一种基于Python的国际中文教育水平等级词汇统计方法

薛原　王子勋　龙擎天　何志永
天津科技大学 人工智能学院

摘　要：随着全球对中文学习需求的增加，以及数字化技术的不断发展，传统的教学模式已经无法满足学生的需求。基于国际中文教育创新项目——“中国人的一天”故事化、场景化、交互式汉语听说数字化教材开发及编写项目，需要一款词汇统计软件，旨在能够通过预处理文本、提取词汇并将其归类到不同级别的字典中，最终得出各级词汇在文本中的出现次数、所占百分比的词汇统计方法。用来检测教材脚本中各级中文词汇量，进而评估教材质量。本文主要介绍开发国际中文教育水平等级词汇统计软件功能的技术路线、算法设计、实现过程，及发现的问题的解决方案。

关键词：国际中文教育；词汇统计；滑动窗口
【DOI】10.12252/j.issn.2096-6288.2024.06.033

一、项目背景

该项目来源于国际中文教育创新项目：“中国人的一天”，故事化、场景化、交互式汉语听说数字化教材开发及编写。随着全球对中文学习需求的增加，以及数字化技术的不断发展，传统的教学模式已经无法满足学生的需求。随着数字化教材开发与编写，其教材脚本是否符合教学需求，尤其是脚本所涉及的词汇是否在大纲范围内急需统计。由于词汇量巨大，传统的人工统计很难实现。因此，急需开发一款国际中文教育水平等级词汇统计软件，来解决这个问题。我们的统计软件基于《国际中文教育中文水平等级标准》（国家标准•应用解读本）中的词汇表，共九级11092个词汇，对脚本中每个词汇、分级统计出现次数、百分比等。

二、技术路线

（一）开发语言Python

如今随着国际影响力的提升，国际中文教育的热度升高，国际中文教育急需提高数字化，项目选择python进行开发看重其高效性、可扩展性，为后期在程序中加入其他功能提供保障，因此项目选择使用Python进行开发。

（二）数据结构-字典

程序开发过程中需要将读取的文本与词汇库进行匹配，其中词汇库分为初等词汇（1-3级）、中等词汇（4-6级）、高等词汇（7-9级），需要合适的数据结构来存储词汇。同时，字典是一种可变容器模型，并且可以储存任意类型的对象。

基本结构为：dict={key1：value1，key2：value2 }。

在字典中所有键值对都是独一无二的，这样就确保了所有中文词汇都能准确地逐一与读取文段进行匹配。并且字典由哈希表实现，可以进行快速的数据检索，无论字典的内容有多大都不会减慢字典查找的速度。

（三）滑动窗口

在Python中，滑动窗口是一种用于解决字符串或数组相关问题的常见技巧。滑动窗口通常用于在一个大的数据结构（如字符串或数组）中查找一个子串或子数组，以及在给定约束条件下找到满足条件的子串或子数组。

在字符串匹配中，滑动窗口算法通常具有较低的时间复杂度，因为它只需要遍历一次整个字符串或数组。并且通常只需要存储窗口的起始索引和结束索引，而不需要额外的空间。同时，可以根据问题的需求进行调整，可以选择增大或缩小窗口的大小，以及调整窗口的移动步长。

（四）语音识别

在实现文本识别的基础下，调用开源的语音识别PaddlePaddle的API，向程序中加入语音识别的方法，可提升程序的处理能力，这样不仅仅具备基本的文本识别能力，还同时支持更便捷的语音识别功能，提升方案的灵活性、易用性。

三、实现过程

（一）算法概要

1.算法实现目的

该算法的目的是对使用者上传的文本文件中的中文内容进行词汇拆解，并对所拆词汇进行分级统计，计算出各级词汇在整个内容中所占比例。

2.算法实现优势

该算法可以灵活适应不同级别词汇的统计需求，通过定义各级词汇字典的方式，实现这一功能。可以根据需要轻松扩展到更多级别的词汇统计，只需对子字典进行更新即可，方便实时更新。并且通过对文本的预处理

以及对滑动窗口的使用，有效地避免了重复遍历文本以及词汇的重复统计，提高了效率。

3.算法复杂度分析

假设文本长度为 n，词汇级别字典的总大小为m。“while i<len（s）”循环在最坏情况下需要执行n次。“for j in range（4，0，-1）”循环则最多需要循环4次。另外，遍历字典操作的时间复杂度为0（m）。因此，总体时间复杂度为 0（n*m）。假设存储各级词汇的字典总大小为m。额外空间消耗较小，主要由循环变量和临时变量所需空间决定，可以忽略不计。因此，总体空间复杂度为0（m）。

