Table of Contents

前言

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
```

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)$$是可导的，凸的，并且是 $$\beta$$-Smooth 的，但是对于$$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 $$ \min_{x} \dfrac{1}{2}||y-Ax||^2+\lambda ||x||_1 $$

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)$$ 是 $$\beta$$ smooth的，所以 $$m(x)\ge f(x)+g(x)$$.

很容易验证（配方），

$$ m(x)=f(x_t)-\dfrac{1}{2\beta}||\nabla f(x_t)||^2+\dfrac{1}{2\beta}||x-(x_t-\dfrac{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})\le m(x*{t+1})\le 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} \dfrac{1}{2\beta}||x-(x_t-\dfrac{1}{\beta}\nabla f(x_t))||^2+g(x).

$$ 我们定义

$$ Prox*{g}(t)=\arg\min*{x}{\dfrac{1}{2}||x-t||^2+g(x)}.

$$ 则

$$ x*{t+1}=Prox*{\frac{1}{\beta}g}(x_t-\dfrac{1}{\beta}\nabla f(x_t)).

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

简介

经验似然方法是Owen(1980)提出的一种非参数统计推断方法,不需要假定数据来自一个具体的分布族.

经验似然的主要作用是用来寻找有效的估计， 并且构造待估参数（如均值）的置信区间。当然也可以用来做假设检验,这本身就是置信区间的对偶问题.当然,在参数模型假定正确的时候，非参数方法达不到参数方法的有效性，但是事实上很多时候，参数模型的假定都是不正确的.我们经常假定数据是正态的,这种假定在很多时候是合理的，但是也有很多状况下，数据不是正态的，这种时候我们再进行正态性的假定，肯定就是不对的. 因此，现在很多时候大家都倾向于用一些非参数的方法，比如Bootstrap, Jackknife等方法.

相比于Bootstrap来说，由于经验似然方法是用似然函数(likelihood function)，因此当有一些限制的时候， 经验似然方法只需变成有约束的优化问题就可以了，处理起来相对来说很简单.而且， 经验似然方法可以用 Bartlett修正来提高推断的精确性.通过Bartlett修正,很多时候可以将置信区间的覆盖错误率降低到 $O(1/n)$,达到和参数方法下MLE的精度.当然,经验似然的方法对计算的要求比较大，这是很多非参数方法都面临的一些问题，不过这些计算在今天来说都不是什么大问题.

经验似然的另一个巨大优势是在于其适用性很广,这本书里面讨论的更多的是样本独立同分布($i.i.d.$)的情况,不过经验似然方法可以推广到非独立同分布的情况，这在另一份报告里面有详细的说明，这里就不加赘述了. 接下来,我们将具体来描述经验似然方法是如何实施的.接下来的描述是我对[@owen2001empirical]的前四章的总结.

经验似然方法总结

随机变量的经验似然

对于一个随机变量 $X\in \mathbb{R}$,其累积分布函数(CDF) 是 $F(x)=Pr(X\le x),-\infty<x<\infty$. 我们用$F(x-)$ 表示 $Pr(X<x)$，那么 $Pr(X=x)=F(x)-F(x-)$. $X_1,\dots,X_n$ 的经验累积分布函数定义为:

$$F*n(x)=\frac{1}{n}* \sum{i=1}^n 1_{X_i\le x}.$$

假设 $X1,\dots,X_n \in \mathbb{R}$ 独立同分布，其分布函数为 F_0, CDF的非参数似然定义为 $$L(F)=\prod{i=1}^n(F(X_i)-F(X_i-)).$$ 非参数似然的最大值点就是ECDF,因此我们可以称ECDF是非参数的极大似然估计. 考虑一个原假设为 $H_0:T(F_0)=\theta_0$的经验似然检验,我们定义其 profile 的似然比函数为：

$$\mathcal{R}(\theta)=\sup {R(F)|T(F)=\theta,F\in \mathcal{F} }.$$ 这里, $R(F)=\dfrac{L(F)}{L(\hat{F})}$. 当 $R(\theta_0)$小于一个临界值 $<r_0$的时候，拒绝原假设.这里r_0的选取可以用经验似然定理.经验似然置信区间的形式如下：

$${ \theta|R(\theta)\ge r_0}.$$ 均值的profile 经验似然比函数是：

$$R(\mu)=\max {\prod{i=1}^n nw_i|\sum{i=1}^n w_i X_i=\mu,w_i\ge0,\sum{i=1}^n w_i=1 }.$$ 并且其对应的经验似然置信区间是:

$${ \mu|R(\mu)\ge r0}={ \sum{i=1}^n wiX_i|\prod{i=1}^n nwi\ge r_0,w_i\ge 0,\sum{i=1}^n w_i=1}.$$

[[UniELT]]{#UniELT label="UniELT"} 假设 $X_1,\dots,X_n$ 是独立同分布的随机变量，其分布函数为 F_0. 令 $\mu_0=E(X_i)$, 并且假设 th$0<Var(X_i)<\infty$. 那么当$n\to \infty$时, $-2\log(R(\mu_0))$ 会依分布收敛到一个卡方分布，其自由度为1.

