自然语言处理大作业一报告

07152001 班 史桠彬 1120201198

一. 实验要求

1. 实现基于词典的分词方法和统计分词方法：两类方法中实现一种即可；
2. 对分词结果进行词性标注，也可以在分词的同时进行词性标注；
3. 对分词及词性标注结果进行评价，包括 4 个指标：正确率、召回率、F1 值和效率。

二. 实现环境

ASUS VivoBook + Win10 + Pycharm 2021.2.3 + Python 3.9 + Anaconda 3.7

三. 实验内容

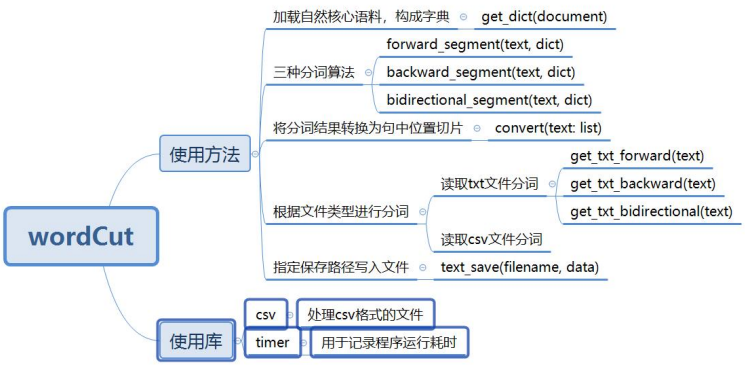
首先，基于 HanLP 自然核心词典，我们使用正向最长匹配、逆向最长匹配及双向最长匹配三种方法，对网络文章及人民日报语料两份素材进行分词（提供了对 txt 和 csv 两种保存格式的素材的分词接口），并结合作业二中对网络文章的手工分词结果及人民日报语料的已有分词结果进行比较，计算三种分词方法的 Precise、Recall 及 F-measure 值，进而评价分词的效果，使用计时器获得分词时间，进而评价分词的效率。

接着，基于人民日报词性标注语料库，我们根据前边得到的人民日报语料的分词结果，使用 Viterbi 算法对其进行词性标注，并结合语料库中手工词性标注的结果进行比较，计算由三种分词方法得到的分词结果对应的词性标注的 Precise、Recall 及 F-measure 值，进而评价预料标注的效果，使用计时器获得词性标注时间，进而评价分词的效率。

四. 实验过程

4.1 分词

4.1.1 程序结构



4.1.2 算法设计

本代码的核心算法是三种分词算法，实现思路见代码注释。

- 正向最长匹配：

```
def forward_segment(text, dict):
    word_list = []
    i = 0
    while i < len(text):
        longest_word = text[i]                          # 记录当前扫描位置的单字
        for j in range(i + 1, len(text) + 1):           # 对于所有可能的结尾
            word = text[i:j]                            # 记录从当前位置到结尾的连续字符串
            if word in dict:                             # 判断单词是否在词典中
                if len(word) > len(longest_word):       # 判断是否该串更长，若是则优先输出
                    longest_word = word
        word_list.append(longest_word)                  # 输出最长词，并进行正向扫描
        i += len(longest_word)
    return word_list
```

- 逆向最长匹配：

```
def backward_segment(text, dict):
    word_list = []
    i = len(text) - 1
    while i >= 0:                                       # 扫描位置作为终点
        longest_word = text[i]                          # 扫描位置的单字
        for j in range(0, i):                           # 遍历[0, i]区间作为待查询词语的起点
            word = text[j: i + 1]                       # 取出[j, i]区间作为待查询单词
            if word in dict:
                if len(word) > len(longest_word):       # 越长优先级越高
                    longest_word = word
                    break
        word_list.insert(0, longest_word)               # 逆向扫描，所以越先查出的单词在位置上越靠后
        i -= len(longest_word)
    return word_list
```

- 双向最长匹配：

```
def bidirectional_segment(text, dict):                                                                    ▲42 ✓15
    forward = forward_segment(text, dict)
    backward = backward_segment(text, dict)
    # 词数更少更优先
    if len(forward) < len(backward):
        return forward
    elif len(forward) > len(backward):
        return backward
    else:
        # 单字更少更优先
        if sum(1 for word in forward if len(word) == 1) < sum(1 for word in backward if len(word) == 1):
            return forward
        else:
            return backward
```

