中文分词算法报告

姓名：崔庆尧　　学号：2018140546

概述

本次中文分词任务使用 python3.x 版本，基于最大概率分词的思想实现了正向最大匹配，逆向最大匹配，二元语法模型（Bigram Model），正向+逆向+Bigram，隐马尔科夫模型（Hidden Markov Model，HMM）五种分词方法，并对比分析这五种分词方法的结果，最终使用正向+逆向+Bigram 进行分词，本次作业程序已同步至 github

数据集格式说明

本次实验的数据集是 199801.txt，格式如图 1 所示：

```
19980131-04-009-001/m  戊寅/t  吉祥/n  肖形虎/n  （/w  图片/n  ）/w

19980131-04-010-001/m  肖形虎/n  （/w  附/v  图片/n  3/m  张/q  ）/w

19980131-04-011-001/m  戊寅/t  吉祥/n  （/w  篆刻/n  ）/w
19980131-04-011-002/m  （/w  马/nr  东生/nr  ）/w
```

图 1 原始数据集 199801.txt

实验流程

Step 1:数据清洗，构建训练数据集和测试数据集。

对 199801.txt 去除空行，行标记 19980131-*，词性，按 9:1 的比例分成训练数据集和测试数据集，清洗后的数据格式如图 2 所示：

```
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
同胞 们 、 朋友 们 、 女士 们 、 先生 们 ：
```

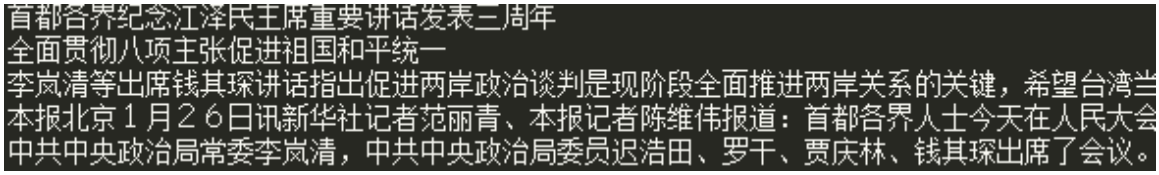

图 2 清洗后的训练集(上)和测试集(下)

Step 2:针对五种不同的方法分别统计准确率(precision)，召回率(recall)和 F1 值。

Step3:对比不同方法的分词效果，选择最佳的分词模型用于最终的分词任务。

分词算法描述

正向最大匹配和逆向最大匹配

正向最大匹配算法（FMM）针对某一给定的句子，从句首开始，设置一个最大词长窗口，将窗口所包围的词与词典中的词进行匹配，若匹配成功，则将窗口中的词取出，若匹配不成功，则减小窗口的大小再次进行匹配，若窗口大小缩减至 1，则将单字切分为词，在经历了一次匹配成功之后，将窗口大小复原，并将其向前推动一格，重复上述流程，直至将整个句子切分完毕，算法的示意图如图 3 所示：

图 3 正向最大匹配算法示意图

逆向最大匹配算法（BMM）针对某一给定的句子，从句尾开始，设置一个最大词长窗口，将窗口所包围的词与词典中的词进行匹配，若匹配成功，则将窗口中的词取出，若匹配不成功，则减小窗口的大小再次进行匹配，若窗口大小缩减至 1，则将单字切分为词，在经历了一次匹配成功之后，将窗口大小复原，并将其向后推动一格，重复上述流程，直至将整个句子切分完毕，算法的示意图如图 4 所示：