定理 [UniELT]{reference-type="ref" reference="UniELT"} 给出了 $-2\log(R(\mu0))$的渐近分布， 这可以来帮我们选择假设检验问题（和置信区间）中的临界值，当显著性水平为 α 时, 如果 $-2\log R(\mu_0)>\chi^{2,1-\alpha}{(1)}$，我们就拒绝原假设. 从证明的细节，和一些模拟的结果来看，如果将临界值$\chi{(1)}^{2,1-\alpha}$ 替换为$F{1,n-1}^{1-\alpha}$，效果会更好.有了临界值，我们就可以构造均值的置信区间了,其具体形式如下:

$${ \mu|-2\log R(\mu)\le\chi^{2,1-\alpha}{(1)} }={\mu| R(\mu) \ge \exp(-\chi{(1)}^{2,1-\alpha}/2)}.$$ 在一些比较宽松的矩条件下, 置信区间的覆盖错误率大概以 $O(1/n)$的速度收敛到0, 这种收敛速度是和参数模型下的极大似然估计是一样的, 并且经验似然推断的势和参数模型下的推断差不多，因此经验似然方法能达到很好的精度. 这里 $R(\mu)$的计算是一个带约束的优化问题，因此我们考虑用Lagrange乘子法, 简单的计算可以得到

$$wi=\dfrac{(X_i-\gamma)^{-1}}{\sum{j=1}^n (X_j-\gamma)^{-1}},$$ 这里λ 是下列方程的解

$$\dfrac{1}{n} \sum_{i=1}^n \dfrac{X_i-\mu}{1+\lambda(X_i-\mu)}=0.$$ 具体的计算，之前也写过一份文档来说明，这里就不加赘述了.

随机向量的经验似然方法

随机向量的经验似然其实就是对随机变量的推广，从一维到多维.首先有一些概念要重新定义一下.假设随机变量 X的分布函数是 F, $F(A)$ 的意思是 $Pr(X\in A)$ 对于 $A\subset \mathbb{R}^d$. 让 $\deltax$ *表示 以概率1，$X=x$的一个分布，* $\delta_x(A)=1${x\in A}$. $X_1,\dots,X_n$的经验累积分布函数为

$$F*n=\frac{1}{n}* \sum{i=1}^n \delta*{X_i}.$$ *假设 $X_1,\dots,X_n \in \mathbb{R}^d$独立同分布于共同的分布函数 F_0, F的非参数似然是 $$L(F)=\prod*{i=1}^n F({ X_i}).$$* 随机向量的均值的 profile经验似然比函数为

$$R(\mu)=\max { \prod*{i=1}^n nw_i |\sum*{i=1}^n wi X_i=\mu,w_i\ge0,\sum{i=1}^n w_i=1}.$$ 构造的置信区间如下

$$C*{r,n}={ \sum*{i=1}^n wiX_i |\prod* {i=1}^n nwi \ge r,w_i\ge 0,\sum{i=1}^n w_i=1}.$$ 随机变量的 ELT(定理 [UniELT]{reference-type="ref" reference="UniELT"}) 可以推广到随机向量的情况，

[[VectorELT]]{#VectorELT label="VectorELT"} 假设 $X*1,\dots,X_n$ 是 $\mathbb{R}^d$上的随机向量，其独立同分布于一个共同的分布函数 F_0，其均值是 μ_0 ，并且协方差矩阵V_0有限，秩是 $q>0$. 则 $C*{r,n}$ 是一个凸集并且随着 $n\to \infty$， $-2\log R(\mu*0)$ 收敛到一个自由度是q的卡方分布 $\chi^2*{(q)}$.

