实验报告

一、 实验成果

在”分词.ipynb”中实现了中文分词工具包，包括：FMM、BMM、最短路径分词法、最大概率法分词法和基于 HMM 的分词法。这些算法应用于同一个测试集得到的分词结果分别储存在“FMM 分词.txt”、“BMM 分词.txt”、“最短路径分词.txt”、“最大概率分词.txt”和“HMM 分词.txt”中。

二、算法介绍

1. FMM

（1）算法介绍及伪代码

FMM 为正向最大匹配法，是指以词典为依据，从左向右取词典中最长词长度作为第一次取字数量的长度，每次右边减一个字，直到词典中存在或剩下 1 个单字。

伪代码如下：

```
def FMM_func(word_dict, sentence): #word _dict 词典；sentence 句子
    max_len = max([len(item) for item in word_dict]) #  词典中最长词长度
    start = 0
    while start != len(sentence):
        index = start+max_len
        if index>len(sentence):
            index = len(sentence)
        for i in range(max_len):
            #直到词典中存在或只剩一个字
            if (sentence[start:index] in word_dict) or (len(sentence[start:index])==1):
                print(sentence[start:index], end='/')
                start = index
                break
            index += -1 #不断递减尝试
```

（2）效果展示

选择如下的一个小测试集：

```
1 我们要更好地坚持解放思想实事求是的思想路线解放思想实事求是是邓小平理论的精髓实践证明只有解放思想实事求是才能冲破各种不切合实际的或者过时的观念的束缚真正做到尊重认识和掌握客观规律勇于突破勇于创新不断开创社会主义现代化建设的新局面党的十五大是我们党解放思想实事求是的新的里程碑进一步认真学习和掌握十五大精神解放思想实事求是我们的各项事业就能结出更加丰硕的成果
2 北京举行新年音乐会
3
4 向广大职工祝贺新年对节日坚守岗位的同志们表示慰问
5
6 明天气象预报１９９８年元月１日２０时元月２日２０时
```

测试一下，分词效果如下，可以看出比较准确。

```
我们/要/更/好/地/坚持/解放思想/实事求是/的/思想/路线/解放思想/实事求是/是/邓小平理论/的/精髓/实践/证明/只有/解放思想/实事求是/才能/冲破/各种/不/切合/实际/的/或者/过时/的/观念/的/束缚/真正/做到/尊重/认识/和/掌握/客观/规律/勇于/突破/勇于/创新/不断/开创/社会主义/现代化/建设/的/新/局面/党/的/十五大/是/我们/党/解放思想/实事求是/的/新/的/里程碑/进一步/认真/学习/和/掌握/十五大/精神/解放思想/实事求是/我们/的/各项/事业/就/能/结/出/更加/丰硕/的/成果/
/北京/举行/新年/音乐会/
/
/向/广大/职工/祝贺/新年/对/节日/坚守/岗位/的/同志/们/表示/慰问/
/
/明天/气象/预报/１９９８年/元月/１日/２０时/元月/２日/２０时/
/
```

2. BMM

（1）算法介绍及伪代码

BMM 为逆向最大匹配法，基本原理与 FMM 相同，只不过取词时按照从右向左的顺序，因此输出结果时也需要反向输出。

伪代码如下：

```
def BMM_func(word_dict, sentence): #word _dict 词典； sentence 句子
    max_len = max([len(item) for item in word_dict]) #  词典中最长词长度
    result = []
    start = len(sentence) #从末尾开始
    while start != 0:
        index = start - max_len
        if index < 0:
            index = 0
        for i in range(max_len):
            #直到词典中存在或只剩一个字
            if (sentence[index:start] in word_dict) or (len(sentence[start:index])==1):
                result.append(sentence[index:start])
                start = index
                break
            index += 1
    for i in result[::-1]:
        print(i, end='/')
```

（2）效果展示

应用于小测试集，结果如下，整体效果不错，但有部分地方不够准确。

```
我们/要/更/好/地/坚持/解放思想/实事/求/是的/思想/路线/解放思想/实事求是/是/邓小平理论/的/精髓/实践/证明/只有/解放思想/实事求是/才能/冲破/各种/不/切合/实际/的/或者/过时/的/观念/的/束缚/真正/做到/尊重/认识/和/掌握/客观/规律/勇于/突破/勇于/创新/不断/开创/社会主义/现代化/建设/的/新/局面/党/的/十五大/是/我们/党/解放思想/实事/求/是的/新/的/里程碑/进一步/认真/学习/和/掌握/十五大/精神/解放思想/实事求是/我们/的/各项/事业/就/能/结/出/更加/丰硕/的/成果/
/北京/举行/新年/音乐会/
/
/向/广大/职工/祝贺/新年/对/节日/坚守/岗位/的/同志/们/表示/慰问/
/
/明天/气象/预报/１９９８年/元月/１日/２０时/元月/２日/２０时/
/
```

