- os.getcwd() 显示当前python的工作目录

重命名

```
os.rename(old_name, new_name)
```

Python 使用 smtp 发送邮件

内容很简单，只是知道怎么用就好了

首先导入要用的模块

```
from smtplib import SMTP_SSL
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
from email.mime.application import MIMEApplication
from email.header import Header
```

然后输入收发信任的信息

```
host_server='smtp.163.com'
sender='chenliujun0556@163.com'
pwd='******'
receivers_list=[['sven163','chenliujun0556@163.com'],['svenfudan','191106
```

然后对收件人信息做个循环就好了

```
for receiver in receivers_list:
    mail_title='python办公自动化的邮件'

    mail_content='''Dear {}:
    你好，这是一个测试邮件，
    <p><a href="http://www.baidu.com">百度</a></p>
    <p>图片演示：</p>
    '''.format(receiver[0])
    msg=MIMEMultipart()
    msg['Subject']=Header(mail_title,'utf-8')
    msg['From']=sender
    msg['To']=Header(receiver[1])
    #msg.attach(MIMEText(mail_content,'plain','utf-8'))
    msg.attach(MIMEText(mail_content,'html','utf-8'))
    ###插入附件
    attachment=MIMEApplication(open('D:/test.xlsx','rb').read())
    attachment.add_header('content-Disposition','attachment',filename='te
    msg.attach(attachment)
    try:
        smtp=SMTP_SSL(host_server)
        smtp.ehlo(host_server)
        smtp.login(sender,pwd)
        smtp.sendmail(sender,receiver[1],msg.as_string())
        smtp.quit()
    except:
        print('发送给{}的邮件失败了'.format(receiver[1]))
```

Python的类

创建新类

使用 class 语句来创建一个新类，class 之后为类的名称并以冒号结尾,一般遵循习惯类的名称首字母大写. 以下创建一个简单的类

```
class Student:
    def __init__(self,name,ID):
        self.name=name
        self.ID=ID
    def get_ID(self):
        return self.ID
```

数据结构与算法

排序算法

这里介绍插入，选择，冒泡，归并，快速排序这5种常见的排序算法。当然还有其他算法，这里不加以介绍了。这里的代码不是最优的，没有考虑一些空间的占用问题（特别是对于归并排序和快速排序，其实是不用占用额外空间的），不过比较好理解。

插入排序

插入排序是一种简单直观的排序算法。它的工作原理是通过构建有序序列，对于未排序数据，在已排序序列中从后向前扫描，找到相应位置并插入。

从第一个元素，如果第1个元素小于第0个元素，就把第1个元素和第0个元素交换（前2个位置已经排好序）

到第二个元素，如果第2个元素小于第1个元素，就把第2个元素和第1个元素交换，然后比较一直交换到前3个元素排好位置

以此类推....

插入排序的平均时间复杂度是

```
def insert_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i
        while j >= 1 and key <= arr[j - 1]:
            arr[j] = arr[j - 1]
            j = j - 1
        arr[j] = key
```

选择排序

选择排序是一种简单直观的排序算法。

- 首先找出整个序列中最小的元素，放在第一位
- 然后从未排序的序列中找出最小的，放到已排序序列的末尾

```
def selection_sort(arr):
    for i in range(0,len(arr)):
        key_index=i
        for j in range(i+1,len(arr)):
            if arr[j]<arr[key_index]:
                key_index=j
        arr[i],arr[key_index]=arr[key_index],arr[i]
```

冒泡排序

冒泡排序（Bubble Sort）也是一种简单直观的排序算法。

它重复地走访过要排序的数列，一次比较两个元素，如果他们的顺序错误就把他们交换过来。

走访数列的工作是重复地进行直到没有再需要交换，也就是说该数列已经排序完成。

这个算法的名字由来是因为越小的元素会经由交换慢慢"浮"到数列的顶端。

```
def bubble_sort(arr):
    print('开始进行冒泡排序')
    for j in range(len(arr)-1,1,-1):
        for i in range(0,j):
            if arr[i]>arr[i+1]:
                arr[i],arr[i+1]=arr[i+1],arr[i]
```

