文本词汇丰富度分析

一、 实验目的

类符与形符之比（type token ratio，简称 TTR）即一个文本中所有的类符（type）与所有的形符（token）的比值，其中形符指一个文本中所有的词数，类符指不重复计算的形符数，意思是在一个文本中，重复出现的形符只能算作一个类符。

通过计算文本的 TTR 值来判断一个文本的复杂程度，如果一篇文章用了很多不同的单词来写作，那么 TTR 的值就会接近于 1，说明这篇文章的用词比较复杂，词汇丰富度较高，阅读难度较大，反之，词汇丰富度越低，阅读难度越小。

二、 实验名称：

文本词汇丰富度分析（Text vocabulary richness analysis）

三、 实验材料

任一需要处理的文本或者语料库。

四、 实验时长

两课时

目录

一、 实验背景

采用概率法对一篇文章的词汇进行统计和量化分析，从中寻找语言的规律，从而对语言本身进行更加客观的描写和解释，这也便于我们了解一篇文章的文学词汇特征或者写作特征。其中一个重要的分析因素是词频分析，即计算文本的类符与形符比（Type-Token Ratio，TTR）。

形符（token）是一个语言单位，指相邻空格之间的连续字符串，即一个单词，一个文本的形符数就是该文本的长度，文本一共有多少个词，就有多少个形符。

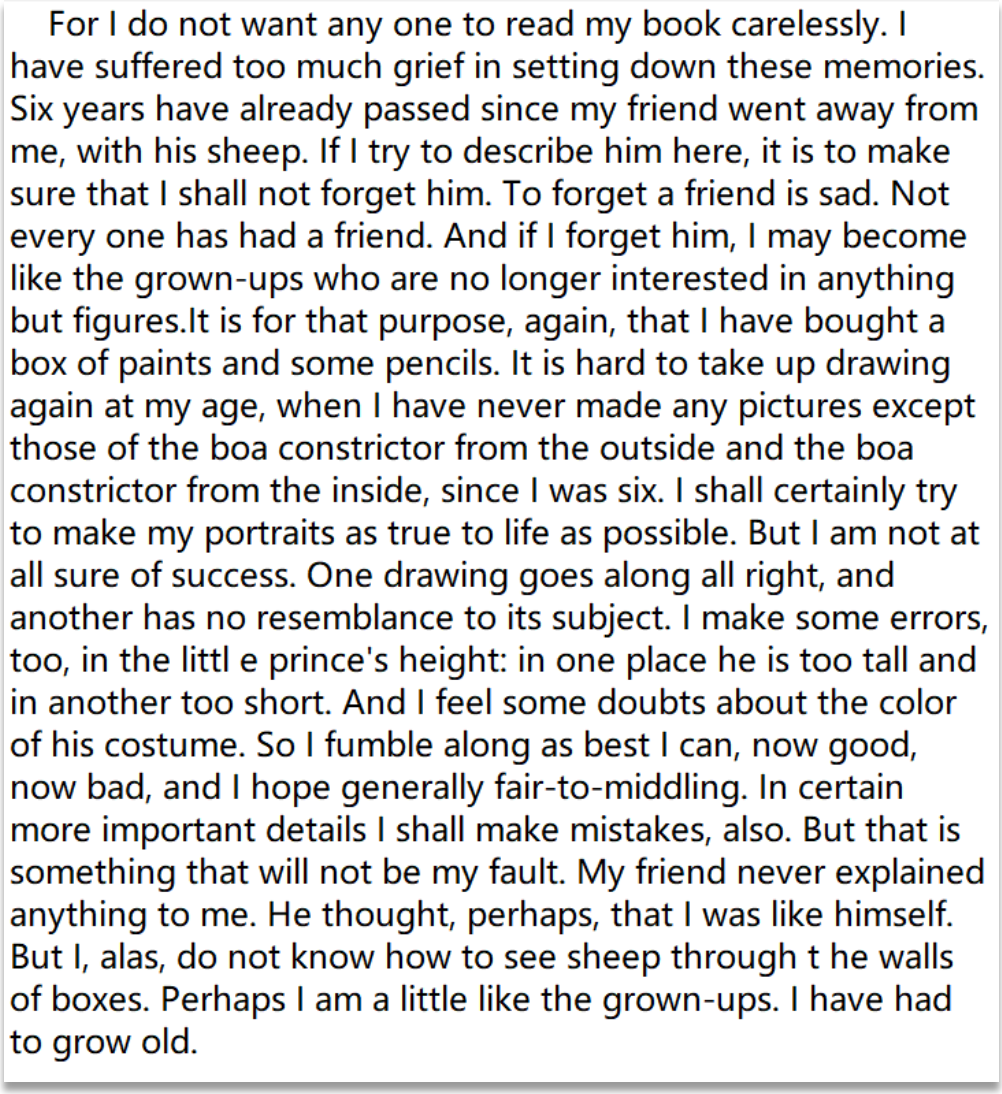
类符（type）是一个统计单位，指要分析的文本中任何一个独特的词形，在词频统计中相同的形符称为类符，一个文本的类符数就是该文本不同词形的数量。