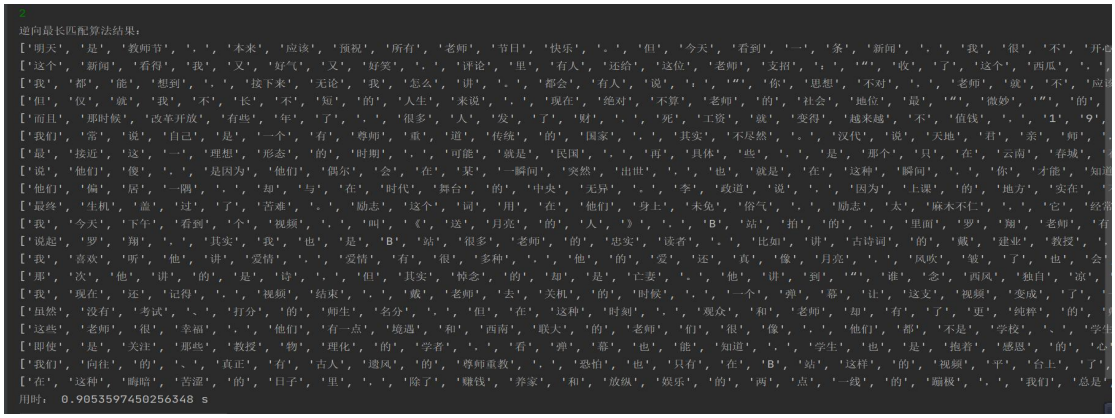
4.1.3 程序运行及结果

在检验分词算法的效果和效率时，我们选用了两份素材，对于网络文章，各算法的运行结果如下：

yuxiuhua.txt：

- 正向最长匹配：

- 逆向最长匹配：



- 双向最长匹配：

算法类型	用时
正向最长匹配	0.706s
逆向最长匹配	0.905s
双向最长匹配	0.930s

yuxiuhua_wordcut.csv：

- 正向最长匹配：

```
F:\BigHomework1\venv\Scripts\python.exe F:/BigHomework1/wordcut.py
yuxiuhua_wordcut.csv
请输入分词方法，其中1为正向最长分词，2为逆向最长分词，3为双向最长分词：
1
该方法的分词精度为： 0.8362778918262075
该方法的分词召回率为： 0.8853680760251
该方法的分词F值为： 0.860123116987363
用时： 0.8305871486663818 s

Process finished with exit code 0
```

- 逆向最长匹配：

```
F:\BigHomework1\venv\Scripts\python.exe F:/BigHomework1/wordcut.py
yuxiuhua_wordcut.csv
请输入分词方法，其中1为正向最长分词，2为逆向最长分词，3为双向最长分词：
2
该方法的分词精度为： 0.8370185521320852
该方法的分词召回率为： 0.8867032924628143
该方法的分词F值为： 0.8611448631985085
用时： 0.9121100902557373 s

Process finished with exit code 0
```

- 双向最长匹配：

```
F:\BigHomework1\venv\Scripts\python.exe F:/BigHomework1/wordcut.py
yuxiuhua_wordcut.csv
请输入分词方法，其中1为正向最长分词，2为逆向最长分词，3为双向最长分词：
3
该方法的分词精度为： 0.8355065837083521
该方法的分词召回率为： 0.884573322532978
该方法的分词F值为： 0.8593401179414538
用时： 1.0903453826904297 s

Process finished with exit code 0
```

算法类型	P 值	R 值	F 值	用时
正向最长匹配	0.836	0.885	0.860	0.831s
逆向最长匹配	0.837	0.887	0.861	0.912s
双向最长匹配	0.836	0.885	0.859	1.090s

corpus.csv：

- 正向最长匹配：

```
F:\BigHomework1\venv\Scripts\python.exe F:/BigHomework1/wordcut.py
corpus.csv
请输入分词方法，其中1为正向最长分词，2为逆向最长分词，3为双向最长分词：
1
该方法的分词精度为： 0.8213103781045212
该方法的分词召回率为： 0.8454091409142171
该方法的分词F值为： 0.8331855399234308
用时： 17.236299753189087 s

Process finished with exit code 0
```

- 逆向最长匹配：

```
F:\BigHomework1\venv\Scripts\python.exe F:/BigHomework1/wordcut.py
corpus.csv
请输入分词方法，其中1为正向最长分词，2为逆向最长分词，3为双向最长分词：
2
该方法的分词精度为： 0.8276577926968515
该方法的分词召回率为： 0.8517078823997086
该方法的分词F值为： 0.8395106276409047
用时： 17.455381155014038 s

Process finished with exit code 0
```

