第三次作业

1.程序关键步骤

```python
def backward_maximum_matching(text, word_dict):
    result = []
    max_dict_length = max(len(word) for word in word_dict)
    max_length = min(len(text), max_dict_length)
    while text:
        # 遍历长度，从最长的词开始匹配
        for i in range(max_length, 0, -1):
            word = text[-i:]
            if word in word_dict:
                result.insert(0, word)
                text = text[:-i]
                break
        else: # 遍历完所有长度，没有匹配到词，则把最后一个字母加到结果中
            result.insert(0, text[-1])
            text = text[:-1]

    return result
```

评估并计算正确率、召回率、F-测度

```python
def evaluate(predicted, target):
    predicted_set = set(predicted)
    target_set = set(target)

    true_positive = len(predicted_set & target_set)
    precision = true_positive / len(predicted_set) if predicted_set else 0 # 预测正确的词数 / 预测词数
    recall = true_positive / len(target_set) if target_set else 0 # 预测正确的词数 / 目标词数
    f_measure = (2 * precision * recall) / (precision + recall) if (precision + recall) != 0 else 0

    return precision, recall, f_measure
```

程序测试结果：其中（“王五”不在词表中）

```
PS D:\研究生\研一上课程\01-11下午自然语言处理\3> & D:/ProgramFilesFolder/05-Anaconda3/python.exe d:/研究生/研一上课程/01-11下午自然语言处理/3/reverse_max_matching.py
程序分词结果: ['我', '喜欢', '吃', '香蕉']
准确率: 1.00, 召回率: 1.00, F-测度: 1.00
程序分词结果: ['我', '喜欢', '吃饭']
准确率: 1.00, 召回率: 1.00, F-测度: 1.00
程序分词结果: ['王', '五', '喜欢', '吃饭']
准确率: 0.50, 召回率: 0.67, F-测度: 0.57
```

2.设计并实现一个汉语未登录词（汉族人名）的识别算法(可限定条件)，并通过实验分析该算法的优缺点。

(1) 算法设计

a. 核心思想

基于规则：利用中文姓氏库和名字特征规则，确定潜在人名的边界。

概率模型：通过统计姓氏和名字字符的联合分布，计算人名的概率值，并设置阈值判断。

修饰规则：使用上下文特征校正识别结果，减少误识别。

b. 算法流程

构建姓氏库：从百家姓和其他数据来源收集常见姓氏。

分词处理：先用逆向最大匹配（BMM）对句子进行分词，识别潜在的未登录词。

在分词的基础上选择人名识别触发点：

利用姓氏库作为触发点，从分词结果中寻找可能是姓的词。

对姓后的 1~2 个字进行组合，判断是否可能是名字，进行有界范围的框定。

计算频率：

姓氏在文本中的出现频率。

姓名首字和尾字的联合分布概率。

识别潜在人名：

匹配姓氏作为触发点，结合后续字符判断是否构成姓名。

(2) 算法实现

```python
# 人名识别函数
def recognize_names(text, word_dict, threshold=0.0001):
    segmented = backward_maximum_matching(text, word_dict)
    potential_names = []
    for i, word in enumerate(segmented):
        if word in surnames:  # 姓氏触发点
            # 寻找潜在名字
            if i + 1 < len(segmented):  # 至少需要两个字
                m1 = segmented[i + 1]
                m2 = segmented[i + 2] if i + 2 < len(segmented) and len(segmented[i + 2]) == 1 else "" # 可能有第三个字
                if len(m1) + len(m2) >= 1 and len(m1) + len(m2) <=2:  # 有界范围
                    prob = calculate_name_probability(word, m1, m2)
                    if prob > surname_threshold[word]:  # 判断是否满足概率条件
                        potential_names.append((word + m1 + m2, prob))
    # 去重和规则修正
    final_names = []
    for name, prob in potential_names:
        if all([
            not name.isdigit(),  # 修饰规则：避免数字
            name not in word_dict,  # 避免已有词典中的词
        ]):
            final_names.append(name)
    return segmented, final_names
```

(3) 实验结果

```
PS D:\研究生\研一上课程\01-11下午自然语言处理\3> & D:/ProgramFilesFolder/05-Anaconda3/python.exe d:/研究生/研一上课程/01-11下午自然语言处理/3/name.py
==================================人名识别测试==================================
输入文本：张建国正在演讲。
分词结果：['张', '建', '国', '正', '在', '演', '讲', '。']
识别的人名：['张建国']

输入文本：李国庆发表了意见。
分词结果：['李', '国', '庆', '发', '表', '了', '意', '见', '。']
识别的人名：['李国庆']

输入文本：南京市长江大桥是著名景点。
分词结果：['南', '京', '市', '长', '江', '大', '桥', '是', '著名', '景点', '。']
识别的人名：['江大桥']
```