归并排序

归并排序是递归算法在排序上的应用。或者说是 Divide and Conquer 的一个非常典型的应用。

- 递归地把当前序列平均分割成两半
- 在保持元素顺序的同时将上一步得到的子序列集成到一起（归并）

```
def merge(l_arr,u_arr):
    new_arr=[]
    while len(l_arr)>0 and len(u_arr)>0:
        if l_arr[0]>u_arr[0]:
            new_arr.append(u_arr.pop(0))
        else:
            new_arr.append(l_arr.pop(0))
    if len(l_arr)>0:
        new_arr.extend(l_arr)
    if len(u_arr)>0:
        new_arr.extend(u_arr)
    return (new_arr)

def merge_sort(arr):
    if len(arr)<=1:
        return (arr)
    if len(arr)>=2:
        m=int(len(arr)/2)
        l_arr=merge_sort(arr[:m].copy())
        u_arr=merge_sort(arr[m:].copy())
        return (merge(l_arr,u_arr))
```

快速排序

快速排序使用分治法（Divide and conquer）策略来把一个序列（list）分为较小和较大的2个子序列，然后递归地排序两个子序列。

- 挑选基准值：从数列中挑出一个元素，称为"基准"（pivot）
- 重新排序数列，所有比基准值小的元素摆放在基准前面，所有比基准值大的元素摆在基准后面（与基准值相等的数可以到任何一边）。在这个分割结束之后，对基准值的排序就已经完成;
- 递归排序子序列：递归地将小于基准值元素的子序列和大于基准值元素的子序列排序。

递归到最底部的判断条件是数列的大小是零或一，此时该数列显然已经有序。

```
def quick_sort(arr):
    if len(arr)<=1:
        return (arr)
    else:
        mid = arr[int(len(arr)/2)]
        arr.remove(mid)
        l_array,u_array=[],[]
        for item in arr:
            if item>=mid:
                u_array.append(item)
            else:
                l_array.append(item)
    return quick_sort(l_array)+[mid]+quick_sort(u_array)
```

Proximal Gradient

一般的非线性优化问题主要是通过Gradient Descent方法来解决的。但是这必须要求目标函数是可导的，但是实际上，我们经常会碰到一些目标函数不可导的优化问题（如LASSO,分位数回归等）。Proximal Gradient (以下简称PG) 算法主要是为了解决一类不可导无约束优化问题（大部分的有约束优化问题也可以通过 Lagriange 方法转化为无约束优化问题。）

我们考虑如下的优化问题：

$$\min_x F(x) = f(x) + g(x)$$

这里我们要求$f(x)$是可导的，凸的，并且是 β-Smooth的，但是对于$g(x)$我们只要求 $g(x)$是凸的（其实很多的非凸问题也可以用Proximal Gradient Descent 来解决，不过理论证明会相对比较复杂，所以这里我们假设函数都是凸函数）。

这种问题有很多，例如

- 投影问题 ，可以写做

$$\min_x f(x) + \delta_C(x)$$

  其中，如果 $x \in C, \delta_C(x) = 0$，否则 $\delta_C(x) = \infty$. 这里C是我们需要投影的空间。

- Lasso 问题

$$\min_x \frac{1}{2}||y - Ax||^2 + \lambda||x||_1$$

Proximal Gradient 算法

我们构建 $F(x)$的一个upper bound.

我们定义

$$m(x) = f(x_t) + < \nabla f(x_t), x - x_t > + \frac{\beta}{2}||x - x_t||^2 + g(x).$$

由于 $f(x)$ 是 β smooth的，所以 $m(x) \geq f(x) + g(x)$.

很容易验证（配方），

$$m(x) = f(x_t) - \frac{1}{2\beta}||\nabla f(x_t)||^2 + \frac{1}{2\beta}||x - (x_t - \frac{1}{\beta}\nabla f(x_t))||^2 + g(x)$$

我们定义

$$x_{t+1} = \arg\min_x m(x),$$

则 $F(x_{t+1}) \leq m(x_{t+1}) \leq m(x_t) = f(x_t) + g(x_t) = F(x_t)$.