- 双向最长匹配：

```
F:\BigHomework1\venv\Scripts\python.exe F:/BigHomework1/wordcut.py
corpus.csv
请输入分词方法，其中1为正向最长分词，2为逆向最长分词，3为双向最长分词：
3
该方法的分词精度为： 0.8292029552162836
该方法的分词召回率为： 0.8528643340158862
该方法的分词F值为： 0.8408672241493528
用时： 32.23924469947815 s

Process finished with exit code 0
```

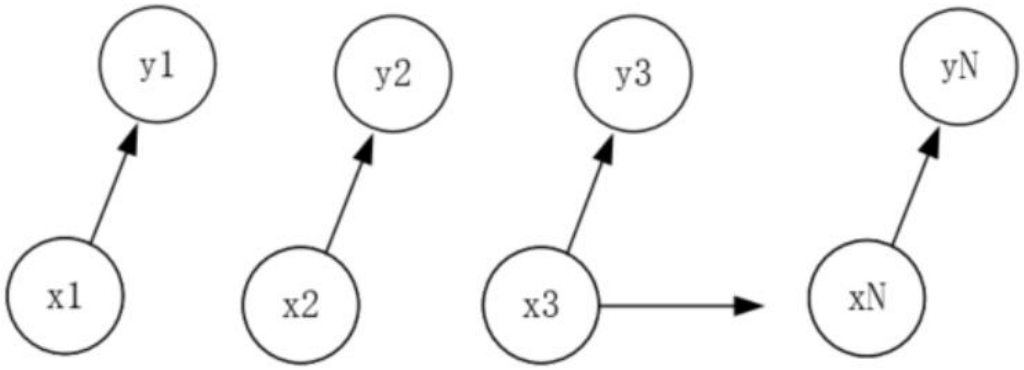
算法类型	P 值	R 值	F 值	用时
正向最长匹配	0.821	0.845	0.833	17.236s
逆向最长匹配	0.828	0.852	0.840	17.455s
双向最长匹配	0.829	0.853	0.841	32.239s

4.2 词性标注

4.2.1 程序结构

4.2.2 算法设计

本代码的核心算法是 Viterbi 算法，实现思路见代码注释。



适用维特比算法的隐含马尔可夫模型

```
def Viterbi(text,text_percent,trans,dic):
    dict_len = len(dic)
    dis = [dict() for _ in range(len(text))]
    node = [dict() for _ in range(len(text))]
    for first in text_percent[0].keys():
        dis[0][first] = 1
    for i in range(len(text) - 1):
        word_one = text[i]
        word_two = text[i + 1]
        word_one_percent_dict = text_percent[i]
        word_two_percent_dict = text_percent[i + 1]

        word_one_percent_key = list(word_one_percent_dict.keys())
        word_one_percent_value = list(word_one_percent_dict.values())
        word_two_percent_key = list(word_two_percent_dict.keys())
        word_two_percent_value = list(word_two_percent_dict.values())
        for word_two_per in word_two_percent_key:
            tmp_dis = []
            for word_one_per in word_one_percent_key:
                if word_two_per in trans[word_one_per]:
                    tmp_num = dis[i][word_one_per] * (
                            (trans[word_one_per][word_two_per] + 1) / (dic[word_one_per] + dict_len)) * (
                                    text_percent[i + 1][word_two_per] / dic[word_two_per])
```

```
                tmp_dis.append(tmp_num)
            else:
                tmp_num = dis[i][word_one_per] * (1 / (dic[word_one_per] + dict_len)) * (
                        text_percent[i + 1][word_two_per] / dic[word_two_per])
                tmp_dis.append(tmp_num)

        max_tmp_dis = max(tmp_dis)
        max_tmp_dis_loc = tmp_dis.index(max_tmp_dis)
        dis[i + 1][word_two_per] = max_tmp_dis
        node[i + 1][word_two_per] = word_one_percent_key[max_tmp_dis_loc]

path = []
f_value = list(dis[len(dis) - 1].values())
f_key = list(dis[len(dis) - 1].keys())
f = f_key[f_value.index(max(f_value))]
path.append(f)
for i in range(len(dis) - 1, 0, -1):
    f = node[i][f]
    path.append(f)
path.reverse()
return path
```