(A Sketch):

1. 首先证明 μ_0 在 X_i的凸壳中,
2. 然后说明 Lagrange 乘子的阶是 $\lambda=O_p(n^{-1/2})$
3. 说明 $\lambda=S^{-1}(\bar{X}-\mu_0)+o_p(n^{-1/2})$， S是样本协方差矩阵
4. 把 λ 的表达式带入 profile 经验似然比统计量，然后应用中心极限定理，并且证明其他项的阶更小，是可以忽略的,这样就完成的整个定理的证明. 具体的证明请参见Owen(1998).

通常情况下,$V*0$ 是满秩的 $q=d$, 但是如果 V_0不是满秩的,假设 V_0的秩是q的话，我们只需调整卡方分布的自由度就可以了.在实际应用的时候，我们很多时候是不知道V_0的秩的，这个时候我们一般就用样本协方差矩阵的秩来代替，前提是 $n \gg d$. 同样的，和一维的情况（随机变量），定理 *[VectorELT]{reference-type="ref" reference="VectorELT"}* 给了我们一种选取临界值r的方法，在显著性水平为α的时候，我们选取 $r=\exp(-\chi*{(q)}^{2,1-

\alpha}/2).$同样，将 $\chi^2*{(q)}$ *替换成 $(n-1)q/(n-q)F*{q,n-q}$ 效果会更好. As $n \to \infty$ 两者其实是等价的. 其实在参数模型里面也是一样，这种 F修正也是有用的.

Fisher, Bartlett, and bootstrap 修正

- Fisher: 对于正态随机向量 $X*i$, 我们知道 $n(\bar{X}-\mu_0)^{'} V_0^{-1}(\bar{X}-\mu_0)\sim \chi*{(d)}^2$. 但是在 $V0$不知道的情况下,我们一般会选择用 Hotelling's T^2统计量的分布, $(n-d)T^2/((n-1)d)\sim F*{d,n-d}$, 这就是 Fisher修正.
- Bartlett: Bartlett 修正 是将 $\chi*{(d)}^{2,1-\alpha}$ *替换为 : $$(1-\frac{a}{n})^{-1} \chi*{(d)}^{2,1-\alpha}, \quad \text{或者} \quad (1+\frac{a}{n})^{-1} \chi_{(d)}^{2,1-\alpha}$$ 这里 a是一个需要选取的常数. Bartlett 修正将置信区间的覆盖错误率降低到了 $O(1/n^2)$.
- Bootstrap: 利用 Bootstrap的方法来得到置信域, 当 n 比较小的时候，这种方法的效果会比 Bartlett修正好.

均值的光滑函数

假设 $X_i\in \mathbb{R}^d$是独立同分布的随机向量，其共同分布为 F_0，均值为 μ_0. 假设 h 是一个光滑函数，$h:\mathbb{R}^d\to \mathbb{R}^q$,这里 $1\le q\le d$. 我们想要估计的参数是 $\theta_0=h(\mu_0)$,显然非参数极大似然估计 NPMLE 是 $\hat{\theta}=h(\bar{X})$. 其 profile 经验似然比函数是

$$R(\theta)=\max { \prod*{i=1}^n nw_i | h(\sum{i=1}^n wiX_i)=\theta,w_i\le 0,\sum{i=1}^n w_i=1 }.$$

估计方程

我们首先来给出估计方程的定义, 对于一个随机向量 $X \in \mathbb{R}^d$, 参数 $\theta \in \mathbb{R}^p$,和一个方程 $m(X,\theta)\in \mathbb{R}^s$，则 $$E(m(X,\theta))=0,$$ 构成了一个估计方程.通常情况下，$p=s$, 估计方程只有一个解. 参数的真实值 $\theta*0$ *可以通过下面这个方程解出来: $$\label{EstimatingEquation} \frac{1}{n} \sum*{i=1}^n m(X_i,\hat{\theta})=0$$ 但是有些时候, s 未必一定等于 p. When $s>p$,方程可能是没有解的. 常用的方法是尽可能找到一个值 $\hat{\theta}$去接近 ([EstimatingEquation]{reference-

type="ref" reference="EstimatingEquation"}). 当 $s<p$,估计方程 ([EstimatingEquation]{reference-type="ref" reference="EstimatingEquation"})有 $s-p$维的解.