由于我们是对 x 求最小($m(x)$的前两项和x无关)，所以

$$x_{t+1} = \arg\min_x \frac{1}{2\beta}||x - (x_t - \frac{1}{\beta}\nabla f(x_t))||^2 + g(x).$$

我们定义

$$Prox_g(t) = \arg\min_x \{\frac{1}{2}||x-t||^2 + g(x)\}.$$

则

$$x_{t+1} = Prox_{\frac{1}{\beta}g}(x_t - \frac{1}{\beta}\nabla f(x_t)).$$

这里 $\frac{1}{\beta}$实际上是迭代过程中的步长, 记为 t_k.(如果 $f(x)$ 是一个β smooth 的函数，则步长就取 $1/\beta$. 但是实际上，我们有时候很难求这个 β， 或者 $f(x)$也可能不是 β smooth的，因此我们考虑更一般的步长 t_k)。

完整的 Proximal Gradient 算法叙述如下.

Initialization: 选择 x_0

General Step 对于$t = 0, 1, 2, \cdots$, 执行

- 选择步长 t_k
- $$x_{t+1} = Prox_{t_k g}(x_t - t_k \nabla f(x_t)).$$

Proximal 算子的计算

优化算法

这里，我们主要关注一些凸优化的理论。

索引

索引可以帮助快速的查找数据，提高查询的效率。

索引建立方法, 如果要建立的索引字段是字符串，最好写上字符串的长度。以下是在test表中对id字段建立索引。

```
create index 索引名称 on 表名(字段名(长度))
```

删除索引

```
drop index 索引名称 on 表名
```

查看索引

```
show index from 表名
```

主键，外键其实都是一种索引

常用的列，并且数据量很大，才会考虑使用索引。索引太多容易影响更新和插入的速度。

Mongodb

这里给出在一个新的Ubuntu服务器上安装我常用的软件及其配置的方法。

用户管理

添加用户

```
useradd -s /bin/bash -m -G sudo sven
```

然后使用以下语句设置密码

```
passwd sven
```

删除用户

```
userdel -r sven
```

安装R及Rstudio Server

安装R

用 vim 打开/etc/apt/sources.list, 加入

```
deb https://cloud.r-project.org/bin/linux/ubuntu xenial-cran35/
```

然后

```
apt-get update
apt-get install r-base-core
```

安装Rstudio Server

```
sudo apt-get install gdebi-core
wget https://download2.rstudio.org/server/xenial/amd64/rstudio-server-1.3
sudo gdebi rstudio-server-1.3.959-amd64.deb
```

去服务器用户组配置规则8787

安装配置python环境

Anaconda的安装

下载

```
wget https://mirrors.tuna.tsinghua.edu.cn/anaconda/archive/Anaconda3-4.0.
```

找到对应的文件夹

```
bash Anaconda3-4.0.0-Linux-x86_64.sh
```

配置conda虚拟环境

conda 命令可能用不了

```
vim ~/.bashrc
```

在最后加上

```
export PATH="/root/anaconda3/bin:$PATH"
```

然后

```
source ~/.bashrc
```

再用conda 创造虚拟环境，先查看可用版本

```
conda search "^python$"
```

创造虚拟环境

```
conda create --name sven-Linux python=3.6.2
```

Pycharm

配置远程连接服务器的虚拟环境，就可以直接远程同步文件夹，运行本地的代码了

jupyter notebook

- 生成配置文件 jupyter notebook --generate-config
- 打开python 生成秘钥

```
from notebook.auth import passwd
passwd()
```

- 设置自己的登录密码，复制秘钥，秘钥开头是 sha
- 修改配置文件

```
vim ~/.jupyter/jupyter_notebook_config.py
```

- 然后修改或者直接加入

```
c.NotebookApp.ip='你的IP地址'                          # 就是多个用户使用
c.NotebookApp.password = u'sha:...'       # 刚才复制的那个密文，
c.NotebookApp.open_browser = False          # 禁止自动打开浏览器
c.NotebookApp.port =8888
c.NotebookApp.all_root=True     ##如果平时是用root用户登录的话可能需要
c.NotebookApp.notebook_dir=u'/root/' ##平时使用的默认地址
```