4.2.3 程序运行及结果

fortrain_forward.txt：

```
第 8483 行: ['在', '村口', '.', '王', '三', '运', '说', ':', '“', '大年初一', '把', '同志们', '叫', '到', '这里', '来', '.', '是', '要', '给', '大家', '留下',
[{'p': 11390, 'vt': 356, 'd': 315}, {'s': 6}, {'wd': 74496}, {'nrf': 1000, 'n': 10}, {'m': 1019, 'nrg': 2}, {'vt': 38, 'Ng': 1, 'nrg': 3}, {'vt': 253
算法词性标注结果： ['在/p', '村口/s', '. /wd', '王/nrf', '三/nrg', '运/vt', '说/vt', ': /wm', '“/wyz', '大年初一/t', '把/p', '同志们/n', '叫/vt', '到/vq', '这里/
人工词性标注结果 ['在/p', '村口/s', '. /wd', '王/nrf', '三运/nrg', '说/vt', ': /wm', '“/wyz', '大年初一/t', '把/p', '同志/n', '们/k', '叫/vt', '到/vq', '这里/rz'
预测成功数： 36
预测成功率： 0.8780487804878049
召回成功数： 36
召回成功率： 0.8571428571428571

该方法的分词精度为： 0.7543061002314114
该方法的分词召回率为： 0.7721566093558201
该方法的分词F值为： 0.7631269825498658
用时： 73.0882728099823 s

Process finished with exit code 0
```

fortrain_backward.txt：

```
第 8483 行: ['在', '村口', '.', '王', '三', '运', '说', ':', '“', '大年初一', '把', '同志们', '叫', '到', '这里', '来', '.', '是', '要', '给', '大家', '留下',
[{'p': 11390, 'vt': 356, 'd': 315}, {'s': 6}, {'wd': 74496}, {'nrf': 1000, 'n': 10}, {'m': 1019, 'nrg': 2}, {'vt': 38, 'Ng': 1, 'nrg': 3}, {'vt': 253
算法词性标注结果： ['在/p', '村口/s', '. /wd', '王/nrf', '三/nrg', '运/vt', '说/vt', ': /wm', '“/wyz', '大年初一/t', '把/p', '同志们/n', '叫/vt', '到/vq', '这里/
人工词性标注结果 ['在/p', '村口/s', '. /wd', '王/nrf', '三运/nrg', '说/vt', ': /wm', '“/wyz', '大年初一/t', '把/p', '同志/n', '们/k', '叫/vt', '到/vq', '这里/rz'
预测成功数： 36
预测成功率： 0.8780487804878049
召回成功数： 36
召回成功率： 0.8571428571428571

该方法的分词精度为： 0.760132020099037
该方法的分词召回率为： 0.7779016304815407
该方法的分词F值为： 0.7689141750481998
用时： 69.16646957397461 s

Process finished with exit code 0
```

fortrain_bidirectional.txt：

```
第 8483 行: ['在', '村口', '.', '王', '三', '运', '说', ':', '“', '大年初一', '把', '同志们', '叫', '到', '这里', '来', '.', '是', '要', '给', '大家', '留下',
[{'p': 11390, 'vt': 356, 'd': 315}, {'s': 6}, {'wd': 74496}, {'nrf': 1000, 'n': 10}, {'m': 1019, 'nrg': 2}, {'vt': 38, 'Ng': 1, 'nrg': 3}, {'vt': 253
算法词性标注结果： ['在/p', '村口/s', '. /wd', '王/nrf', '三/nrg', '运/vt', '说/vt', ': /wm', '“/wyz', '大年初一/t', '把/p', '同志们/n', '叫/vt', '到/vq', '这里/
人工词性标注结果 ['在/p', '村口/s', '. /wd', '王/nrf', '三运/nrg', '说/vt', ': /wm', '“/wyz', '大年初一/t', '把/p', '同志/n', '们/k', '叫/vt', '到/vq', '这里/rz'
预测成功数： 36
预测成功率： 0.8780487804878049
召回成功数： 36
召回成功率： 0.8571428571428571

该方法的分词精度为： 0.760132020099037
该方法的分词召回率为： 0.7779016304815407
该方法的分词F值为： 0.7689141750481998
用时： 68.09205293655396 s

Process finished with exit code 0
```

算法类型	P 值	R 值	F 值	用时
正向最长匹配	0.754	0.772	0.763	75.088s
逆向最长匹配	0.760	0.778	0.769	69.166s
双向最长匹配	0.760	0.778	0.769	68.092s