类符/形符比（TTR）用来衡量文本中词汇的复杂度，即用所有不重复的词的个数除以词的总数的值，计算公式如下：

$$TTR = \frac{\text{Amount of unique words}}{\text{Amount of total words}}$$

二、实验数据

将需要处理的文本数据以 txt 文件形式存放，内容可以是任何需要处理的文本内容，在此节选了小说《小王子》中的一段文字进行词汇复杂度的分析，内容如图一所示：



图一　示例文本

三、关键代码说明

1、读取文本信息

这个实验的目的是可以处理任一文本的词汇复杂度，而不是仅仅处理特定的某一个文本，因此将需要处理的文本数据通过 cmd 命令窗口发送，使程序可以读取，读取到的文本路径存放在变量 filePath 中，代码如图二所示：

```
import sys
filePath = sys.argv[1]
```

图二 读取文本路径

2、计算形符数

创建两个列表对读取到的数据进行分词处理，其中列表 list1 中暂时存放当前进行分词的句子，分词之后的句子依次存放到列表 list2 中，所以 list2 最终存放的是分词之后的所有数据，计算 list2 的长度并将结果存放在 token_count 中，这个值就是该文本的形符数，即文本的长度。

这部分的代码如图三所示，list2 和 token_count 中数据存放情况如图四所示：

```
for f in open(filePath,'r').readlines():
    list1=f.strip().split(' ')
    list2+=list1
token_count = len(list2)
```