（二）算法实现过程

1.数据预处理

我们需要读取用户上传的文本文件，并对其进行预处理。其中包括去除文本中的标点符号和空格，以便于后续的词汇拆解与统计。

2.字典创建

依据《国际中文教育中文水平等级标准》的教材规定，我们创建了七级字典用于分级存储所有标准中文词汇，后续与拆解词汇的匹配统计。

3.词汇统计

我们选择以滑动窗口的方式提取连续的字符串片段作为所需拆解词汇。之后我们将每个拆解出的词汇与创建的词汇字典进行匹配比较，若匹配成功，则字典中的相应词汇的值加一，表示该词在文本中出现了一次。此外，我们创建一个信标，用于记录每一次成功的词汇拆解。

4.统计百分比

统计各级词汇在整个文本文件中的出现次数，该项数据在词汇统计中通过字典进行了存储，则计算时，只需读用其数据即可。然后将各级词汇的出现次数与所有词汇的出现次数分别进行除法运算，即可算出各级词汇在该文本文件中占有的内容百分比。

5.结果输出

为了让用户更加方便直观地阅览想要的数据结果，我们将算法结果输出至对应的CSV文件中，可以方便用户进行进一步的数据分析和可视化展示。

（三）关键代码

1.拆解信标的建立

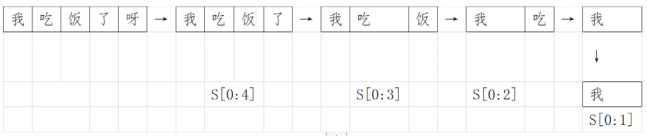
在我们正式进行词汇拆解操作之前，应该思考一个问题，我们该如何让计算机做到“适可而止”，即当计算机拆解出一个正确词汇时，立即停止进一步的拆解工作，而不是将已经正确的词汇进行二次拆解，反而变成我们不需要的词汇。例如：计算机拆解出“算法”一词后，下一步操作是将“算法”一词与字典匹配，匹配成功后将字典中的“算法”一词的值加一，并继续拆解“算法”之外的词汇。而不是将其继续拆解为“算”和“法”，导致统计操作时的数据重复。因此我们设置拆解信标“flag”，用于记录拆解的成功与否，flag=0时则继续进行拆解操作。一旦拆解词汇与字典匹配成功，则flag变为1，退出此次操作，进行下一次拆解。

2.滑动窗口的创建与使用

由于《国际中文教育中文水平等级标准》中的词汇规定最大字数为4，因此我们需要从四字词汇开始逐字递减进行词汇拆解匹配。因此我们创建滑动窗口s[i：i+j]对文本切片操作。其中i用于表示其起始位置（初始值为0），i+j用于表示滑动窗口的结束位置（初始值为4，但切片结束位置不包括索引4）。j表示滑动窗口的长度，它用range（4，0，-1）函数进行递减操作，从4开始，0结束（不包括0），即从四字词汇开始，每次减少一个字，直至单字。

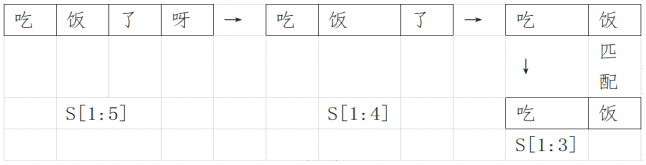
以“我吃饭了呀”这句话为例：

（1）



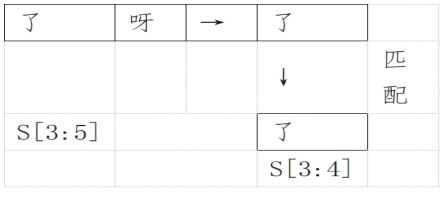
“我”匹配成功，则滑动窗口起始位置i变为1，对后四个文字进行下一步词汇拆解。

（2）

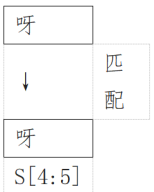


“吃饭”匹配成功，则滑动窗口起始位置i变为3，因为整个文本只有五个字符，因此下一步只对后两个字符进行下一步词汇拆解。

（3）



（4）



最终拆解出“我”，“吃饭”，“了”，“呀”四个词汇。

四、问题及解决方案

（一）问题发现

在软件测试的过程中，我们发现了循环遍历字符截取问题。

原始代码：

```
for i in range(0, len(s)):
    flag = 0
    for j in range(4, 0, -1):
        if flag == 1:
            break
        s_word = s[i:i + j]
        for k_s1 in dict_s1.keys():
            if s_word == k_s1:
                dict_s1[k_s1] = dict_s1[k_s1] + 1
                flag = 1
        if flag == 1:
            continue
```