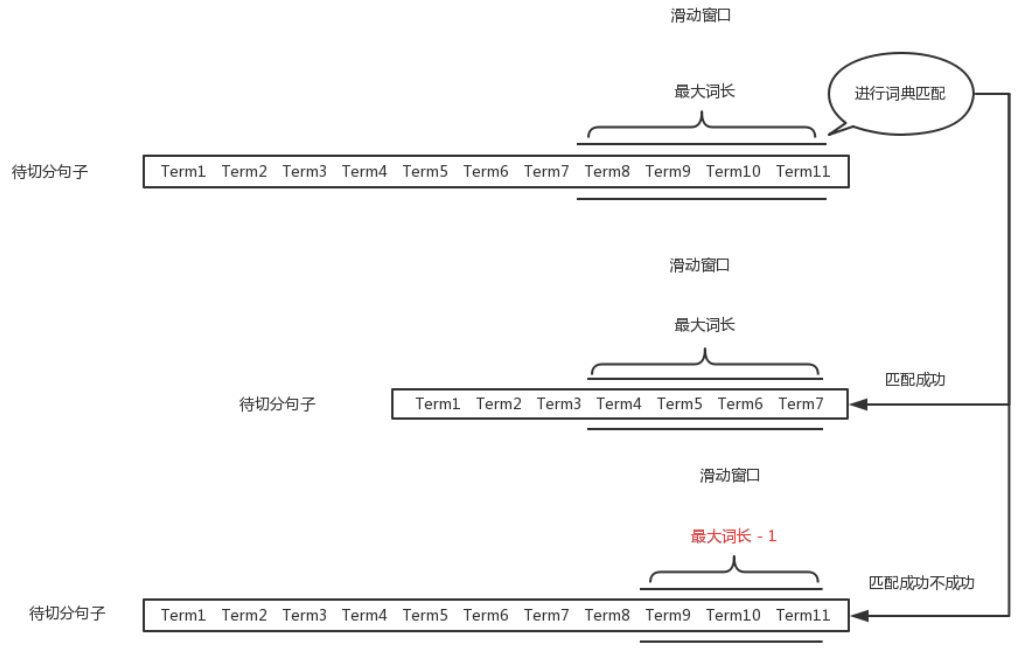

图 4 逆向最大匹配算法示意图

隐马尔科夫模型

利用隐马尔科夫模型（Hidden Markov Model，HMM）进行中文分词，可看作是一个序列标注任务，对于给定一个字符序列，将每个字符打上一个状态标签，最终根据状态序列确定分词结果。

模型介绍

HMM 模型的三个基本假设为：

- $P(status_i \mid status_{i-1}status_{i-2},\ldots,status_1) = P(status_i \mid status_{i-1})$ ，即当前状态值只依赖于前一个状态而与前一个状态以外的其他状态无关。
- $P(status_i \mid status_{i-1}) = P(status_j \mid status_{j-1})$，即状态和当前的时刻无关。
- $P(observed_i \mid status_i status_{i-1}status_{i-2},\ldots,status_1) = P(observed_i \mid status_i)$，即观察值只取决于当前的状态值。

基于分词场景下的 HMM 包含如下的五元组：

- 状态标签集合 Q = {B:词起始字符，E:词结束字符，M:词中间字符，S:单字词}。
- 观察值集合 $O = \{o_1, o_2, \ldots o_n\}$ ，输入句子的中文字符。
- 状态转移概率矩阵 $A[a_{ij}]$ ，其中 a_{ij} 表示从 i 状态转移到 j 状态的概率。
- 发射概率矩阵 $B[b_j(k)]$ ，其中 $b_j(k)$ 表示在状态 j 的条件下生成观测值 o_k 的概率。
- 初始状态概率分布 π

统计结果

其中在对状态转移概率矩阵，发射概率矩阵和初始状态概率分布的统计过程中，为防止计算概率累乘的溢出问题，对所有的概率值取对数(当概率为 0 时，取一个很大的负数-3.14e+100)，从而使得概率累乘变成累加，同时提升了程序的运行速度，HMM 训练所得结果如图 5 所示：

```
-4.532649439277728343e-01
-3.139999999999999844e+100
-3.139999999999999844e+100
-1.009365160097460246e+00
```


{"斗": -7.703382905852996, "孙": -10.018390518845598, "绅": -13.196444349193545, "量": -9.977568524325344, "贺": -8.5050964669644, "姣": -13.196444349193545, "
%.117002807513709, "贩": -10.105401895835229, "讳": -13.196444349193545, "铝": -10.200712075639553, "蟋": -12.097832060525434, "逃": -9.86423983901834, "惆": -11
%"荫": -12.097832060525434, "蛋": -9.938347811172061, "褂": -12.5032971686336, "律": -9.507564895079607, "梁": -11.250534200138231, "拷": -12.097832060525434, "
%.898126982645508, "公": -5.26336457731313, "智": -8.451512220830294, "喋": -12.5032971686336, "我": -5.158578114483927, "渗": -10.15192191147012, "霄": -12.5032
%-13.196444349193545, "撒": -10.423855626953763, "牡": -9.795246967531389, "爬": -10.999219771857325, "滑": -9.053309622802011, "短": -7.997947317927719, "随": -
%"暮": -10.999219771857325, "涪": -12.5032971686336, "扩": -7.240606979728713, "盟": -10.060950133264395, "别": -8.133849316166577, "幅": -9.053309622802011, "阔
%.117002807513709, "凋": -12.5032971686336, "恰": -9.938347811172061, "齐": -9.762457144708398, "敌": -8.685584842676693, "优": -6.256221880073905, "从": -6.7895
%-10.999219771857325, "借": -8.313642426607172, "毓": -10.631494991732007, "妙": -11.117002807513709, "诣": -13.196444349193545, "励": -11.587006436759443, "呼":
% "夸": -9.762457144708398, "抱": -10.557387019578286, "孰": -13.196444349193545, "涣": -11.810149988073654, "仪": -8.054780792690885, "翔": -11.587006436759443,


```
-3.139999999999999844e+100 -1.561894784235418610e-01 -1.933763884546201517e+00 -3.139999999999999844e+100
-7.160332628032077817e-01 -3.139999999999999844e+100 -3.139999999999999844e+100 -6.707731735318983590e-01
-3.139999999999999844e+100 -3.943366862505039427e-01 -1.121247693573048210e+00 -3.139999999999999844e+100
-5.549009876808767006e-01 -3.139999999999999844e+100 -3.139999999999999844e+100 -8.536190402329933979e-01
```