五．实验结论

最终，基于本次实验，我们得出结论，就分词效果而言：正向最长匹配 ＜ 逆向最长匹配 ≈ 双向最长匹配,就分词效率而言:正向最长匹配 ≈ 逆向最长匹配 ＞ 双向最长匹配;就词性标注效果而言：正向最长匹配 ＜ 逆向最长匹配 ≈ 双向最长匹配，就词性标注效率而言，正向最长匹配 ＜ 逆向最长匹配 ＜ 双向最长匹配。此外，我们发现，处理 txt 文件比 csv 文件的效率更高。

六．实验中遇到的问题及解决办法

1. 对于含有较多数字、符号的语料素材(如包括时间的新闻、包括运算式的数学论文等)，手工分词及算法分词的结果会有较大的出入，分词及相关评价指标的计算会受到较大的影响。

【解决办法】指定分词方式，如将数字序列(1997 年的“1997”)统一视为整体进行划分。

2. 用于词性标注训练的基词典无法涵盖所有可能出现的词，算法分词过程中会存在缺失键。

【解决办法】默认按照缺失键均为名词进行处理。

3. 在写 Viterbi 算法的过程中，无法很好地将流程表达为程序语言。

【解决办法】参考了其他高校课件内容进行学习。

七．实验心得

在本次实验过程中,我成功地将三种分词算法及词性标注的 Viterbi 算法以程序语言的形式表达出来，脱离了单纯的理论学习，看到程序“跑”出来的瞬间，我有种说不上来的成就感。由于原先很少接触过自然语言处理领域的相关程序，我在学习写代码的过程中遇到了很多困难，所幸最后这些都被一一解决。