我们定义θ的经验似然比函数

$$R(\theta)=\max {\prod*{i=1}^n nw_i|\sum{i=1}^n w*im(X_i,\theta)=0,w_i\ge0,\sum{i=1}^n w_i=1 }$$ 令 $Y_i=m(X_i,\theta_0)$,并且 $Var(Y_i)=Var(m(X_i,\theta_0))$是有限的并且秩是$q>0$, 我们可以从 随机向量的经验似然定理可以直接得到下面的这个定理.

[[EE ELT]]{#EE ELT label="EE ELT"} 假设 $X*i,\dots,X_n\in \mathbb{R}^d$ *独立同分布于一个共同的分布函数 F_0. 对于 $\theta_0 \in \Theta \subseteq \mathbb{R}^p$, $X\in \mathbb{R}^d$, $m(X,\theta)\in \mathbb{R}^s$. 令 $\theta_0 \in \Theta$ 并且使得 $Var(m(X_i,\theta_0))$ 有限,并且秩是 $q>0$. 当 θ_0 满足 $E(m(X,\theta_0))=0$时, 我们有当$n \to \infty$,$-2\log R(\theta_0)$依分布收敛到$\chi^2{(q)}$.*

定理 [EE ELT]{reference-type="ref" reference="EE ELT"}没有任何对θ_0的估计$\hat{\theta}$的限制，甚至都没有要求θ_0存在.因此定理 [EE ELT]{reference-type="ref" reference="EE ELT"} 可以应用到 $s=p,sp$的情况.

讨厌参数

当参数集合里面有讨厌参数的时候,我们考虑将我们感兴趣的参数和讨厌参数分开,我们将估计方程写为 $m(X,\theta,\nu)=0$, 这里 $\theta \in \mathbb{R}^p$ 是我们感兴趣的参数, $\nu \in \mathbb{R}^q$ 是讨厌参数, $m \in \mathbb{R}^s$. 参数集合 (θ,ν) 满足估计方程 $E(m(X,\theta,\nu))=0$. 一般来说, $s=p+q$. 现在我们定义:

$$R(\theta,\nu)=\max { \prod*{i=1}^n nw_i|\sum{i=1}^n m(X*i,\theta,\nu)=0,w_i\ge0,\sum{i=1}^n w*i=1 },$$ *以及 $$R(\theta)=\max{\nu} R(\theta,\nu).$$* 我们的方法就是对讨厌参数取使似然函数的最大值,其实也就是把讨厌参数用他们的NPMLE来代替.

分位数

X的α分位数Q^{α}的定义是:$Pr(X\le Q^{\alpha})\ge \alpha$,并且 $Pr(X\ge Q^{\alpha})\le \alpha$.

这一节,我们不考虑结的情况，并且将分位数的定义简化为

$$E(1{X\le Q^{\alpha}}-\alpha)=0.$$ *我们定义:*$X_0=-\infty$,$X{n+1}=\infty$ 已确保可以包括 $Q^{\alpha}<X*{(1)}$和 $Q^{\alpha}\ge X*{(n)}$的情况. 定义$Z*i(p,q)=1*{X_i\le q}-p$， 则分位数经验似然比方程为

$$R(p,q)=\max {\prod*{i=1}^n nw_i |\sum*{i=0}^{n+1} w*i Z_i(p,q)=0, w_i\ge 0.\sum*{i=0}^{n+1}=1 }.$$ 经过一些简单的计算,我们可以发现:

- 当$X*{(1)}<q\le X*{(n)}$, 经验似然比方程为 $$R(p,q)=(\frac{p}{\hat{p}})^{n\hat{p}}(\frac{1-p}{1-\hat{p}})^{n(1-\hat{p})}.$$ 这里 $\hat{p}=\hat{p}(q)=#{X_i \le q}/n$.
- 当 $q<X_{(1)}$, 经验似然比方程为 $$R(p,q)=(1-p)^n.$$
- 当 $q\ge X_{(n)}$, 经验似然比方程为 $$R(p,q)=p^n.$$

将这三种情况综合起来,就是

$$-\log R(p,q)=n[\hat{p}\log(\hat{p}/p)+(1-\hat{p})\log((1-\hat{p})/(1-p))],$$ 这里 $\hat{p}=\hat{p}(q)=#{X_i \le q}/n$.

