**1.程序源码**

import math

from collections import Counter

import os

import re

import time

def \_parse\_people\_daily\_line(line\_text):

    """

    解析人民日报语料库格式的单行文本。

    例如: "本报/rz 北京/ns ... [24/m 个/q]/mq ... 。/w"

    返回一个词语列表。

    """

    words = []

    # 将多个空格替换为单个空格，然后按空格分割

    cleaned\_line = re.sub(r'\s+', ' ', line\_text.strip())

    tokens = cleaned\_line.split(' ')

    for token in tokens:

        token = token.strip()

        if not token:

            continue

        # 检查形如 "[词1/词性1 词2/词性2]/短语词性" 的模式

        # 例如: "[24/m 个/q]/mq"

        match\_bracketed\_phrase = re.match(r'\[(.\*?)\]/([a-zA-Z0-9\_]+)$', token)

        if match\_bracketed\_phrase:

            phrase\_content = match\_bracketed\_phrase.group(1)

            # phrase\_content 类似于 "词1/词性1 词2/词性2"

            inner\_tokens = phrase\_content.split(' ')

            for inner\_token in inner\_tokens:

                inner\_token = inner\_token.strip()

                if inner\_token:

                    # 从内部词块中移除词性标记

                    word\_only = inner\_token.rsplit('/', 1)[0] if '/' in inner\_token else inner\_token

                    words.append(word\_only)

        else:

            # 普通的 "词/词性" 或可能只是一个单独的词（例如标点符号）

            # 移除词性标记

            word\_only = token.rsplit('/', 1)[0] if '/' in token else token

            words.append(word\_only)

    return words

def load\_corpus\_from\_directory(base\_dir\_path):

    """

    从指定目录加载语料库。

    遍历目录及其子目录下的所有 .txt 文件，解析它们的内容。

    :param base\_dir\_path: 语料库的根目录路径 (str)

    :return: 一个句子列表，其中每个句子是词的列表 (list of list of str)

    """

    corpus\_sentences = []

    # 将相对路径转换为绝对路径，以便os.path.isdir等函数能正确工作

    # \_\_file\_\_ 是当前脚本的路径

    # os.path.dirname(\_\_file\_\_) 是当前脚本所在的目录

    # 如果base\_dir\_path已经是绝对路径，os.path.abspath会返回它本身

    # 如果base\_dir\_path是相对路径，它会相对于当前工作目录解析。

    if not os.path.isabs(base\_dir\_path):

        script\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))

        base\_dir\_path = os.path.join(script\_dir, base\_dir\_path)

    if not os.path.isdir(base\_dir\_path):

        print(f"错误: 目录 '{base\_dir\_path}' 未找到。")

        return corpus\_sentences

    print(f"开始从目录 '{base\_dir\_path}' 加载语料库...")

    file\_count = 0

    sentence\_count = 0

    for root, \_, files in os.walk(base\_dir\_path):

        for file\_name in files:

            if file\_name.endswith(".txt"):

                file\_path = os.path.join(root, file\_name)

                file\_count += 1

                try:

                    # 尝试使用 UTF-8 编码打开文件。

                    with open(file\_path, 'r', encoding='utf-8', errors='ignore') as f:

                        for line in f:

                            stripped\_line = line.strip()

                            if not stripped\_line: # 跳过空行

                                continue

                            if re.match(r'^\d{8}-\d{2}-\d{3}-\d{3}$', stripped\_line): # 跳过ID行

                                continue

                            if stripped\_line.startswith(""): # 跳过特殊标记行

                                continue

                            processed\_words = \_parse\_people\_daily\_line(stripped\_line)

                            if processed\_words:

                                corpus\_sentences.append(processed\_words)