3. 最短路径分词法

（1）算法介绍及伪代码

最短路径分词法的基本思想是在词图上选择一条词数最少的路径。因此，需要为每个句子建立有向无环图，每个字作为图的一个定点，边代表可能的分词。并且为了使分词结果更具有一般性和普适性，为每条边赋了权值，权值为对应词语的出现次数。然后最终求图上权值和最大的一条路径作为分词结果即可。（完整的词语的出现次数会比单字更多，更有机会被选中为一个词，所以可以达到最短路径的目的）

在建好 DAG 图求解最优路径时，本应用 Floyd 或 Dijsktra 算法，但是考虑到效率问题，选择使用贪心算法求解，虽然只能得到近似最优解，不过从效果来看已经足够好。

伪代码:

```
class DAG():     #word_dict 是词频词典，sentence 是句子
    def build_dag(self, sentence): #构建有向无环图
        dag = {}
        for start in range(len(sentence)):
            #对每个字，为以其为开头的所有词语建立结点和连边（权值为词频）
            unique = [start + 1] #与 start 相连的结点在句子中下标
            tmp = [(start + 1, 1)] #边及其权重
            for stop in range(start+1, len(sentence)+1):
                fragment = sentence[start:stop]
                num = word_dict.get(fragment, 0)
                if num > 0 and (stop not in unique): #遇到新的词语，添加连边和权重
                    tmp.append((stop, num))
                    unique.append(stop)
            dag[start] = tmp
        return dag

    def predict(self, sentence):    #选择最优路径
        Len = len(sentence)
        route = [0] * Len
        dag = self.build_dag(sentence)
        for i in range(0, Len):
            route[i] = max(dag[i], key=lambda x: x[1])[0] #贪心算法求解
        return route

    def cut(self, sentence):    #按照最优路径来切分原句
        route = self.predict(sentence)
        next = 0
        word_list = []
        i = 0
        while i < len(sentence):
            next = route[i]
            word_list.append(sentence[i:next])
```

```
            i = next
        for word in word_list:
            print(word+"/")
        return word_list
```

（2）效果展示

应用于小测试集，结果如下，整体比较准确，但“实事求是”被分成了两个词。

```
1 我们/要/更/好/地/坚持/解放思想/实事/求是/的/思想/路线/解放思想/实事/求是/是/邓小平理论/的/精髓/实践/
  证明/只有/解放思想/实事/求是/才能/冲破/各种/不/切合/实际/的/或者/过时/的/观念/的/束缚/真正/做到/尊重/
  认识/和/掌握/客观/规律/勇于/突破/勇于/创新/不断/开创/社会/主义/现代化/建设/的/新/局面/党/的/十五大/
  是/我们/党/解放思想/实事/求是/的/新/的/里程/碑/进一步/认真/学习/和/掌握/十五大/精神/解放思想/实事/求
  是/我们/的/各项/事业/就/能/结/出/更加/丰硕/的/成果/
2 /北京/举行/新年/音乐/会/
3 /
4 /向/广大/职工/祝贺/新年/对/节日/坚守/岗位/的/同志/们/表示/慰问/
5 /
6 /明天/气象/预报/１９９８年/元月/１日/２０/时/元月/２日/２０/时
7 /
```

4. 最大概率分词法

（1）算法介绍及伪代码

最大概率分词法的基本思想是在词图上选择概率最大的分词路经作为最优结果，需要通过选出句子中所有的候选词，计算它们的累计概率，在不同的词语组合中选出累计概率最大的组合作为最终的分词结果。而每个词的累计概率，等于它原来的概率乘上累计概率最大的左邻词的概率，因此要先按从左到右的顺序取出全部候选词，然后在计算时按照动态规划的思路，选择累积概率最大的词作为最佳左邻词，最终从右往左依次输出最佳左邻词即可得到分词结果。不过在实现过程中，为了防止句子过长导致概率相乘的值太小，在此对概率取对数并加符号，从而把相乘的最大值变成相加的最小值。