结点和分位数

结点

如果数据有结,假设总共有 k个不同的取值,记为 $z_1,\dots,z_k$, 假设 z_j出现了　n_j次, 并且在F下出现的概率是 p_j.则

$$R(F)=\prod*{j=1}^k (\frac{p_j}{\hat{p}_j)^{n_j}=\prod*{j=1}^k (\frac{np*j}{n_j})^{n_j}.$$ *我们可以不考虑结点，并假定每个点在F下出现的概率分别为*w_i，只要满足 p_j*是所有的满足　*$X_i=z_j$*的那些*i*对应的*w_i*相加.而如果不考虑这些约束,*$\prod{i=1}^n w*i$*的最大值点肯定*$w_i=1/n$*.而考虑这些约束,最大值点肯定是满足* $w_i=p{j(i)}/n_{j(i)}=1/n$,在有没有约束情况下，最大值点都是一样的. 因此实际上,我们可以不考虑这个约束条件.

这也就意味着,结点不会对经验似然方法构造的置信区间产生影响.

结点和分位数

根据我们上一节的讨论，结点在经验似然方法应该不是一个值得关注的问题,因为结点不影响置信区间的构造.　但是我们这一节还是要考虑结点, 原因在于结点会影响我们对分位数的定义. 在3.7那一节,我们假设没

有结点,并将分位数的定义简化为$E(1_{X\le Q^{\alpha}}-\alpha)=0,$但是实际上我们知道分位数的定义是

$$\label{Quantiledef} Pr(X\le Q^{\alpha})\ge \alpha,\quad \text{和}\quad Pr(X\ge Q^{\alpha})\ge 1-\alpha.$$ 而如果有结点的话，那么这种简化明显是有问题的.一节简单的接近办法就是我们将每个数据随机的加上一个极小的量, 比如我们给每个 $X*i$*加上一个来源与均匀分布 $(-A,A)$的扰动, 这里 A取非常小的值.这样我们就解决了结点的问题.但即使是没有结点的情况，这种对分位数定义的简化也是有一定偏差的. 一个简单的例子是在*t$n = 2k + 1$时,样本中位数 $X*{k+1}$的经验似然比应该是 1 ,但是X_{k+1}的经验似然比是

$$(\frac{n}{2k})^k (\frac{n}{2(k+1)})^{k+1} \stackrel{\cdot}{=} 1-\frac{1}{2n}.$$ 对于这种中位数估计方程,我们给出一个修正: $E(1*{X\le Q^{\alpha}}+\alpha 1*{X=Q^{\alpha}}-\alpha)=0,$

经验似然的最大似然原理不变性

类似与参数似然中的最大似然原理不变性,经验似然也有类似的性质.

在经验似然中,最大似然原理不变性分为两个方面:

- 参数的变换: if $\sum*{i=1}^n w_i m(X_i,\theta,\nu)=0$, then $\sum{i=1}^n w_i m(X_i,\tau^{-1}(\phi),\nu)=0$.
- 数据的变换: 假设 $Y=\tau(X)$ 是个一一对应的变换. 那么 $\sum*{i=1}^n w_im(X_i,\theta)=0$ 当且仅当 $\sum{i=1}^n w_i m(\tau^{-1}(Y_i),\theta)=0$.

辅助信息

假设 $(X,Y)\in \mathbb{R}^2$,并且我们知道 X的均值,我们想要构造 Y的均值的估计和置信区间. 我们定义条件经验似然比函数如下. 首先,我们定义$R_{X,Y}(\mu_x,\mu_y)$为

$$\max{\prod*{i=1}^n nw_i|\sum{i=1}^nwiX_i=\mu_x,\sum{i=1}^nwiY_i=\mu_y,w_i\ge 0,\sum{i=1}^n w_i=1 },$$ 并且令

$$RX(\mu_x)=\max{\prod{i=1}^n nwi|\sum{i=1}^nwiX_i=\mu_x,w_i\ge 0,\sum{i=1}^n w_i=1 },$$ 然后定义

$$R*{Y|X}(\mu_y|\mu_x)=\frac{R{XY}(\mu_x,\mu_y)}{R_X(\mu_x)}.$$

我们有如下定理:

[[SideInf]]{#SideInf label="SideInf"} 假设 $(X*i,Y_i)\in \mathbb{R}^{p+q}$是独立同分布的随机向量,其共同分布为$F_0$, 均值是 $(\mu{x0},\mu{y0})$, 方差矩阵满秩，秩为 $p+q$. 那么随着 $n \to \infty$, $-2\log R{Y|X}(\mu{y0}|\mu{x0})$依分布收敛到$\chi^2_{(q)}$.