图三 计算形符数

```
list2中的数据是:
['For', 'I', 'do', 'not', 'want', 'any', 'one', 'to', 'read', 'my', 'book', 'carelessly.', 'I',
'have', 'suffered', 'too', 'much', 'grief', 'in', 'setting', 'down', 'these',
'memories.', 'Six', 'years', 'have', 'already', 'passed', 'since', 'my', 'friend',
'went', 'away', 'from', 'me,', 'with', 'his', 'sheep.', 'If', 'I', 'try', 'to', 'describe',
'him', 'here,', 'it', 'is', 'to', 'make', 'sure', 'that', 'I', 'shall', 'not', 'forget', 'him.',
'To', 'forget', 'a', 'friend', 'is', 'sad.', 'Not', 'every', 'one', 'has', 'had', 'a', 'friend.',
'And', 'if', 'I', 'forget', 'him,', 'I', 'may', 'become', 'like', 'the', 'grown-ups', 'who',
'are', 'no', 'longer', 'interested', 'in', 'anything', 'but', 'figures.It', 'is', 'for', 'that',
'purpose,', 'again,', 'that', 'I', 'have', 'bought', 'a', 'box', 'of', 'paints', 'and',
'some', 'pencils.', 'It', 'is', 'hard', 'to', 'take', 'up', 'drawing', 'again', 'at', 'my',
'age,', 'when', 'I', 'have', 'never', 'made', 'any', 'pictures', 'except', 'those', 'of',
'the', 'boa', 'constrictor', 'from', 'the', 'outside', 'and', 'the', 'boa', 'constrictor',
'from', 'the', 'inside,', 'since', 'I', 'was', 'six.', 'I', 'shall', 'certainly', 'try', 'to',
'make', 'my', 'portraits', 'as', 'true', 'to', 'life', 'as', 'possible.', 'But', 'I', 'am', 'not',
'at', 'all', 'sure', 'of', 'success.', 'One', 'drawing', 'goes', 'along', 'all', 'right,',
'and', 'another', 'has', 'no', 'resemblance', 'to', 'its', 'subject.', 'I', 'make', 'some',
'errors,', 'too,', 'in', 'the', 'littl', 'e', "prince's", 'height:', 'in', 'one', 'place', 'he',
'is', 'too', 'tall', 'and', 'in', 'another', 'too', 'short.', 'And', 'I', 'feel', 'some',
'doubts', 'about', 'the', 'color', 'of', 'his', 'costume.', 'So', 'I', 'fumble', 'along',
'as', 'best', 'I', 'can,', 'now', 'good,', 'now', 'bad,', 'and', 'I', 'hope', 'generally',
'fair-to-middling.', 'In', 'certain', 'more', 'important', 'details', 'I', 'shall', 'make',
'mistakes,', 'also.', 'But', 'that', 'is', 'something', 'that', 'will', 'not', 'be', 'my',
'fault.', 'My', 'friend', 'never', 'explained', 'anything', 'to', 'me.', 'He', 'thought,',
'perhaps,', 'that', 'I', 'was', 'like', 'himself.', 'But', 'I,', 'alas,', 'do', 'not', 'know',
'how', 'to', 'see', 'sheep', 'through', 't', 'he', 'walls', 'of', 'boxes.', 'Perhaps', 'I',
'am', 'a', 'little', 'like', 'the', 'grown-ups.', 'I', 'have', 'had', 'to', 'grow', 'old.', '']

token_count =  297
```

图四 list2 和 token_count 中数据存放情况

3、计算类符数

使用 set ( ) 函数对上一步中分词之后的数据进行去重处理，去掉重复的单词，列表 list3 中存放的是去重之后的结果，计算 list3 的长度并存放在 type_count 中，这个值就是该文本的类符数，即不重复的单词个数，代码如图五所示，list3 和 type_count 中数据存放情况如图六所示：

```
list3 = set(list2)
type_count = len(list3)
```

图五 计算类符数

```
list3中的数据是:
{'', 'sure', 'some', 'friend', 'make', 'He', 'is', 'as', 'forget', 'the',
'generally', 'book', 'memories.', 'old.', 'am', 'bought', 'To',
'My', 'these', 'It', 'more', 'Six', 'except', 'down', 'now',
'mistakes,', 'through', 'possible.', 'pencils.', 'setting',
'suffered', 'errors,', 'are', 'made', 'again,', 'But', 'longer', 'not',
'carelessly.', 'how', 'color', 'and', 'went', 'anything', 'fault.',
'little', 'resemblance', 'shall', 'to', 'take', 'hope', 'him.',
'thought,', 'constrictor', 'fumble', 'outside', 'himself.', 'right,',
'feel', 'portraits', 'no', 'something', 'too', 'from', 'drawing', 'e',
'fair-to-middling.', 'it', 'him,', 'have', 'want', 'sheep.', 'If',
'explained', 'tall', 'In', 'if', 'who', 'figures.It', 'also.', 'a',
'success.', 'much', 'true', 'best', 'another', 'inside,', 'has',
'walls', 'box', 'costume.', 'had', 'here,', 'bad,', 'grief', 'of',
'with', 't', 'For', 'any', 'certainly', 'never', "prince's", 'friend.',
'I', 'grown-ups', 'passed', 'life', 'in', 'at', 'doubts', 'try', 'littl',
'boxes.', 'will', 'away', 'may', 'Perhaps', 'him', 'one',
'perhaps,', 'alas,', 'One', 'subject.', 'that', 'certain', 'years',
'details', 'but', 'boa', 'So', 'be', 'up', 'short.', 'good,',
'purpose,', 'interested', 'my', 'all', 'those', 'sheep', 'describe',
'age,', 'me.', 'read', 'for', 'he', 'become', 'sad.', 'I,', 'goes',
'can,', 'pictures', 'know', 'height:', 'its', 'see', 'And', 'along',
'important', 'was', 'six.', 'do', 'place', 'Not', 'me,', 'every',
'when', 'his', 'grow', 'like', 'since', 'paints', 'too,', 'about',
'grown-ups.', 'hard', 'already', 'again'}

type_count =  179
```