这次实验让我对自然语言处理的基本任务有了更加深刻的认识,也让我的编程水平有了提高，非常感谢能有这次机会进行学习。

附录 1

```
import csv
import time

# 加载词典
def get_dict(document):
    dt = set()
    # 按行读取字典文件，每行第一个空格之前的字符串提取出来。
    for line in open(document, "r",encoding='utf-8'):
        dt.add(line[0:line.find(' ')])
    return dt

# 正向最长匹配算法
def forward_segment(text, dict):
    word_list = []
    i = 0
    while i < len(text):
        longest_word = text[i]                          # 记录当前扫描位置的单字
        for j in range(i + 1, len(text) + 1):           # 对于所有可能的结尾
            word = text[i:j]                            # 记录从当前位置到结尾的连续字符
串
            if word in dict:                             # 判断单词是否在词典中
                if len(word) > len(longest_word):       # 判断是否该串更长，若是则优先输
出
                    longest_word = word
        word_list.append(longest_word)                  # 输出最长词，并进行正向扫描
        i += len(longest_word)
    return word_list

# 逆向最长匹配算法
def backward_segment(text, dict):
    word_list = []
    i = len(text) - 1
    while i >= 0:                                       # 扫描位置作为终点
```

```
        longest_word = text[i]                          # 扫描位置的单字
        for j in range(0, i):                           # 遍历[0, i]区间作为待查询词语的起点
            word = text[j: i + 1]                       # 取出[j, i]区间作为待查询单词
            if word in dict:
                if len(word) > len(longest_word):       # 越长优先级越高
                    longest_word = word
                    break
        word_list.insert(0, longest_word)               # 逆向扫描，所以越先查出的单词在位置上越靠后
        i -= len(longest_word)
    return word_list

# 双向最长匹配，实际上是做了个融合
def bidirectional_segment(text, dict):
    forward = forward_segment(text, dict)
    backward = backward_segment(text, dict)
    # 词数更少更优先
    if len(forward) < len(backward):
        return forward
    elif len(forward) > len(backward):
        return backward
    else:
        # 单字更少更优先
        if sum(1 for word in forward if len(word) == 1) < sum(1 for word in backward if len(word) == 1):
            return forward
        else:
            return backward

# 使用正向最长匹配对文本分词
def get_txt_forward(text):
    ls = []
    # 按行读取字典文件，每行第一个空格之前的字符串提取出来。
    for line in open(text, "r",encoding='utf-8'):
```

```
        if(line != '\n'):
            ls.append(forward_segment(line.strip(), dt))
    return ls

# 使用正向最长匹配对文本分词
def get_txt_backward(text):
    ls = []
    # 按行读取字典文件，每行第一个空格之前的字符串提取出来。
    for line in open(text, "r",encoding='utf-8'):
        if(line != '\n'):
            ls.append(backward_segment(line.strip(), dt))
    return ls

# 使用正向最长匹配对文本分词
def get_txt_bidirectional(text):
    ls = []
    # 按行读取字典文件，每行第一个空格之前的字符串提取出来。
    for line in open(text, "r",encoding='utf-8'):
        if(line != '\n'):
            ls.append(bidirectional_segment(line.strip(), dt))
    return ls

# 将分词结果转换为其在语句中的起始位置
def convert(text: list):
    ans = []
    i = 1
    for word in text:
        ans.append([i, i + len(word) - 1])
        i += len(word)
    return ans

def wordcut(file):
    if(file.split('.')[1] == 'txt'):
        ##########################################################################
        # 根据暂无分词结果的.txt 文件进行分词
```

```
        ##########################################################################
        print("请输入分词方法，其中 1 为正向最长分词，2 为逆向最长分词，3 为双向最长分词：")
        type = input()
        if(type == '1'):
            # 正向最长
            # 输入待分词文本进行正向分词
            wordcut_forward = get_txt_forward(file)
            # 正向最长匹配算法结果：
            print("正向最长匹配算法结果：")
            for i in wordcut_forward:
                print(i)
            return wordcut_forward
        elif(type == '2'):
            # 逆向最长
            # 输入待分词文本进行逆向分词
            wordcut_backward = get_txt_backward(file)
            # 逆向最长匹配算法结果：
            print("逆向最长匹配算法结果：")
            for i in wordcut_backward:
                print(i)
            return wordcut_backward
        elif(type == '3'):
            # 双向最长
            # 输入待分词文本进行双向分词
            wordcut_bidirectional = get_txt_bidirectional(file)
            # 双向最长匹配算法结果：
            print("双向最长匹配算法结果：")
            for i in wordcut_bidirectional:
                print(i)
            return wordcut_bidirectional
    elif(file.split('.')[1] == 'csv'):
        ##########################################################################
        # 根据已有分词结果的.csv 文件进行分词
        ##########################################################################
```

```
        reader = csv.reader(open(file, "r", encoding='utf-8'))
        test_sents = []
        ans_sents = []
        for item in reader:
            #    print(item)              item 是每条语句分词后的词列表
            test_sent = ""
            ans_sent = []
            for word in item:
                #    print(word)          word 是每个词列表中的每个词
                test_sent += word.strip()
                #    print(test_sent)     test_sent 最终是一条完整的语句
                ans_sent.append(word)
                #    print(ans_sent)      ans_sent 最终是分词列表(item)
            test_sents.append(test_sent)
            #   test_sents 是所有完整语句构成的列表
            ans_sents.append(ans_sent)
            #   ans_sents 是所有分词列表构成的列表

        # test_sents_num 是 test_sents 的长度
        test_sents_num = len(test_sents)

        # 分别指代精度、召回率、F-measure 值
        p_sum = 0
        r_sum = 0
        ans = []
        print("请输入分词方法，其中 1 为正向最长分词，2 为逆向最长分词，3 为双向最长分词：")
        type = input()
        if (type == '1'):
            for i in range(test_sents_num):
                # 指定“正向最长匹配”方法对每一条语句进行训练
                train = forward_segment(test_sents[i], dt)
                ans.append(train)
                train_list = convert(train)
                ans_list = convert(ans_sents[i])
```

```
                train_set = set()
                for ele in train_list:
                    train_set.add(tuple(ele))  # train_set 中元素为经过训练得到的不
计位置的元组

                ans_list_set = set()
                for ele in ans_list:
                    ans_list_set.add(tuple(ele))  # ans_set 中元素为已有的不计位置的
元组

                # TP 为两个集合中共有的元组
                TP = ans_list_set & train_set

                # 对每一条语句计算精度和召回率并加和
                p = len(TP) / len(train_list)
                r = len(TP) / len(ans_list)
                p_sum += p
                r_sum += r
        elif (type == '2'):
            for i in range(test_sents_num):
                # 指定“逆向最长匹配”方法对每一条语句进行训练
                train = backward_segment(test_sents[i], dt)
                ans.append(train)
                train_list = convert(train)
                ans_list = convert(ans_sents[i])

                train_set = set()
                for ele in train_list:
                    train_set.add(tuple(ele))  # train_set 中元素为经过训练得到的不
计位置的元组

                ans_list_set = set()
                for ele in ans_list:
                    ans_list_set.add(tuple(ele))  # ans_set 中元素为已有的不计位置的
```