                                sentence\_count +=1

                except Exception as e:

                    print(f"处理文件 {file\_path} 时发生错误: {e}")

    print(f"语料库加载操作完成。共处理 {file\_count} 个文件，提取了 {sentence\_count} 个句子。")

    if sentence\_count == 0 and file\_count > 0:

        print(f"注意: 从 {file\_count} 个文件中未能提取任何有效句子。")

    return corpus\_sentences

class BigramDisambiguator:

    def \_\_init\_\_(self, corpus\_sentences):

        """

        初始化消歧器

        :param corpus\_sentences: 一个列表，其中每个元素是一个已经分好词的句子 (词列表)

        """

        self.corpus\_sentences = corpus\_sentences

        self.unigram\_counts = Counter()

        self.bigram\_counts = Counter()

        self.total\_words = 0

        self.vocab\_size = 0

        self.\_train()

    def \_train(self):

        """

        根据提供的语料库训练模型，计算unigram和bigram的频率

        """

        word\_set = set()

        if not self.corpus\_sentences:

            print("错误: 语料库为空，模型训练中止。")

            return

        print("开始训练模型...")

        for sentence in self.corpus\_sentences:

            if not sentence:

                continue

            padded\_sentence = sentence

            for i in range(len(padded\_sentence)):

                word = padded\_sentence[i]

                self.unigram\_counts[word] += 1

                word\_set.add(word)

                if i > 0:

                    prev\_word = padded\_sentence[i-1]

                    self.bigram\_counts[(prev\_word, word)] += 1

        self.total\_words = sum(self.unigram\_counts.values())

        self.vocab\_size = len(word\_set)

        if self.total\_words > 0:

            print(f"模型训练完成。词汇表大小: {self.vocab\_size}, 总词数: {self.total\_words}。")

        else:

            print("模型训练失败: 未能从语料库中提取有效词汇。")

    def get\_log\_probability(self, word\_sequence):

        """

        计算给定词序列的对数概率（使用Add-One平滑）

        :param word\_sequence: 一个词列表

        :return: 该序列的对数概率 (float)

        """

        if not word\_sequence:

            return -float('inf')

        if self.vocab\_size == 0 or self.total\_words == 0:

            return -float('inf')

        log\_prob = 0.0

        first\_word = word\_sequence[0]

        # 对于非常大的词汇表，total\_words + vocab\_size 可能会很大，导致 prob\_w1 非常小

        # 但对于 Add-One 平滑，这是预期的行为

        prob\_w1\_numerator = self.unigram\_counts.get(first\_word, 0) + 1

        prob\_w1\_denominator = self.total\_words + self.vocab\_size

        if prob\_w1\_denominator == 0: # 避免除以零

            prob\_w1 = 1e-20 # 赋一个极小值

        else:

            prob\_w1 = prob\_w1\_numerator / prob\_w1\_denominator

        log\_prob += math.log(prob\_w1 if prob\_w1 > 0 else 1e-20)

        for i in range(1, len(word\_sequence)):

            prev\_word = word\_sequence[i-1]

            current\_word = word\_sequence[i]

            bigram\_count = self.bigram\_counts.get((prev\_word, current\_word), 0)

            prev\_word\_unigram\_count = self.unigram\_counts.get(prev\_word, 0)

            denominator = prev\_word\_unigram\_count + self.vocab\_size

            if denominator == 0: # 避免除以零，理论上 vocab\_size > 0

                 conditional\_prob = 1 / self.vocab\_size if self.vocab\_size > 0 else 1e-20

            else:

                conditional\_prob = (bigram\_count + 1) / denominator

            log\_prob += math.log(conditional\_prob if conditional\_prob > 0 else 1e-20)

        return log\_prob

    def disambiguate(self, candidate\_segmentations):

        """

        从多个候选分词结果中选择最可能的一个

        :param candidate\_segmentations: 候选分词结果列表

        :return: (最佳分词结果, 最佳结果的对数概率, 所有结果及概率的列表)

