《自然语言处理》实验报告

一、实验目的

1. 理解并掌握最大匹配算法，并将其应用于实际情景；

2. 理解双向最大匹配算法消除歧义。

二、实验项目内容

使用前向、后向算法实现一个简单的中文分词器（必做）。对可能的分词歧义，利用语言模型进行消歧（该功能可选做）。

要求：
（1）提交电子文档一份（word），内含两部分内容：程序源码文本，程序运行结果截图；
（2）同时提交源程序文件（可采用任意语言开发）。

报告创建时间：2022.11.11

三、实验过程或算法（源程序）

1. 最大匹配算法描述

汉语中文本是基于单字的，词与词之间没有显性的界限标志，因此分词是汉语文本分析处理中首先要解决的问题。但是汉语分词时常面临一些难点：歧义切分问题、未登录词识别问题。

最大匹配算法（MM）可以有效实现分词，分为正向最大匹配算法（FMM）、逆向最大匹配算法（BMM）和双向最大匹配算法（BiDMM）。

1.1 正向最大匹配算法

假设词库中最长词条长度为 l，则正向最大匹配算法过程如下：

（1）取样本未分割部分的前 l 个字（不足 l 个就取到尾），进入（2）；

（2）在词库中查找当前字段：

若词库中存在，则匹配成功，转（1）；

否则转（3）；

（3）只要当前字段长度大于 1，就去掉当前字段的最后一个字，然后转（2）；否则当前字作为一个分词，转（1）。

执行上述过程直至分词结束，得到的句子就是成功分词的结果。

1.2 逆向最大匹配算法

逆向最大匹配算法过程与正向相似，只是从句尾开始，匹配失败时去掉开头汉字：

（1）取样本未分割部分的后 l 个字（不足 l 个就取到开头），进入（2）；

（2）在词库中查找当前字段：

若词库中存在，则匹配成功，转（1）；

否则转（3）；

（3）只要当前字段长度大于 1，就去掉当前字段的第一个字，然后转（2）；否则当前字作为一个分词，转（1）。

执行上述过程直至分词结束，得到的句子就是成功分词的结果。实验表明，逆向最大匹配算法比正向最大匹配算法具有更好的分词效果。

1.3 双向匹配算法

双向匹配算法是对正向最大匹配算法和逆向最大匹配算法的结合，选取两个划分结果中分词数量较少的结果，因为这往往更可能满足正确的分词效果。如果分词数量也相同，就返回单字较少的结果。

双向匹配算法可以识别出分词中的交叉歧义，往往具有良好的分词效果。**本实验使用双向最大匹配算法进行中文分词**。

2. 实验过程

本实验使用**双向最大匹配算法**进行中文分词，实验流程为：读取语料库->正向最大匹配算法->逆向最大匹配算法->选取分词数量较少的作为结果->分词数相同时选择单字较少的作为结果

2.1 读取语料库

读取读取 SogouLabDic.dic 文件，得到词库和最长词条长度。该文件格式如下：

```
我们    770027797   PRON,
时间    767969294   N,
中国    727787725   N,ADJ,
可以    685520165   ADJ,AUX,
```

因此每次读取一行，用 tab 分割词条和频数、词性。本实验只用到词条，对该字段进行存储即可。因为语料库中含有重复词条，因此使用集合存储。

由于 FMM、BMM 算法均需要使用词库的最长词条长度，因此在构建词库时便记录最长词条长度 l：

```python
def getLexicon():
    '''读取SogouLabDic.dic文件,得到词库和最长词条长度'''
    lexicon=set()                           #词库
    path=r'NLP\data\SogouLabDic.dic'
    I=0                                     #最长词条所含汉字数
    with open(path,'r') as f:
        while True:
            line=f.readline()
            if not line:                    #读取结束
                break
            line=line.split('    ')         #分割为词组、频数和词性
            word=line[0]                    #词组
            I=max(I,len(word))
            lexicon.add(word)
    return lexicon,I
```

2.2 正向最大匹配算法

对测试样本从前向后分词，具体思路见 1.1。代码如下：

```
def FMM(lexicon,I,sentence):
    '''使用FMM算法对样本sentence进行分词'''
    splt_sent=[]                          #存储划分成词的句子
    ptr1=0                                #指向sentence正在查找字段的指针
    ptr2=min(I,len(sentence))
    while(ptr1<len(sentence)):
        if((ptr1+1)==ptr2):               #只剩一个字
            splt_sent.append(sentence[ptr1])
            ptr1=ptr2
            ptr2=min(ptr1+I,len(sentence))
            continue
        word=sentence[ptr1:ptr2]
        if word in lexicon:               #匹配成功
            splt_sent.append(word)
            ptr1=ptr2                     #更新指针
            ptr2=min(ptr1+I,len(sentence))
        else:                             #匹配失败
            ptr2-=1
    return splt_sent
```