图 5 初始状态概率分布（上），发射概率矩阵（中），转移概率矩阵（下）

其中初始状态分布从上至下依次为 B，E，M，S 四个状态的初始概率分布，发射概率矩阵以 json 格式存储，列表下标从 0-3 以此表示 B，E，M，S 四个状态下得到观察值得概率，状态转移概率矩阵行和列从上到下，从左至右各个维度依次为 B，E，M，S 四个状态，例如第一行第二列表示从 B 状态转移至 E 状态的概率值。

Viterbi 算法

在 HMM 模型分词过程中，当给定某一观察序列，可用 Viterbi 算法找出得到该观察序列概率最大的状态序列，其算法示意图如图 6 所示：

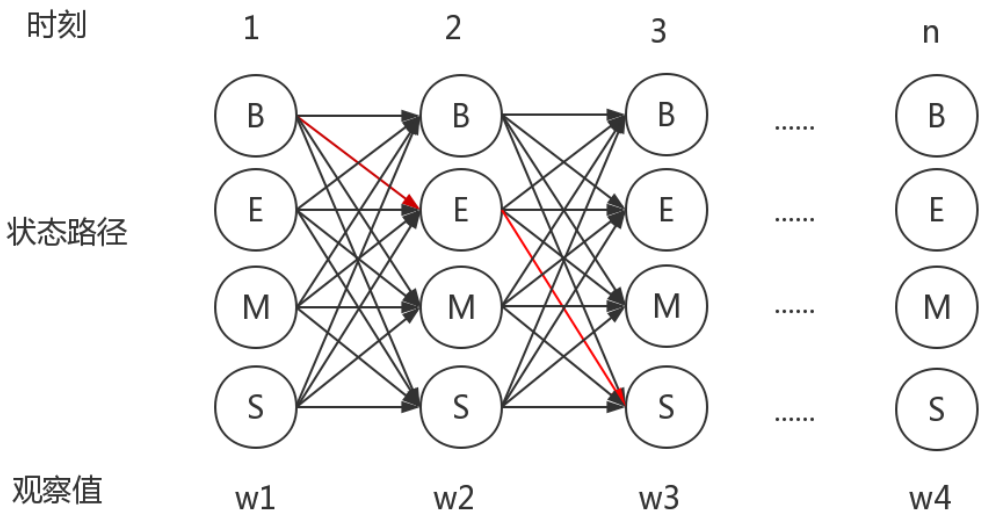

图 6 viterbi 算法示意图

从图中可看出，每一列看做一组节点，初始时可以从任意节点出发（但由于分词任务的限定，一句话的起始只能是由 B 状态或者 S 状态出发，这与初始状态概率分布中第 2,3 行的均为负无穷保持一致），沿着某一条边（转移概率）前进至下一时刻的某一节点并考虑此刻该状态下所能得出该时刻的观察值得概率，将概率累乘起来，选择最大概率的节点继续向下前进，并保存这条最优路径，最后通过回溯的方法得到最佳状态序列。

