利用数学建模安排学习计划

——以对Ebbinghaus遗忘曲线的拟合分析为例

14-唐化-廉欣然 12140912

**摘要**：本文不用很严肃的tune描述，因为我没参加过建模比赛，什么都不会，也不懂格式，而且时间比较仓促，只是粗糙地分析了一下遗忘曲线，并且浅显地分析了一下应该怎么学习，当然学习这件事也是因人而异的，所有叙述仅供参考。

**关键词：**Ebbinghaus遗忘曲线 复习 Python

**一、Ebbinghaus遗忘曲线的拟合**

先用小学数学计算一下你付出的努力：

假设是普通的64课时的课，16周，一周两大节。你可以每节课好好听，并且用1个小时复习；也可以平时闷声作大死，考前预习…如果选择前者，每学期付出的时间成本是：

16×2×（1.5+1）=80h

只有80小时，似乎有些意外，平时累成狗，一学期居然只花了那么点时间，那么如果你提前10天预习的话每天只需要8小时就能赶上，刷几天夜就够了。但是怎么都感觉不太对，所以还是分析一下比较好。

下面用最普通的Ebbinghaus遗忘曲线分析。

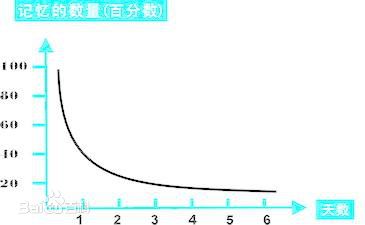


图1 Ebbinghaus遗忘曲线（来自百度百科）[1]

Ebbinghaus遗忘曲线是经验曲线，是根据实验结果得到的，根据下表（来自百度百科）的结果，如果你学完不复习，6天后只能记得25%，也就是1/4，那么考前预习的作死性就很明显了。然而为了严肃，也为了研究一下平时复习的事情，还是用Python进行定量的分析拟合一下。

|  |  |
| --- | --- |
| **时间间隔** | **记忆量** |
| 刚记完 | 100% |
| 20分钟后 | 58.2% |
| 1小时后 | 44.2% |
| 8~9小时后 | 35.8% |
| 1天后 | 33.7% |
| 2天后 | 27.8% |
| 6天后 | 25.4% |

首先是对遗忘曲线的拟合，看那个曲线的样子似乎是幂衰减或指数衰减，文献[2]表明适合用幂函数拟合，但由于我很好奇，所以花了些时间用指数函数试着拟合了一下。

Python的curve\_fit()函数比较好用[3]（因为我不会用matlab…），设好拟合方程几乎就可以一键拟合，并且determination能达到99%以上。因为我是菜鸟，对于拟合还是一知半解，所以就对照别人的文章进行了指数拟合，拟合公式如下：

使用curve\_fit得到的结果：

k=0.30499393

a=0.75843018

c=0.45421855

b=0.24115065

作图：

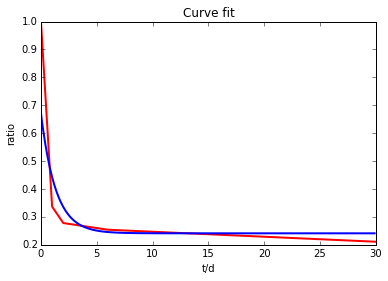


图2 指数拟合结果

（红色为表格中点的作图，蓝色为拟合曲线）

相当地不标准，t较大时遗忘曲线还是有一定的衰减速度的，指数函数参数再多拟合地也不太好，因此我用幂函数拟合：

|  |
| --- |
|  |
|  | import numpy as np |
|  | from scipy.optimize import curve\_fit |
|  |  |
|  | def func(x, a, b,k): |
|  | y = k/(x+a)+ b |
|  | return y |
|  |  |
|  | def polyfit(x, y, degree): |
|  | results = {} |
|  | #coeffs = numpy.polyfit(x, y, degree) |
|  | popt, pcov = curve\_fit(func, x, y) |
|  | results['polynomial'] = popt |
|  |  |
|  | # r-squared |
|  | yhat = func(x ,popt[0] ,popt[1],popt[2] ) # or [p(z) for z in x] |
|  | ybar = np.sum(y)/len(y) # or sum(y)/len(y) |
|  | ssreg = np.sum((yhat-ybar)\*\*2) # or sum([ (yihat - ybar)\*\*2 for yihat in yhat]) |
|  | sstot = np.sum((y - ybar)\*\*2) # or sum([ (yi - ybar)\*\*2 for yi in y]) |
|  | results['determination'] = ssreg / sstot |
|  |  |
|  | return results |