        """

        if not candidate\_segmentations:

            return None, -float('inf'), []

        if self.vocab\_size == 0 or self.total\_words == 0:

            return candidate\_segmentations[0] if candidate\_segmentations else None, \

                   -float('inf'), \

                   [{"segmentation": seg, "log\_probability": -float('inf')} for seg in candidate\_segmentations]

        best\_segmentation = None

        max\_log\_prob = -float('inf')

        results\_with\_probs = []

        for seg in candidate\_segmentations:

            log\_p = self.get\_log\_probability(seg)

            results\_with\_probs.append({"segmentation": seg, "log\_probability": log\_p})

            if log\_p > max\_log\_prob:

                max\_log\_prob = log\_p

                best\_segmentation = seg

        if best\_segmentation is None and candidate\_segmentations:

            best\_segmentation = candidate\_segmentations[0]

        return best\_segmentation, max\_log\_prob, results\_with\_probs

# --- 主程序 ---

if \_\_name\_\_ == "\_\_main\_\_":

    corpus\_base\_path = os.path.join("people-2014", "train") # 这里默认语料库和程序在同一目录下

    print(f"指定语料库相对路径: '{corpus\_base\_path}' (将解析为相对于脚本位置或当前工作目录)")

    processed\_corpus = load\_corpus\_from\_directory(corpus\_base\_path)

    if not processed\_corpus:

        print(f"错误: 未能从路径 '{corpus\_base\_path}' (或其解析后的绝对路径) 加载语料数据。程序将退出。")

        exit()

    disambiguator = BigramDisambiguator(processed\_corpus)

    test\_cases = [

        {

            "original\_sentence": "我从小学电脑",

            "candidates": [

                ["我", "从", "小学", "电脑"],

                ["我", "从小", "学", "电脑"]

            ]

        },

        {

            "original\_sentence": "他喜欢研究生物化学",

            "candidates": [

                ["他", "喜欢", "研究", "生物", "化学"],

                ["他", "喜欢", "研究", "生物化学"],

                ["他", "喜欢", "研究生物", "化学"]

            ]

        },

        {

            "original\_sentence": "在北京大学生活区喝进口红酒",

            "candidates": [

                ["在", "北京大学", "生活区", "喝", "进口", "红酒"],

                ["在", "北京", "大学", "生活区", "喝", "进口", "红酒"],

                ["在", "北京大学", "生活", "区", "喝", "进口", "红酒"]

            ]

        },

        {

            "original\_sentence": "羽毛球拍卖完了 (作业示例)",

            "candidates": [

                ["羽毛球", "拍卖", "完", "了"],

                ["羽毛", "球", "拍卖", "完", "了"] ,

                ["羽毛球拍", "卖完", "了"]

            ]

        }

    ]

    if disambiguator.vocab\_size > 0 and disambiguator.total\_words > 0 :

        print("\n开始进行分词歧义消解测试...")

        for i, case in enumerate(test\_cases):

            print(f"\n测试用例 {i+1}: \"{case['original\_sentence']}\"")

            best\_seg, best\_log\_prob, all\_results = disambiguator.disambiguate(case['candidates'])

            print("  候选分词及对数概率:")

            for res in all\_results:

                seg\_str = " \\ ".join(res['segmentation'])

                log\_prob\_display = f"{res['log\_probability']:.4f}" if res['log\_probability'] > -float('inf') else "-inf"

                print(f"    - \"{seg\_str}\": {log\_prob\_display}")

            if best\_seg:

                best\_seg\_str = " \\ ".join(best\_seg)

                best\_log\_prob\_display = f"{best\_log\_prob:.4f}" if best\_log\_prob > -float('inf') else "-inf"

                print(f"  预测最佳分词: \"{best\_seg\_str}\" (对数概率: {best\_log\_prob\_display})")

            else:

                print("  未能确定最佳分词结果。")

    else:

        print("\n模型未成功训练或语料库为空，无法执行分词歧义消除测试。")

**2.程序运行结果截图**