2.3 逆向最大匹配算法

对测试样本从后向前分词，具体思路见 1.2。代码如下：

```
def BMM(lexicon,I,sentence):
    '''使用BMM算法对样本sentence进行分词'''
    splt_sent=[]                          #存储划分成词的句子
    ptr1=len(sentence)-1                  #指向sentence正在查找字段的指针
    ptr2=max(len(sentence)-I,-1)
    while(ptr1>-1):
        if((ptr1-1)==ptr2):               #只剩一个字
            splt_sent.insert(0,sentence[ptr1])
            ptr1=ptr2
            ptr2=max(ptr1-I,-1)
            continue
        word=sentence[ptr2+1:ptr1+1]
        if word in lexicon:               #匹配成功
            splt_sent.insert(0,word)
            ptr1=ptr2                     #更新指针
            ptr2=max(ptr1-I,-1)
        else:                             #匹配失败
            ptr2+=1
    return splt_sent
```

2.4 双向最大匹配算法选择结果

选取两个划分结果中分词数量较少的作为最终结果，如果分词数量相同则返回单字较少的结果：

```
def Disambiguate(lst1,lst2):
    '''分词消歧启发式规则,选择分词数量较少的作为结果;如果一样,选择单字较少的作为结果'''
    if(len(lst1)>len(lst2)):
        return lst2
    elif(len(lst1)<len(lst2)):
        return lst1
    else:
        cnt1=0
        cnt2=0
        for i in range(len(lst1)):
            if(len(lst1[i])==1):
                cnt1+=1
            if(len(lst2[i])==1):
                cnt2+=1
        if(cnt1<cnt2):
            return lst1
        else:
            return lst2
```

2.5 循环读入并进行分词

使用while循环，不断待分词的句子，输出**双向最大匹配算法**分词后的结果：

```
if __name__ == '__main__':
    lexicon,I=getLexicon()                  #该词库中最长词条长度为14
    while(True):
        print("Please input a Chinese sentence:")
        sentence=str(input())
        print("分词结果:")
        fmm=FMM(lexicon,I,sentence)
        bmm=BMM(lexicon,I,sentence)
        print(Lst2Str(Disambiguate(fmm,bmm)))
```

其中Lst2Str()函数是将分词结果由列表转换为“/”间隔的字符串，详见附录代码。

四、实验结果及分析和（或）源程序调试过程

实验结果：

如图所示，每输入一句话，就可以输出双向最大匹配算法的分词结果，以“/”为分词间隔：

```
Please input a Chinese sentence:
中国人为了实现自己的梦想
分词结果:
中国人/为了/实现/自己的/梦想
Please input a Chinese sentence:
部分居民生活水平
分词结果:
部分/居民/生活水平
Please input a Chinese sentence:
门把手弄坏了
分词结果:
门把手/弄坏/了
Please input a Chinese sentence:
```

可以看到，该算法的分词效果较好，可以对教学过程的歧义语句实现较好的分词。更多样例如下：

```
Please input a Chinese sentence:
他还兼任何应钦在福州办的东路军军官学校的政治教官
分词结果:
他还/兼任/何应钦/在/福州/办/的/东路/军/军官/学校/的/政治/教官
Please input a Chinese sentence:
大不列颠及北爱尔兰联合王国外交和英联邦事务大臣
分词结果:
大不列颠/及/北爱尔兰/联合/王国/外交/和/英联邦/事务/大臣
Please input a Chinese sentence:
他是研究生物化学的一位科学家
分词结果:
他是/研究/生物化学/的/一位/科学家
Please input a Chinese sentence:
他只会诊断一般的疾病
分词结果:
他/只会/诊断/一般/的/疾病
Please input a Chinese sentence:
重庆大学是一所双一流高校
分词结果:
重庆大学/是一所/双/一流/高校
Please input a Chinese sentence:
自然语言处理真有趣
分词结果:
自然语言/处理/真/有趣
```

模型对于常见的分词情况较为准确，但遇到专有名词后者人名、地名时还是会出现误分词的现象：

```
Please input a Chinese sentence:
特朗普是美国的前任总统
分词结果：
特/朗/普/是/美国/的/前任/总统
Please input a Chinese sentence:
博尔特是世界上跑得最快的男人
分词结果：
博/尔特/是世界上/跑得/最快/的/男人
Please input a Chinese sentence:
新冠疫情对人类的影响很大
分词结果：
新/冠/疫情/对/人类/的/影响/很大
```

调试过程：

1. 问题：读取 SogouLabDic.dic 文件编码错误：

```
  File "D:\KSoftware\Miniconda\Miniconda3\lib\codecs.py", line 322, in decode
    (result, consumed) = self._buffer_decode(data, self.errors, final)
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xb8 in position 2: invalid start byte
```