我建模建完觉得使用的公式还是不太好，但对长期遗忘率的拟合度比指数函数不知道高到哪里去了。

最后结果如下：

作图：

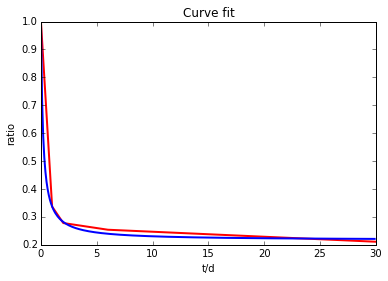


图2 幂函数拟合结果

（红色为表格中点的作图，蓝色为拟合曲线）

值得一提的是我舍弃了最初几小时的点（见表格），因为前几小时的遗忘速度实在是凶残，都纳入考虑会导致长期结果准确度降低，当然不排除是参数少的恶果。后面对于复习的分析就使用这个公式了。

**二、学习计划的分析**

首先分析“上课认真听，每次复习1小时，期末再次复习”的计划，看上去这是最好的，实际上算起来也是不错的选择。

由于参数简单，分析也比较容易，见下：

k：遗忘速度参数，越大忘的越快

b：我称为复习基数，即不复习，时间趋向于无穷的时候的最终记忆量。因人而异，因科目而异，偏向于理解性的值比较大，因为如果你理解了就算不复习也容易形成长期记忆，而死记硬背的就比较小了。

a：与复习时间有关，在分析中用处不大，k，b确定的时候a用来确定初始值（下面会讨论）

再次复习的时候k和b会变化，要进行修正，下面进行详细分析：

显然随着复习次数n的增加，

当然n并不需要很大，复习N次（通常<10）就能达到全部记忆，而根据不知道哪里的说法是7次。N因人而异，1/N可以作为一个学习效率的指标。

但是实际分析过程中我没有用到N，或者说本来考虑用到但最后没有使用。下面分析k和b与n的关系。设第i次学（复）习后，参数分别是ki，bi，

考虑使用线型公式

但是此式不如指数形式准确，在此不加赘述。

详细的指数形式推导见下述代码：

|  |
| --- |
|  |
|  | import numpy as np |
|  | import matplotlib.pyplot as plt |
|  |  |
|  | k=[0,0,0] |
|  | a=[0,0,0] |
|  | b=[0,0,0] |
|  | k[0]=0.14307295 |
|  | a[0]=0.18259039 |
|  | b[0]=0.21640213 |
|  | #假设过0.5天复习一次,过60天再复习一次(即期末复习),第66天考试 |
|  | tt=[0,0.5,60] |
|  |  |
|  | def func(i,x): |
|  | y=k[i]/(x+a[i])+ b[i] |
|  | return y |
|  |  |
|  | for i in range(1,3): |
|  | k[i]=k[i-1]/2 |
|  | b[i]=b[i-1]+(1-b[i-1])/2 |
|  | a[i]=k[i]/(1-b[i])-tt[i] |

最后得出如下复习曲线

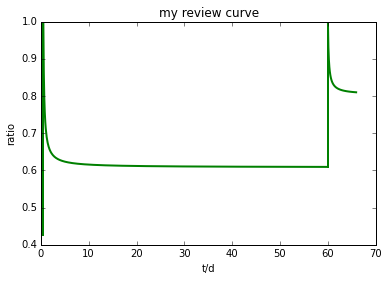


图2 幂函数复习曲线

好了，看一下考试时的结果（假设第二种情况预习了10天）：

|  |  |
| --- | --- |
|  |  |
|  | print(func(2,66)) |
|  | print(func(0,10)) |

分别输出0.8098858536030598;0.23045287195467073。

但这并不意味着如果你平时学只能考81分，考前预习只能考23分。。。因为记忆量和成绩虽然呈正相关但肯定不一样的，考试还有很多其它因素，比如重点与非重点，步骤分，平时分，师生情什么的。而且听课的效率和自学预习的效率不一样，当然不能确定，因为有些人自习效率非常高，膜拜他们。

而且很多大神考前预习一遍再复习一遍，效果拔群。当然那肯定考完试马上就忘了，只能应付考试而已。

所以如果真正想掌握一门知识，还是平时好好学比较好。

不说了，我要复习去了。。

**三、分析的缺陷**

1、在幂函数拟合公式中自变量没有乘线型系数，而且直接用的-1次幂函数，都对结果有影响，虽然参数简单降低了分析难度但也会降低准确度。

2、考虑的因素比较少，忽略了考虑考前突击对心态、健康的影响。默认平常学习和考前学习效率是一样的。

3、在复习时间中取了平均值，实际上不论怎么学习都有一个阶段性的过程，不是点式的，这必将导致不同章节最后掌握结果不同，但这又需要一个axis了，我hold不了…

**四、其它**

这个学期才开始学习Python和数学建模，当然要感谢老师的讲课和ppt…

所有代码在我的github上：<https://github.com/chemgeeklian/homework01>

Thanks.

**参考文献**

[1]百度百科

[2] 于洪, 李转运. 基于遗忘曲线的协同过滤推荐算法[J]. 南京大学学报：自然科学版, 2010, 46(5):520-527.

[3] <http://blog.itpub.net/12199764/viewspace-1743145/>