定理 [SideInf]{reference-type="ref" reference="SideInf"} 说明了条件经验似然方程和简单的经验似然方程有相同的渐近分布. 但是,在有辅助信息的情况下,方程是要比没有辅助信息的情况下小的, 因此最后构造的置信域也是要更小一点的.

当估计方程的个数多于未知参数的个数时， 参数推断经常是有问题的. 假设在一个参数模型下, 有 $p+q$个估计方程,有 q个未知参数, 我们可以选择其中的q个估计方程，忽略其余的p个. 更加general的来说， 我们可以用这$p+q$个估计方程的 q个线性组合:

$$\label{linCombinEE} E(m(X,\theta)A(\theta))=0$$ 这里 $A(\theta)$是 $(p+q)\times p$的矩阵，其秩是 q. 当估计方程m关于 参数足够光滑的时候, 经验似然方法得到的渐近方差一般来说是要比这种线性组合得到的方差小的,至少是不会比它大的.

假设 $Xi \in \mathbb{R}^d$ 是独立同分布的随机向量, 并且假设 $\theta_0\in \mathbb{R}^p$ 是被 $E(m(X,\theta))=0$ 唯一决定的, 这里$m(X,\theta)\in \mathbb{R}^{p+q}$, for $q\ge 0$. 令 $\tilde{\theta}=\arg \min{\theta} R(\theta)$, 这里

$$R(\theta)=\max { \prod*{i=1}^n nw_i |\sum{i=1}^n wi m(X_i,\theta)=0,w_i\ge 0,\sum{i=1}^n w_i=1}$$ 在一些正则条件下, 我们有

$$\lim _{n \to \infty} n Var(\tilde{\theta})=[E(\frac{\partial m}{\partial \theta})^{'}(E(mm^{'}))^{-1}E(\frac{\partial m}{\partial \theta})]^{-1}$$ 这个渐近方差是至少不比 ([linCombinEE]{reference-type="ref" reference="linCombinEE"}) 估计得到的方差大的.并且我们有

$$-2\log(R(\theta0)/R(\tilde{\theta}))\to \chi^2{(p)},$$ 以及

$$-2\log R(\tilde{\theta}) \to \chi^2_{(q)}.$$

这个定理的证明在 $Qin and Lawless(1994)$ 中有详细的叙述，并且比较复杂，我在这里就不加叙述了.这个定理描述的是独立同分布的状况下，但是其实可以扩展到很多的非独立同分布的状况,这在另一份报告里面有比较详细的描述.

三明治估计量

在估计方程的个数等于参数的个数的时候,即$r=p$时, 给定一些比较宽松的正则性条件,$\sum*{i=1}^n m(X_i,\theta)=0\sum*{i=1}^n m(X_i,\theta)=0$的解 $\hat{\theta}$ 的方差满足

$$n Var(\hat{\theta}) \to I^{-1} C(I^{'})^{-1},$$ 这里 $$\begin{split} I &=\frac{\partial}{\partial \theta} \int m(X,\theta)d F|_{\theta=\theta_0},\ C &=\int m(X,\theta_0)m(X,\theta_0)^{'} dF(x). \end{split}$$ 根据以上的公式,我们可以得到$\hat{\theta}$的方差的三明治估计

$$\hat{Var}_{Sand}(\hat{\theta})=\dfrac{1}{n} \hat{I}^{-1}\hat{C} \hat{I}^{'-1}.$$

稳健估计

如果数据有异常点的时候, 经验似然构造的置信区间变大. 我们考虑两种方式来达到稳健的效果.第一种是利用一个稳健的估计量, 第二种是对似然函数做一些调增，来达到稳健的效果，这个在下一节会有叙述.这一节我们考虑稳健的估计量. 比如用样本均值来估计总体均值是不稳健的, 我们可以考虑用 Huber's-M估计量. Huber's M估计量是通过求解以下估计方程得到

$$\frac{1}{n} \psi(\frac{X_i-\mu}{\hat{\sigma}})=0$$ 这里 $\hat{\sigma}$ 是尺度的一个稳健估计, $$\psi(z)=\left { \begin{array} {lll} -c, && z\le -c \ z,&& |z|\le c\ c,&& z>c. \end{array} \right.$$

稳健似然

这里，我们考虑第二种达到稳健性的方法:对似然函数做出一定的修正. 考虑数据来源于一个离散分布, $Pr(X=x;\theta)=f(x,\theta)$, 并且数据有异常值.也就是说数据$X_i$，有 $1-\epsilon$ 的概率数据来源于 $f(x,\theta)$, 有 where ϵ 的该来了来源于一个未知的分布 G_i, 这里$\epsilon>0$ 是一个很小的数.我们假设观测之间是独立的. 那么似然方程为

$$L(\theta,G_1,\dots,G_n)=\prod((1-\epsilon)f(X_i,\theta)+\epsilon G_i(X_i)).$$ 取 $G_i=1$使似然方程最大化,则,

$$L(\theta,G_1,\dots,G_n)=(1-\epsilon)^n \prod(f(X_i,\theta)+\eta),$$ 这里$\eta=\epsilon/(1-\epsilon)$.