伪代码如下：

```
def max_pro(sentence):
# word_dict 是词典，包括词语和其出现概率的对数值；sentence 是句子
    dp = [9999] * len(sentence)   #  到每个字为止的最大概率
    root = [0] * len(sentence)   #  每个字所在词的起始下标
    max_len = 0
    for i in word_dict.keys():
        max_len=max(max_len,len(i))##获得最大长度

    for i in range(len(sentence)):
        findflag = 0
        for j in range(i, i+max_len):
            if j < len(sentence):   #对每个字，尝试以其为开头的所有词
                word = sentence[i: j+1]
                if word in word_dict.keys():
                    findflag = 1
                    temp_pro = word_dict[word]
```

```
                    if i > 0:
                        temp_pro += dp[i-1]
                    if temp_pro < dp[j]:    #选择保留概率最大的词
                        dp[j] = temp_pro
                        root[j] = i
                else:
                    break
            if (findflag == 0) and (dp[i] == 9999):
#如果以这个字开头的所有词都不在词典中，并且这个字也不在其他词中，则需要单独处理
                dp[i] = dp[i - 1] + 20
                root[i] = i    #单字为词，并且代价较大

        # 输出结果
        result = []
        word_tail = len(sentence) - 1
        while word_tail >= 0:    #从词尾开始向前不断寻找最优解
            result.append(sentence[root[word_tail]: word_tail + 1])
            word_tail = root[word_tail] - 1
        result.reverse()
        for word in result:
            print(word+"/")
        return dp, root, result
```

（2）效果展示

应用于小测试集，结果如下，整体效果很好。

```
1 我们/要/更/好/地/坚持/解放思想/实事求是/的/思想/路线/解放思想/实事求是/是/邓小平理论/的/精髓/实践/证明/只有/解放思想/实事求是/才能/冲破/各种/不/切合/实际/的/或者/过时/的/观念/的/束缚/真正/做到/尊重/认识/和/掌握/客观/规律/勇于/突破/勇于/创新/不断/开创/社会主义/现代化/建设/的/新/局面/党/的/十五大/是/我们/党/解放思想/实事求是/的/新/的/里程碑/进一步/认真/学习/和/掌握/十五大/精神/解放思想/实事求是/我们/的/各项/事业/就/能/结/出/更加/丰硕/的/成果
2 /北京/举行/新年/音乐会
3 /
4 /向/广大/职工/祝贺/新年/对/节日/坚守/岗位/的/同志/们/表示/慰问
5 /
6 /明天/气象/预报/１９９８年/元月/１日/２０时/元月/２日/２０时
7 /
```

5. 基于 HMM 的分词法

（1）算法介绍及伪代码

HMM 为隐马尔可夫模型，是用于描述由隐藏的状态序列和显性的观测序列组合而成的双重随机过程。HMM 由状态值集合、观察值集合、状态转移概率矩阵、发射概率矩阵和初始状态分布五个参数组成。

其中，状态值集合为(B, M, E, S)，每个状态代表的是该字在词语中的位置，B 代表该字是词语中的起始字，M 代表是词语中的中间字，E 代表是词语中的结束字，S 则代表是单字

成词；观察值集合就是所有汉字，甚至包括标点符号所组成的集合；状态转移概率矩阵的含义就是从状态 X 转移到状态 Y 的概率，是一个 4×4 的矩阵，即{B,E,M,S}×{B,E,M,S}；发射概率矩阵的每个元素都是一个条件概率，代表某状态下观察到某个观测值的概率；初始状态概率分布表示句子的第一个字属于{B,E,M,S}这四种状态的概率。

我们需要对训练集的输入数据进行处理，来计算 HMM 的参数，跟据每个状态值和观测值填充矩阵，最后根据转移矩阵进行预测，输出分词结果。但是由于预测时的效率问题，需要采用维特比算法，利用类似动态规划的思路记录概率最大的值。