解决方法：中文字符的 Unicode 编码 0x0800-0xFFFF 之间，而 utf-8 只包含了部分汉字，因此将中文字符转成 utf-8 时超出了其范筹，需要用 Unicode 打开：

```
with open(path,'r',encoding='unicode_escape') as f:
```

其实还有一种办法，就是将 SogouLabDic.dic 文件转换编码，后续读取也不会有问题，本实验就是采用后者。

2. 问题：FMM 和 BMM 算法中 while 容易陷入死循环：

解决方法：使用双指针进行检索。

实验感悟：

最大匹配算法较为简单，在分词方面有较好的效果。结合 FMM 和 BMM 的双向最大匹配算法结合了二者的优越性，可以在分词过程中一定程度上消除语句歧义问题。但对于一些专有名词和人名、地名，分词效果欠佳。该遗憾可以通过增大词库规模来解决，或者采用词性识别等更加高级的算法减少误分词的可能性。

五、附录（完整代码）

```
'''双向最大匹配 BiDMM 实现中文分词
    语料库：'NLP\data\SogouLabDic.dic'
    算法：正向匹配算法->逆向匹配算法->选取分词数量较少的作为结果->分词数相同时选择单字较少的作为结果
    思路：读取语料库->FMM->BMM->BiDMM
'''

def getLexicon():
    '''读取 SogouLabDic.dic 文件,得到词库和最长词条长度'''
    lexicon=set()                          #词库
    path=r'NLP\data\SogouLabDic.dic'
    I=0                                    #最长词条所含汉字数
    with open(path,'r') as f:
        while True:
            line=f.readline()
            if not line:                   #读取结束
                break
            line=line.split('   ')         #分割为词组、频数和词性
            word=line[0]                   #词组
            I=max(I,len(word))
            lexicon.add(word)
    return lexicon,I

def FMM(lexicon,I,sentence):
    '''使用 FMM 算法对样本 sentence 进行分词'''
    splt_sent=[]                           #存储划分成词的句子
    ptr1=0                                 #指向 sentence 正在查找字段的指针
    ptr2=min(I,len(sentence))
    while(ptr1<len(sentence)):
        if((ptr1+1)==ptr2):                #只剩一个字
            splt_sent.append(sentence[ptr1])
            ptr1=ptr2
            ptr2=min(ptr1+I,len(sentence))
            continue
        word=sentence[ptr1:ptr2]
        if word in lexicon:                #匹配成功
            splt_sent.append(word)
```

```
            ptr1=ptr2                   #更新指针
            ptr2=min(ptr1+I,len(sentence))
        else:                           #匹配失败
            ptr2-=1
    return splt_sent

def BMM(lexicon,I,sentence):
    '''使用 BMM 算法对样本 sentence 进行分词'''
    splt_sent=[]                        #存储划分成词的句子
    ptr1=len(sentence)-1                #指向 sentence 正在查找字段的指针
    ptr2=max(len(sentence)-I,-1)
    while(ptr1>-1):
        if((ptr1-1)==ptr2):             #只剩一个字
            splt_sent.insert(0,sentence[ptr1])
            ptr1=ptr2
            ptr2=max(ptr1-I,-1)
            continue
        word=sentence[ptr2+1:ptr1+1]
        if word in lexicon:             #匹配成功
            splt_sent.insert(0,word)
            ptr1=ptr2                   #更新指针
            ptr2=max(ptr1-I,-1)
        else:                           #匹配失败
            ptr2+=1
    return splt_sent

def Lst2Str(lst):
    '''将列表分词转换成"/"间隔的字符串'''
    s_out=''                            #分词后语句,'/'间隔
    for i in range(len(lst)):
        if i>0:
            s_out+='/'
        s_out+=lst[i]
    return s_out

def Disambiguate(lst1,lst2):
    '''分词消歧启发式规则,选择分词数量较少的作为结果;如果一样,选择单字较少的作为结果
'''
    if(len(lst1)>len(lst2)):
```

```
        return lst2
    elif(len(lst1)<len(lst2)):
        return lst1
    else:
        cnt1=0
        cnt2=0
        for i in range(len(lst1)):
            if(len(lst1[i])==1):
                cnt1+=1
            if(len(lst2[i])==1):
                cnt2+=1
        if(cnt1<cnt2):
            return lst1
        else:
            return lst2

if __name__ == '__main__':
    lexicon,I=getLexicon()              #该词库中最长词条长度为 14
    while(True):
        print("Please input a Chinese sentence:")
        sentence=str(input())
        print("分词结果:")
        fmm=FMM(lexicon,I,sentence)
        bmm=BMM(lexicon,I,sentence)
        print(Lst2Str(Disambiguate(fmm,bmm)))
```