- 如果是阿里云服务器的话，给阿里云服务器添加安全规则，开放8888端口，其他的服务器可能也需要
- 在terminal输入jupyter notebook即可开始
- 如果是需要后台开启输入 nohup jupyter notebook &

Ubuntu配置定时任务

- 安装cron 一般已经默认安装了

```
sudo apt-get install cron
```

- 检测是否启动了服务

```
pgrep cron
```

- 如果没有反应就

```
service cron start
```

- cron 命令

```
cron [-u user] {-e | -l | -r}
-u 指定用户
-e edit
-l list
-r delete
```

- 任务计划的语法如下

```
m h dom mom dow commond
分钟 小时 日期 月份 星期 命令
```

- 最好在root用户下配置自动任务，不然可能没有权限，如何增加权限，还没搞清楚

Linux权限

文件权限

用ls -lh可以看到一个文件的权限

e.g. drwxr-xr-x

- 第一位代表 type: d表示文件夹, -表示文件
- 2-4位代表的user的权限，5-7位代表的是group的权限，8-10位代表的是others的权限
- r: read, w:write, x:excute
- 修改文件的权限
    - chmod u+r t1.py (chmod 是change mod的缩写, u代表user, r是read,全部命令就是赋予user对t1.py的read权限， 这里+可以换成-,表示删除对应的权限)
    - u代表user, g代表group, o代表others, a代表所有, 也可以使用ug表示(user+group)

Ubuntu的terminal的一些用法

文件和文件夹的处理

ls的用法

- ls -l 输出详细信息（一般用 ls -lh）
- ls -a 输出全部的文件/文件夹，包括隐藏文件

cp的用法

- cp file filecopy 但是这会导致强制覆盖
- cp -i file filecopy 会提示你是否需要overwrite
- cp file folder/ 把文件复制到文件夹
- cp -r folder1/ folder2/ 这里r表示的是recursive
- cp file *folder2/ 将以file开头的文件复制到folder2,当然也可以使用*.py表示以.py结尾的文件

mv的用法

- mv file folder/ 基础用法

mkdir, rmdir, rm

- mkdir folder 建立一个文件夹
- rmdir folder移除一个文件夹，但是只能移除一个空文件夹
- rm file 直接删除一个文件 也可以加 rm -i 进行interactive
- rm -r foler

cat的用法

- cat a.py 在屏幕上显示a.py里面的内容
- cat a.py > b.py 把a.py里面的内容放到b.py里面
- cat a.py b.py > c.py 把a.py, b.py里面的内容都放到c.py
- cat b.py >> a.py 把b.py里面的内容加到a.py后面

网络共享

Ubuntu Share文件夹之后需要

sudo smbpasswd -a sven(用户名)

然后会让你创建一个password,共享文献的password

但是这种功能只能在局域网里面实现

Ubuntu定时任务

Ubuntu的定时任务主要通过crontab命令来管理

其主要用法是:

```
crontab [-u user] {-e|-l|-r}
```

- -u user 指定用户，一般不写
- -e 编辑定时任务
- -l 列出定时任务
- -r 删除定时任务（应该是remove的缩写), 慎用，最好别用，用 crontab -e 去修改比较好。

在通过 crontab -e 编辑定时任务时，其主要格式是

```
m h dom mon dow command
```

- m 指定分钟
- h 指定小时
- dom 指定天
- mon 指定月份
- dow 指定周几
- command 指定命令

例如

```
0 1 * * * command
```

表示在每天的 1 :00 执行 command 命令

如果不在root用户下，碰到权限问题，可以考虑用

```
echo 'secret' | sudo command
```

来处理, 这里 secret 是当前用户的 sudo 密码。

Gitbook

使用的过程中碰到一些问题，主要都是公式方面的

- 公式不能用是没有 mathjax/katex。建议使用katex (mathjax在生成html的时候还可以，但是导出成pdf的时候格式很模糊), 但是katex对模型latex语法不认识，得修改使其更符合规范。 在根目录下建立book.json, 然后在里面添加

  ```
    "plugins": [
        "mathjax"
      ],
  ```