图六 list3 和 type_count 中数据存放情况

4、计算 TTR

通过上两步得到的形符数和类符数，套用上文中 TTR 的公式来计算文本的类符/形符比(TTR),最后计算结果保留两位小数 ,计算部分的代码如图七所示，计算结果如图八所示：

```
ratio = float(type_count / token_count)*100
print('Type Token Ratio: {:.2f}%'.format(ratio))
```

图七 计算 TTR

```
Type Token Ratio: 60.27%
```

图八 TTR 的计算结果

四、怎样运行代码

在 cmd 窗口进行操作：

1、在刚刚打开 cmd 时路径默认的是 C 盘用户所在的位置，因此需要首先进入 TTR.py 文件所在的位置，如图九：

```
F:\python\code\TTR>
```

图九

2、然后输入以下命令：python TTR.py (文件名).txt

在本次实验中将要处理的文本命名为 1.txt，因此输入 python TTR.py 1.txt，得到的结果如图十所示：

```
F:\python\code\TTR>python TTR.py 1.txt
Type Token Ratio: 60.27%
```

图十

即可得到此篇文本的类符/形符比，即 TTR 值。

五、扩充内容

STTR 的简介

1、STTR 的定义

在多次实验中发现，简单计算 TTR 会受到文本大小的直接影响，导致分析结果不准确，例如在一篇 300 词的文本里，TTR 值可能为 60%，在一篇 1000 词的文本里，TTR 值可能为 40%，但是在 100 万词甚至更大的文本里，TTR 值则可能只有 4%。这

样的结果显然不具备普适性，需要进行改进。

为了使 TTR 值具有可比性，使计算结果更加标准，引入了标准类符/形符比的概念，即 STTR。STTR(standard type-token ratio）是对 TTR 的计算进行标准化，方法是先计算每 n 个词的 TTR 值之后，再对得到的所有 TTR 取平均值，最终得到的结果就是该文本的 STTR 值，计算公式如下，其中 j 是文本根据 n 个词为一组进行分割得到的组数。

$$STTR = \sum_{i=0}^{j} TTRi/\mathrm{j}$$

2、STTR 和 TTR 的比较

STTR 反映了文本用词的变化性和多样性，并且得到的计算结果比 TTR 更加可观和准确，STTR 值越高，说明文本中使用的词形越多，因此文本的词汇丰富度也越高。当处理的文本词汇过多时，STTR 与 TTR 相比，可以调整 n 的取值来得到更加可靠的结果。

3、计算本次实验中文本的 STTR 值

设置以 n=100 个词为一组分别计算 TTR 值，那么这个文本就可以分为三组，通过计算可以得到第一组的 TTR 值为 77%，第二组的 TTR 值为 71%，第三组的 TTR 值为 82.47%，所以可得到这个文本的 STTR 值为（77%+71%+82.47%）/3=76.82%。这个值与之前计算的整个文本的 TTR 值相比更加能够反应一个文本或者语料库的词汇特性。

六、附录

完整代码

```
import sys
filePath = sys.argv[1]

list1 = []
list2 = [] #存放分词后的所有数据
list3 = [] #存放去重后的数据

for f in open(filePath,'r').readlines():
    list1=f.strip().split(' ')
    list2+=list1

token_count = len(list2)

list3 = set(list2)
type_count = len(list3)

ratio = float(type_count / token_count)*100
print('Type Token Ratio: {:.2f}%'.format(ratio))
```