计算与对偶性

经验似然的求解问题,其实是一个带约束的优化问题,可以用Lagrange乘子法, 然后用牛顿迭代求解,这个在之前的一份作业里面已经将此与代码一并说明,这里就不加赘叙了.

欧式似然和其他似然

可以把$-\sum_{i=1}^n \log(nw_i)$看成是一种对$(w_1,\dots,w_n)$到$(n^{-1},\dots,n^{-1})$距离的测度. 那么，一个自然的想法，就是利用其他距离来考虑这个问题，比如欧式距离,KL距离，Hellinger距离,Cressie-Read power距离. 以欧式距离为例, 欧式对数似然比方程为

$$l*E*=-*frac{1}{2}* *sum*{i=1}^n (nw_i-1)^2.$$

回归与建模

这一章考虑用经验似然方法来推断广义线性模型.

随机预测变量

随机预测和非随机预测变量对应的其实是回归的两种假设. 在非随机预测里面, 我们假设 X_i是非随机的，更多是我们设计的，所以也被成为设计矩阵,也是之前比较常见的回归模型. 但是现在, 我们很多时候碰到的是观察数据，而不是实验数据，因此认为X_i也是随机产生的.

假设 $(X*i,Y_i)$ *是独立同分布的随机观测. 假设 $E(X^{'}X)$ 是满秩的 p. $\beta*{LS}$的最小二乘估计就是 $$\beta*{LS}=E(X^{'}X)^{-1}E(X^{'}Y)$$ 并且对*$\beta*{LS}$的估计就是(样本代替总体):

$$\hat{\beta}*{LS}=(\frac{1}{n}\sum*{i=1}^n X*i^{'}X_i)^{-1}(\frac{1}{n}\sum*{i=1}^n X*i^{'}Y_i).$$ *回归模型可以用估计方程来表示, 根据 $\beta*{LS}$的定义, 我们知道

$$E(X(Y-X^{'}\beta_{LS}))=0.$$ 定义一个随机变量 $Z_i=Z_i(\beta)=X_i(Y_i-X_i^{'}\beta)$, β 的经验似然比函数就被定义为

$$R(\beta)=\max {\prod*{i=1}^n nw_i|\sum*{i=1}^n Z*i(\beta)=0,w_i\ge 0,\sum*{i=1}^n w_i=1 } .$$

非随机预测变量

这部分内容, 我们假定$X*i$是非随机的.因此我们可以使用给定$X_1=x_1,\dots,X_n=x_n$下, $Y_1,\dots,Y_n$的条件似然. 在前一章的时候,我们说明过在给定观测$\frac{1}{n}\sum{i=1}^n U_i$下的条件似然等同于加一个约束条件,$\sum w_iU_i=E(U)$. 根据这个思路,我们在计算 $R(\beta)$的时候,我们加上这两个约束:

$$\sum*{i=1}^n w_ix_i=\frac{1}{n}\sum{i=1}^n,$$ 和

$$\sum*{i=1}^n w_ix_ix_i^{'}=\frac{1}{n}\sum{i=1}^n x_ix_i^{'}.$$

三角阵经验似然定理

这是一个经验似然应用到独立但是不同分布的例子.