  然后运行

  ```
  gitbook install ./
  ```

- 行业公式需要用两个 '$', 在markdown中好像两者是一样的，但是在编译成html或者pdf的时候，好像必须要 两个，原因不明
- 碰到 SVG 问题，使用

  ```
  npm install svgexport@0.3.2 -g
  ```

  直接 npm install svgexport -g 的话安装的是0.4的版本，但是有bug, 碰到了坑，所以用0.3.2

  在 npm 默认的源是官方源，是国外的，改成国内的

  ```
  npm config set registry https://registry.npm.taobao.org
  ```

  使用以下代码看目前自己所有的源

  ```
  npm config list
  ```

  如果安装出现问题，可以考虑使用 `npm cache clean --force` 清除缓存，并将安装失败的项目中的**node_modules**文件夹删除，重新 `npm install`

- gitbook 可以导出本地的html

  ```
  gitbook build
  ```

但是导出的 html 不支持跳转， 具体原因是由于点击事件被js代码禁用，所以点击没有反应，但是如果右键，在新窗口/新标签页打开的话是可以跳转的。

去 /_book/gitbook 目录下找到 `theme.js` 文件, 搜索 `if(m)for(n.handler&&` , 将 `if(m)` 改为 `if(false)`

正常还是用 `gitbook serve` 方便

git的使用

git的用户配置

这是为了在每次对版本更改的时候，可以明确身份

- git config --global user.name 'sven'
- git config --global user.email 'chenliujun0556@163.com'

建立本地的仓库

- 在git bash中cd 到需要同步的文件夹
- git init 初始化
- touch a.py 一个示例，可以直接创建，或者修改文件也以
- git add a.py 将a.py加入git的管理
- git commit -m 'message' 提交修改
- git log 查看全部的修改记录
- 在 修改之后，git add 之前可以用git diff显示和之前已经commit的对比

返回之前的版本

- git commit --amend --no-edit 有时候需要提交的内容和上次的合并（那种忘了，突然加上的小修改）
- git reset a.py 返回到 unstaged 的状态，就是返回add之前
- git reset --hard HEAD~1 回到上次的 ~2回到上两次的
- git log --oneline 显示每次修改的ID
- git reset --head ID 直接回到那一次修改的结果
- git reflog 显示所有的修改记录（包括返回到过去的）
- git checkout ID -- a.py 只针对单个文件返回之前的状态，操作相当于对a.py再做了一次修改

分支

- git branch dev 新建一个叫dev的分支

- git branch查看目前所有的分支，*的表示当前所在的分支
- git checkout dev 切换到dev
- git branch -d dev 删除dev分支
- 分支之间是平行的
- git merge --no-ff -m 'message' dev 可以进行分支合并（先切换到主分支)

和Github之间的交互

- git remote add origin git@github.com:Sven1996/sven_asus.git(Github中你的仓库的地址) 建立连接
- git push -u origin master 上传文件到Github -u选项第一次加以后可以不加
- 如果push不成功的话加入 git pull --rebase origin master 再进行push
- git remote rm origin 删除origin
- ssh-keygen -t rsa -C "youremail@example.com" 产生本地的公钥
- 中间要求输入密码，可以不输入
- /c/Users/Sven/下有.ssh目录
- 在github的setting的ssh key中加入id_rsa.pub的内容

```
pandoc -t beamer -V theme:CambridgeUS slides.md -o slides.pdf
```

利用markdown写文件，然后利用pandoc将其转化为slides

```
pandoc -t beamer -V theme:CambridgeUS slides.md -o slides.tex
```

这个是将其变为latex源文件，然后可以方便修改主题等

但是这个有一个问题，不能输出作者和日期，有点尴尬

还有一个问题是对中文的支持可能不太好，得配置一下文件，具体的参见百度

```
pandoc --pdf-engine=xelatex -t beamer -V theme:CambridgeUS slides.md -o s
```