元组

```
                # TP 为两个集合中共有的元组
                TP = ans_list_set & train_set

                # 对每一条语句计算精度和召回率并加和
                p = len(TP) / len(train_list)
                r = len(TP) / len(ans_list)
                p_sum += p
                r_sum += r
        elif (type == '3'):
            for i in range(test_sents_num):
                # 指定“双向最长匹配”方法对每一条语句进行训练
                train = bidirectional_segment(test_sents[i], dt)
                ans.append(train)
                train_list = convert(train)
                ans_list = convert(ans_sents[i])

                train_set = set()
                for ele in train_list:
                    train_set.add(tuple(ele))  # train_set 中元素为经过训练得到的不
计位置的元组

                ans_list_set = set()
                for ele in ans_list:
                    ans_list_set.add(tuple(ele))  # ans_set 中元素为已有的不计位置的
元组

                # TP 为两个集合中共有的元组
                TP = ans_list_set & train_set

                # 对每一条语句计算精度和召回率并加和
                p = len(TP) / len(train_list)
                r = len(TP) / len(ans_list)
                p_sum += p
```

```
                r_sum += r

        # 计算均值
        P = p_sum / test_sents_num
        R = r_sum / test_sents_num
        F = 2 * P * R / (P + R)
        print("该方法的分词精度为：", P)
        print("该方法的分词召回率为：", R)
        print("该方法的分词 F 值为：", P)
        return ans


def text_save(filename, data):#filename 为写入 CSV 文件的路径，data 为要写入数据列表.
    file = open(filename,"w+",encoding='utf-8')
    for i in range(len(data)):
        s = str(data[i])
        file.writelines(s+"\n")
    file.close()
    print("保存文件成功")

# 导入字典
dt = get_dict("CoreNatureDictionary.txt")
# 输入待分词文本
file = input()
starter = time.time()
res = wordcut(file)
# text_save('fortrain_forward.txt',res)
# text_save('fortrain_backward.txt',res)
# text_save('fortrain_bidirectional.txt',res)
ender = time.time()
print("用时：",ender - starter,"s")
```