(4) 优缺点分析

优点:：

触发机制明确。

资源消耗少，轻量，适用于功能嵌入。

缺点：

依赖分词结果：由于词典本身内容不够大，没有包含“长江”这样的词汇，所以会将“长”和“江”分开，然后将“江”作为姓氏。（第三个测试样例）

语料依赖性强：需要足够大的语料库统计支持人名概率的估计。

复姓支持有限：当前算法对复姓和特殊名字支持较弱。

上下文信息缺失：未充分利用上下文信息来识别更复杂的未登录词，如词性等信息。

(5) 优化方向

边界校正：

利用上下文规则校正人名范围，剔除冲突识别结果。

添加修饰规则：

如标点、数字附近的否定规则。

3.

3.1 BiGram

<EOS>李彬　阅读　了　一封　信　。<EOS>

<EOS>郑新喜　阅读　了　一本　有趣　的　书　。<EOS>

<EOS>柳英　阅读　了　李悦　的　一本　书　。<EOS>

李彬		李彬　阅读	
阅读	3	阅读　了	3
了	3	了　一封	
一封		一封　信	
信		郑新喜　阅读	
郑新喜		了　一本	
一本	2	一本　有趣	
有趣		有趣　的	
的		的　书	
书	2	柳英　阅读	
柳英		了　李悦	
李悦		李悦　的	
的		的　一本	
。	3	一本　书	
<EOS>	6	信　。	
		书　。	2
		<EOS> 郑新喜	1
		。<EOS>	3

对于句子：

郑新喜　阅读　了　一封　信　。

首先添加标记：

<EOS>郑新喜　阅读　了　一封　信　。<EOS>

P(郑新喜|<EOS>)	Count(郑新喜 <EOS>) / Count(<EOS>)	1/6
P(阅读|郑新喜)	Count(阅读 郑新喜) / Count(郑新喜)	1/1 = 1
P(了|阅读}	Count(了 阅读) / Count(阅读)	3/3 = 1
P(一封|了)	Count(一封 了) / Count(了)	1/3
P(信|一封)	Count(信 一封) / Count(一封)	1/1 = 1
P(。|信)	Count(。 信) / Count(信)	1/1 = 1
P(<EOS>|。)	Count(<EOS> 。) / Count(。)	3/3 = 1

答案：

P(Sentence) = P(郑新喜|<EOS>) * P(阅读|郑新喜) * P(了|阅读} * P(一封|了) * P(信|一封) * P(。|信) * P(<EOS>|。) =

1/18

PS:若不添加标记则为 1/3，题目中没有进行强制要求

3.2 信息熵

遥	1	下	1	童	1	声	1	河	2
远	1	面	1	年	1	随	1	水	2
的	13	是	2	阿	2	风	1	流	2
夜	1	那	4	娇	2	飘	2	啊	1
空	1	小	4	摇	1	到	1	进	1
有	2	桥	2	着	3	我	2	心	1
一	2	旁	1	唱	1	脸	2		
个	1	边	2	古	1	上	3		
弯	14	条	1	老	1	淌	1		
月	2	船	3	歌	2	泪	1		
亮	2	悠	2	谣	1	像	1		
	40		23		16		16		9

一共 104 字符

计算信息熵：

信息熵公式：

$$H(X) = -\sum_{x \in X} p(x) \log_2 p(x)$$

计算结果为：

5.10742

程序见 h_cal.py

4. 请写出语句“In the classroom, he cleaned the desk with a dishcloth.”的格框架表示。

主语动词：cleaned

主要概念：cleaned，对 clean 进行了形态变化

辅助概念：

施事格（Agent）：he

动作的执行者是 "he"。

受事格（Patient）：the desk

动作的对象或影响的目标是 "the desk"。

处所格（Location）：in the classroom

动作发生的地点是 "in the classroom"。

工具格（Instrument）：with a dishcloth

动作使用的工具是 "a dishcloth"。

格类型	句中元素	语义角色
施事格	he	执行动作的人
受事格	the desk	被清理的对象
处所格	in the classroom	动作发生的地点
工具格	with a dishcloth	用来执行动作的工具