[[Triangular array ELT]]{#Triangular array ELT label="Triangular array ELT"} 假设 $Z*{in}\in \mathbb{R}^p$, $1\le i\le n$,$n\ge n{min}$ 是一组三角数组排列的随机向量. 假设对于任意的n, $Z*{1n},\dots,Z{nn}$都是独立的，并且有共同的均值$\mun$. 令$\mathcal{H}_n$ 表示 $Z{1n},\dots,Z*{nn}$构成的凸壳, 并且令 $\sigma{1n}=maxeig(Vn)$ 以及 $\sigma{pn}=mineig(V_n)$. 假设随着 $n\to \infty$,

$$Pr(\mu_n \in \mathcal{H}_n)\to 1$$ 并且

$$\frac{1}{n^2}\sum*{i=1}^n E(||Z{in}-\mun||^4\sigma{1n}^{-2})\to 0.$$
存在 $c>0$ 使得对 $n\ge n_{min}$,有

$$\frac{\sigma*{pn}}{\sigma{1n}}\ge c.$$ 则随着$n \to \infty$, $-2\log R(\mun)$依分布收敛到$\chi^2{(p)}$,这里

$$R(\mun)=\max { \prod{i=1}^n nwi |\sum{i=1}^n wi(Z{in}-\mun)=0,w_i\ge 0,\sum{i=1}^n w_i=1}.$$

定理 [Triangular array ELT]{reference-type="ref" reference="Triangular array ELT"}的证明和独立同分布情况下随机向量的经验似然定理的证明很类似,最大的差别是在中心极限定理的应用上.独立同分布情况下,可以直接用普通的中心极限定理就可以了,但是在这里,我们需要用到Lyapunov中心极限定理. 定理的条件 $\frac{1}{n^2}\sum*{i=1}^n E(||Z{in}-\mun||^4\sigma{1n}^{-2})\to 0$ 是为了满足Lyapunov条件.

ANOVA分析

ANOVA分析主要用来比较不同组之间的均值. 假设我们观测到独立的随机变量$Y*{ij}\in \mathbb{R}$, $i=1,\dots,k,j=1,\dots,n_i$. 我们把这些数据重新编码成 N对 (I,Y)， 这里$I \in {1,\dots,k}$,$Y\in

\mathbb{R}^d$.这样,观测 Y_ij 可以表示成 $I=i, Y+Y{ij}$. 假设 $\mu_{i}^0=\int y dF_i^0(y)\in \mathbb{R}^d$,定义

$$R(\mu1,\dots,\mu_k)=\max { \prod{i=1}^k \prod{j=1}^N Nw{ij}|w{ij}\ge 0,\sum{i=1}^k \sum{j=1}^{n_i}w{ij}=1,\sum{j=1}^{n_i} w{ij}(Y_{ij}-\mu_i)=0,j=1,\dots,k}.$$

对方差建模

最小二乘估计要求的是Y_i是一个常数, 如果这个条件不满足了，数据有异方差性，那么用最小二乘来估计，模型就会有偏差. 我们就可以考虑用加权最小二乘估计来提高精确性:

非线性最小二乘

有些时候, Y与X的关系不是线性的,在不满足线性约束时，我们更加广义的定义最小二乘,通过以下式子来估计 θ: $$\arg \min{\theta} \sum{i=1}^n (Yi-f(x_i,\theta))^2,$$ 定义 $g(x_i,\theta)=\frac{\partial}{\partial\theta}f(x_i,\theta)$, 则估计方程是 $$\sum{i=1}^n w_i (Y_i-f(x_i,\theta))g(x_i,\theta)=0.$$

广义线性模型

经验似然方法可以应用到广义线性模型上. 广义线性模型一般由残差的分布和连接函数(link function)决定来决定. 这里我们以logistic回归为例，来看这个问题.

假设 $Yi \in {0,1}$是独立的, 并且 $Pr(Y_i=1|X_i=x_i)=\tau(x^{'}\beta)$, 这里 $\tau(z)=\text{logit} (z)$, 则估计方程是 $$\sum{i=1}^n x_i(Y_i-\tau(x_i^{'}\beta))=0.$$ Logistic回归的经验似然推断可以基于以下的经验似然函数

$$R(\beta)=\max { \prod{i=1}^n nw_i |\sum{i=1}^n Zi(\beta)=0,w_i\ge 0,\sum{i=1}^n w_i=1},$$ 这里 $Z_i(\beta)=x_i(Y_i-\tau(x_i^{'}\beta))$.

\newpage \bibliographystyle{plainnat}

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
```