附录 2

```
import csv
import time
```

```
# 加载语料库
def load_corpus(document):
    file = open(document, "r", encoding='utf-8')
    reader = csv.reader(file)
    sents = []
    for item in reader:
        sent = []
        for i in item:
            sent.append(i.strip())
        sents.append(sent)
    return sents

# 获得词性标注字典
def get_dict_wordattr(sents):
    dt_wordattr = dict()
    # 统计所有词性个数
    for sent in sents:
        for word in sent:
            word_part = word.split('/')[-1].split(']')[0].split('!')[0]
            if word_part in dt_wordattr:
                dt_wordattr[word_part] += 1
            else:
                dt_wordattr[word_part] = 1
    return dt_wordattr

# 计算马尔科夫链的转移矩阵
# 获得转移矩阵
def get_transMatrix(sents):
    trans = dict()
    for sent in sents:
        for i in range(len(sent) - 1):
            one = sent[i].split('/')[-1].split(']')[0].split('!')[0]
            two = sent[i + 1].split('/')[-1].split(']')[0].split('!')[0]
            if one in trans:
                if two in trans[one]:
```

```
                    trans[one][two] += 1
                else:
                    trans[one][two] = 1
            else:
                trans[one] = dict()
                trans[one][two] = 1
    return trans

# 每个词的词性概率
def get_percent(sents):
    percent = dict()
    for sent in sents:
        for word in sent:
            word_word = word.split('/')[0].split('{')[0].strip('[')
            word_part = word.split('/')[-1].split(']')[0].split('!')[0]
            if word_word in percent:
                if word_part in percent[word_word]:
                    percent[word_word][word_part] += 1
                else:
                    percent[word_word][word_part] = 1
            else:
                percent[word_word] = dict()
                percent[word_word][word_part] = 1
    return percent

def Viterbi(text,text_percent,trans,dic):
    dict_len = len(dic)
    dis = [dict() for _ in range(len(text))]
    node = [dict() for _ in range(len(text))]
    for first in text_percent[0].keys():
        dis[0][first] = 1
    for i in range(len(text) - 1):
        word_one = text[i]
        word_two = text[i + 1]
        word_one_percent_dict = text_percent[i]
```

```
        word_two_percent_dict = text_percent[i + 1]

        word_one_percent_key = list(word_one_percent_dict.keys())
        word_one_percent_value = list(word_one_percent_dict.values())
        word_two_percent_key = list(word_two_percent_dict.keys())
        word_two_percent_value = list(word_two_percent_dict.values())
        for word_two_per in word_two_percent_key:
            tmp_dis = []
            for word_one_per in word_one_percent_key:
                if word_two_per in trans[word_one_per]:
                    tmp_num = dis[i][word_one_per] * (
                            (trans[word_one_per][word_two_per]      +      1)      /
(dic[word_one_per] + dict_len)) * (
                                    text_percent[i    +    1][word_two_per]    /
dic[word_two_per])
                    tmp_dis.append(tmp_num)
                else:
                    tmp_num  =  dis[i][word_one_per]  *  (1  /  (dic[word_one_per]  +
dict_len)) * (
                            text_percent[i + 1][word_two_per] / dic[word_two_per])
                    tmp_dis.append(tmp_num)

            max_tmp_dis = max(tmp_dis)
            max_tmp_dis_loc = tmp_dis.index(max_tmp_dis)
            dis[i + 1][word_two_per] = max_tmp_dis
            node[i + 1][word_two_per] = word_one_percent_key[max_tmp_dis_loc]

    path = []
    f_value = list(dis[len(dis) - 1].values())
    f_key = list(dis[len(dis) - 1].keys())
    f = f_key[f_value.index(max(f_value))]
    path.append(f)
    for i in range(len(dis) - 1, 0, -1):
        f = node[i][f]
        path.append(f)
```

```
    path.reverse()
    return path

def get_text_percent(text,percent):
    text_percent = []
    for word in text:
        # 对于语料库中不存在的词性标注，我们统一认为是名词
        try:
            word_percent = percent[word]
        except KeyError as e:
            word_percent = {'n': 1}
        text_percent.append(word_percent)
    return text_percent

def get_ans(mytxt,percent_wordattr,trans_wordattr,dt_wordattr,sents_part):
    counter = 0
    p_sum = 0
    r_sum = 0
    for line in open(mytxt, "r", encoding='utf-8'):
        text = eval(line)
        print("第", counter + 1, "行:", text)
        text_percent = get_text_percent(text, percent_wordattr)
        print(text_percent)
        ans = Viterbi(text, text_percent, trans_wordattr, dt_wordattr)
        # print(ans)
        myList = []
        for i in range(len(text)):
            myList.append(text[i] + '/' + ans[i])
        print("算法词性标注结果：", myList)
        print("人工词性标注结果", sents_part[counter])

        dict_train = {}
        dict_check = {}
        for i in myList:
            try:
```

```
            tmp = dict_train[i]
        except:
            dict_train[i] = 1
        else:
            dict_train[i] += 1

    for i in sents_part[counter]:
        try:
            tmp = dict_check[i]
        except:
            dict_check[i] = 1
        else:
            dict_check[i] += 1

    # 求精度
    p = 0
    for i in dict_train.keys():
        try:
            tmp = dict_check[i]
        except:
            p += 0
        else:
            p += min(dict_check[i], dict_train[i])
    print("预测成功数：", p)
    p_percent = p / sum(j for j in dict_train.values())
    print("预测成功率：", p_percent)
    p_sum += p_percent

    # 求召回率
    r = 0
    for i in dict_check.keys():
        try:
            tmp = dict_train[i]
        except:
            r += 0
```

```
            else:
                r += min(dict_check[i], dict_train[i])
        print("召回成功数：", r)
        r_percent = r / sum(j for j in dict_check.values())
        print("召回成功率：", r_percent)
        r_sum += r_percent

        counter += 1
        print()

    # 计算均值
    P = p_sum / counter
    R = r_sum / counter
    F = 2 * P * R / (P + R)
    print("该方法的分词精度为：", P)
    print("该方法的分词召回率为：", R)
    print("该方法的分词 F 值为：", P)


# 加载语料库
sents = load_corpus("train.csv")
sents_part = load_corpus("corpus_wordattr.csv")
mytxt = input()
starter = time.time()
# 获得语料字典
dt_wordattr = get_dict_wordattr(sents)
dt_len = len(dt_wordattr)
# 获得语料转移矩阵
trans_wordattr = get_transMatrix(sents)
# 获得每个词的词性频度列表
percent_wordattr = get_percent(sents)
# print(percent_wordattr)
get_ans(mytxt,percent_wordattr,trans_wordattr,dt_wordattr,sents_part)
ender = time.time()
print("用时：",ender - starter,"s")
```