伪代码如下：

```
#====处理训练集====
Init_Array()
for line in trainset: #对每行分别处理，添加状态序列、词典、状态转移概率和发射概率
    line = line.strip()
    line_num += 1
    word_list = []
    for k in range(len(line)):
        if line[k] == ' ':continue
        word_list.append(line[k])
    if len(word_list) == 0:
        continue
    word_set = word_set | set(word_list)       #训练集所有字的集合
    line = line.split()
    line_state = []         #这句话的状态序列
    for i in line:
        line_state.extend(get_tag(i)) #获得并添加状态序列
    array_Pi[line_state[0]] += 1    # array_Pi 用于计算初始状态分布概率
    for j in range(len(line_state)-1):
        array_A[line_state[j]][line_state[j+1]] += 1    #array_A 计算状态转移概率
    for p in range(len(line_state)):
        count_dic[line_state[p]] += 1    #  记录每一个状态的出现次数
        for state in STATES:
            if word_list[p] not in array_B[state]:
                array_B[state][word_list[p]] = 0.0    #保证每个字都在 STATES 的字典中
        array_B[line_state[p]][word_list[p]] += 1    # array_B 用于计算发射概率
Prob_Array()       #对概率取对数保证精度

#Viterbi 算法求测试集的最优状态序列，之后再按照状态序列进行分词
def Viterbi(sentence,array_pi,array_a,array_b):
    tab = [{}]    #动态规划表
    path = {}
    if sentence[0] not in array_b['B']:
        for state in STATES:
            if state == 'S':
                array_b[state][sentence[0]] = 0
            else:
```

```
                array_b[state][sentence[0]] = -3.14e+100
    for state in STATES:
        tab[0][state] = array_pi[state] + array_b[state][sentence[0]]
        #tab[t][state]表示时刻 t 到达 state 状态的所有路径中，概率最大路径的概率值
        path[state] = [state]
    for i in range(1,len(sentence)):
        tab.append({})
        new_path = {}
        for state in STATES:
            if state == 'B':
                array_b[state]['begin'] = 0
            else:
                array_b[state]['begin'] = -3.14e+100
        for state in STATES:
            if state == 'E':
                array_b[state]['end'] = 0
            else:
                array_b[state]['end'] = -3.14e+100
        for state0 in STATES:
            items = []
            for state1 in STATES:
                if sentence[i] not in array_b[state0]:
                #所有在测试集出现但没有在训练集中出现的字符
                    if sentence[i-1] not in array_b[state0]:
                        prob = tab[i - 1][state1] + array_a[state1][state0] +
array_b[state0]['end']
                    else:
                        prob = tab[i - 1][state1] + array_a[state1][state0] +
array_b[state0]['begin']
                else:
                    prob = tab[i-1][state1] + array_a[state1][state0] +
array_b[state0][sentence[i]]    #计算每个字符对应 STATES 的概率
                items.append((prob,state1))
            best = max(items)    #bset:(prob,state)保留概率最大的
            tab[i][state0] = best[0]
            new_path[state0] = path[best[1]] + [state0]
        path = new_path
    prob, state = max([(tab[len(sentence) - 1][state], state) for state in STATES])
    return path[state] #返回状态序列，之后根据其进行分词即可
```

（2）效果展示

应用于小测试集，结果如下，效果大体不错，但在某些地方不够准确，不是非常好。

```
我们/要/更/好/地/坚持/解放/思想/实事求/是/的/思想/路线/解放/思想/实事求/是/是/邓/小平/理论/的/精髓/实践/证明/只/有/解放/思想/实事求/是/才/能/冲破/各种/不/切合/实际/的/或者/过时/的/观念/的/束缚/真正/做到/尊重/认识/和/掌握/客观/规律/勇于/突破/勇于/创新/不断/开创/社会/主义/现代化/建设/的/新局/面党/的/十五大/是/我们/党/解放/思想/实事求/是/的/新/的/里/程碑/进一步/认真/学习/和/掌握/十五大/精神/解放/思想/实事求/是/我们/的/各项/事业/就/能/结出/更加/丰硕/的/成果/
北京/举行/新年/音乐会/
向/广大/职工/祝贺/新年/对/节日/坚守/岗位/的/同志/们/表示/慰问/
明天/气象/预报/１９９８年/元月/１日/２０时/元月/２日/２０时/
```

三、实验过程

1. 预处理

首先要对所给的“词性标注@人民日报 199801.txt”数据集进行处理。
从其中随机抽取 10%的行数作为测试集，剩余 90%作为训练集，产生的原始训练集为'traindata.txt'，原始测试集为'testdata.txt'

```python
#在txt文件中随机抽取10%作为测试集，剩余90%作为训练集
import random
from random import randint

oldf = open('词性标注@人民日报199801.txt', 'r',encoding='utf-8')    #要被抽取的文件（原文件）
trainf = open('traindata.txt', 'w',encoding='utf-8')  #用于存放训练集
testf = open('testdata.txt', 'w',encoding='utf-8')    #用于存放测试集
ratio = 0.9  #训练集占比

lines = oldf.readlines()
for line in lines:
    if random.random()<ratio: #数据集分割比例
        trainf.write(line)  #训练数据集
    else:
        testf.write(line)  #测试数据集

oldf.close()
trainf.close()
testf.close()
```

由于分词的训练集并不需要词性标注和时间，所以设置函数去除原始文档中的时间和后面的词性标注

```python
import re
#处理原始文档得到词典
def string_process(x):    #处理字符串
    a=re.sub(r'\d{8}-\d{2}-\d{3}-\d{3}/m|[/a-z！。”“，、——\[\]（）：《》……A-Z？]', "", x)
#正则表达式去除时间和后面部分的字母和其他符号
    b=a.replace("  "," ")
    return b.rstrip()
```