在原始代码中，我们的应用逻辑是依次以每一个字符为起点，然后在每个字符位置上，从长度4递减到1，提取不同长度的字符序列（词汇）并检查它是否出现在字典当中。“for i in range（0，len（s））” 这一行代码实现了外层循环，i从 0 开始，逐步增加直到字符串s的长度。这样就能保证每个字符都被遍历到。“for j in range（4，0，-1）”内层循环用于逐步减小步长，汉语词汇最多有4个汉字，所以我们从4开始逐步减小到 1，以便截取长度不同的子字符串。

“s_word = s[i：i + j]”这一行代码截取了当前位置i开始的长度为j的子字符串。

内层循环中，代码通过遍历每个字典的键来检查截取的子字符串是否与某个键匹配。如果匹配成功，则相应字典中对应键的值加1，并且将flag置为1，表示匹配成功，并且内层循环可以提前结束。如果匹配失败，则继续检查下一个字典的键，直到所有字典的键都被检查过。然而，当我们从每一个字符依次遍历时，发现会出现统计重复冗余的情况。例：当字符串是“怒火中烧，四分五裂”时，“怒火中烧”是一个四字词汇，当识别到它是词典内的词汇时，跳出内层循环，从“火”字开始遍历，但是“火”字同样是一个单字词汇，这样就会出现重复统计的情况。根据实际应用情况我们应该当判断到“怒火中烧”是一个四字词汇时，下一次循环遍历应跳过这四个字。

（二）解决方案

解决方案代码：

```
while i<len(s):
    flag=0
    for j in range(4,0,-1):
        if flag==1:
            break
        s_word=s[i:i+j]
        print(s_word)
        for k_s1 in dict_s1.keys():
            if s_word==k_s1:
                count=len(s_word)
                i=i+count-1
                dict_s1[k_s1]=dict_s1[k_s1]+1
                flag=1
    if flag==1:
        continue
```

我们项目组进行了集中讨论，为了解决这个问题，我们引入一个机制来确保在识别到一个词汇后，下一次循环遍历从该词汇的末尾位置之后开始，而不是从当前字符的下一个位置开始，以下是修改后的代码详细解释：

1.主循环

“while i < len（s）：”这个循环用于遍历整个字符串s，由于需要遍历的次数是不确定的，在这里我们使用while循环，根据条件动态地控制循环的终止。

2.内层循环

“for j in range（4，0，-1）：”这个循环用于逐步截取不同长度的子字符串，从长度为 4 的子字符串开始，逐步减小长度到 1。

3.截取子字符串

“s_word=s[i：i+j]”截取从当前位置 `i` 开始的长度为 `j` 的子字符串。

4.匹配子字符串并统计词频

对截取的子字符串s_word，分别在每个字典的键中查找是否存在该子字符串。如果匹配成功，将相应字典中对应键的值加1，并将flag置为1，表示匹配成功。如果匹配失败，则继续检查下一个字典的键。

5.更新索引和循环控制

如果匹配到一个词汇，通过“i=i+len（s_word）-1”将索引i移动到当前词汇的末尾位置。如果flag被置为1，表示已经匹配到了一个词汇，跳过这个词汇的后续字符的检查。通过这个机制，在识别到一个词汇后，确保下一次循环遍历从该词汇的末尾位置之后开始，而不是从当前字符的下一个位置开始，避免了对词汇的重复统计。

参考文献

[1]Vaibhhav Khetapal.Python 中的滑动窗口[EB/OL].delftstack.com/zh/howto/python/sliding-window-python/，2022-5-18

[2]祝永志，荆静.基于Python语言的中文分词技术的研究[J].通信技术，2019，52（07）：1612-1619.

[3]杨菲菲.Python之数据结构分析及教学[J].电脑知识与技术，2019，15（30）：130-132.

[4]裘宗燕著.重点大学计算机教材[M]北京：机械工业出版社，2022.