对训练集的每行用 string_process 函数进行处理，得到"process_1.txt"如下：

```
迈向 充满 希望 的 新 世纪 一九九八年 新年 讲话 附 图片 1 张
中共中央 总书记 国家 主席 江 泽民
一九九七年 十二月 三十一日
１２月 ３１日 中共中央 总书记 国家 主席 江 泽民 发表 １９９８年 新年 讲话 迈向 充满 希望 的 新 世纪
同胞 们 朋友 们 女士 们 先生 们
在 １９９８年 来临 之际 我 十分 高兴 地 通过 中央 人民 广播 电台 中国 国际 广播 电台 和 中央 电视台 向
台湾 同胞 海外 侨胞 向 世界 各国 的 朋友 们 致以 诚挚 的 问候 和 良好 的 祝愿
１９９７年 是 中国 发展 历史 上 非常 重要 的 很 不 平凡 的 一 年 中国 人民 决心 继承 邓 小平 同志 的
前进 中国 政府 顺利 恢复 对 香港 行使 主权 并 按照 一国两制 港人治港 高度 自治 的 方针 保持 香港 的
国 代表大会 高举 邓小平理论 伟大 旗帜 总结 百年 历史 展望 新 的 世纪 制定 了 中国 跨 世纪 发展 的 行动
在 这 一 年 中 中国 的 改革 开放 和 现代化 建设 继续 向前 迈进 国民经济 保持 了 高 增长 低 通胀 的 良
改革 继续 深化 人民 生活 进一步 改善 对外 经济 技术 合作 与 交流 不断 扩大 民主 法制 建设 精神文明 建
注 最近 一个 时期 一些 国家 和 地区 发生 的 金融 风波 我们 相信 通过 这些 国家 和 地区 的 努力 以及 有
国 改革 和 发展 的 全局 继续 保持 了 稳定
在 这 一 年 中 中国 的 外交 工作 取得 了 重要 成果 通过 高层 互访 中国 与 美国 俄罗斯 法国 日本 等 大
针 中国 与 周边 国家 和 广大 发展中国家 的 友好 合作 进一步 加强 中国 积极 参与 亚 太 经合 组织 的 活
会晤 这些 外交 活动 符合 和平 与 发展 的 时代 主题 顺应 世界 走向 多极化 的 趋势 对于 促进 国际 社会
１９９８年 中国 人民 将 满怀信心 地 开创 新 的 业绩 尽管 我们 在 经济 社会 发展 中 还 面临 不少 困难
```

测试集同样进行如上处理得到"test_process_1.txt"，但是由于需要用测试集进行各种算法的分词效果测试，所以需要去掉每行之内的空格连成句子，处理为如下效果存储在“test_endfile.txt”中：

```
我们要更好地坚持解放思想实事求是的思想路线解放思想实事求是是邓小平理论的精髓实践证明只有解放思想实
做到尊重认识和掌握客观规律勇于突破勇于创新不断开创社会主义现代化建设的新局面党的十五大是我们党解放
解放思想实事求是我们的各项事业就能结出更加丰硕的成果
北京举行新年音乐会

向广大职工祝贺新年对节日坚守岗位的同志们表示慰问

明天气象预报１９９８年元月１日２０时元月２日２０时

今天上午中共中央政治局委员李铁映与广播电影电视部部长孙家正国家语委主任许嘉璐等向第一批获得播音员主
近年来贵州省各级党委和政府把扶贫开发工作作为农村中心任务来抓取得很大成绩贫困人口大量减少贫困状况明
众一起研究扶贫开发的路子他说必须坚持开发扶贫的方针通过发展经济解决贫困人口的温饱问题要把农业生产尤
需求充分利用当地资源积极发展多种经营增加农民收入温家宝考察了农田水利建设工地他说要大搞农田基本建设
坚实的物质技术基础之上
```

2. 应用各种算法进行分词

应用以上介绍过的 FMM、BMM、最短路径分词法、最大概率法分词法和基于 HMM 的分词法，用"process_1.txt"进行训练，用“test_endfile.txt”进行测试，并分别输出分词结果到对应文档即可。

四、运行效率对比

分别计算以上算法的运行时间，得到如下结果：

算法	用时
FMM	45min 30s
BMM	46min 50s
最短路径分词法	14.3 s
最大概率法分词法	48.8 s
基于 HMM 的分词法	18.1 s

由此可以看出，FMM 和 BMM 的运行效率最低，明显低于其他算法。其他三种算法效率都明显更高，运行速度更快。其中最短路径分词法由于使用贪心算法求得较优解，所以速度最快。