Viterbi 算法实现代码如下：

```
def __viterbi_cut(self, sentence):
    '''
    viterbi 算法
    :return:
    '''

    # 某状态下，观察值的概率
    weight = np.zeros(shape=(4, len(sentence)))

    # 权重取最大时，前一个观察值的状态
    path = np.zeros(shape=(4, len(sentence)), dtype=np.int)

    # 初始化第一个字
    first_word = sentence[0]

    for s in self.STATUS:
        # 计算条件概率， 给定某状态下，观察值为第一字的概率，这里做加法是因为所有的概率全部都取得 log 值
        weight[s][0] = self.init_status_prob[s] + self.emission_matrix[s].get(first_word, self.LOG_NEGATIVE_INF)

    for word_idx in range(1, len(sentence)):
        for s in self.STATUS:
            weight[s][word_idx] = self.LOG_NEGATIVE_INF
            path[s][word_idx] = -1
            for i in range(4):
                last = weight[i][word_idx - 1] + self.trans_matrix[i][s] + \
                       self.emission_matrix[s].get(sentence[word_idx], self.LOG_NEGATIVE_INF)
                if last > weight[s][word_idx]:
                    weight[s][word_idx] = last
                    path[s][word_idx] = i

    back_through = int(np.argmax(weight, axis=0)[len(sentence) - 1])
    # 回溯过程
    status_result = []
    for idx in range(len(sentence) - 1, -1, -1):
        status_result.append(back_through)
        back_through = path[back_through][idx]
    status_result.reverse()
```

二元语法模型

模型介绍

N-gram 语言模型简单概括就是当前第 n 个单词的概率依赖于前 $n-1$ 个单词的出现。但是在 n 元语法中的这个概率是难以计算的，因此采用一种简化的方法来解决这个问题，即我们假设当前单词的出现的概率只依赖于它前一个单词的出现，这就是二元语法模型（Bigram Model）即：

$$P(w_n \mid w_1^{n-1}) \approx P(w_n \mid w_{n-N+1}^{n-1}), N = 2$$

在实际的分词过程中，首先根据训练语料统计 bigram 频率，即“我|爱”，“我|家”等，将统计所得结果以 json 的格式存储，其格式为:

{前导词:{当前词 1：频率 1，当前词 2：频率 2，…}…}

接着对于给定的一个句子构造多种可能的分割方式，对于每一种分割方式计算出其整个句子的概率（其中对概率值仍然取对数）：

$$P(sentence) = \sum_{i=1}^{len(sentence)-1} \log(P(w_{i+1} \mid w_i))$$

$$P(w_i \mid w_{i-1}) = \frac{C(w_{i-1}w_i)}{C(w_{i-1})}$$

选择所有分割方式中概率最大的作为最为最终的分词结果，其中对于未登录词计算概率时采用的平滑手段是拉普拉斯平滑：

$$P^*_{Laplace}(w_i \mid w_{i-1}) = \frac{C(w_{i-1}w_i)+1}{C(w_{i-1})+V}, V = len(word\_dict)$$

正向+逆向+Bigram

正向+逆向+Bigram 分词方法首先对于输入的句子分别使用正向最大匹配算法和逆向最大匹配算法进行分词，对于两种算法得出的结果不一致的情况，使用上述 Bigram 模型计算二者的概率，选择概率较大的作为最终的分词结果。

实验结果分析

本次分词实验的结果如下表所示：

指标	FMM	BMM	Bigram	FMM+BMM+Bigram	HMM
准确率	0.939	0.941	0.871	**0.948**	0.788
召回率	0.903	0.904	**0.920**	0.911	0.821
F1 值	0.921	0.922	0.893	**0.929**	0.803

从实验结果来看，在准确率和 F1 上 FMM+BMM+Bigram 表现最佳，在召回率上 Bigram 表现最佳， HMM 模型的效果最差。

通过分析分词结果得到结论：

（1）Bigram 模型由于使用拉普拉斯平滑，因此整体降低了概率值，而在进行概率累积的过程中，一步累积所带了的概率衰减太大，因此为了得到更大的概率，Bigram 总是倾向于选择分词数量少的分词方案，不过 Bigram 在处理歧义上具有较好的效果。

（2）由于受到训练语料不完备的限制，HMM 所统计出转移概率矩阵和发射概率矩阵并不具有很好的泛化能力。

（3）FMM+BMM+Bigram 算法的本质是让 Bigram 选择 FMM 和 BMM 那个更好，其中就有四种可能的情况，FMM 和 BMM 都是正确的，FMM 正确 BMM 错误，FMM 错误 BMM 正确，FMM 和 BMM 都是错误。Bigram 在前三种情况下能很好通过自身的歧义处理能力选择正确的方案，但对于第四种情况并不能很好地解决。

程序设计与使用说明

本程序为了调试和计算上的方便，使用了 tqdm，Numpy 扩展库运行之前请安装：

$ pip install tqdm numpy

程序目录结构

```
├─── 2018140546.txt      # 最终分词结果
├─── cut_model.py         # 分词模型实现
├─── cut.py               # 分词程序接口 2，用于生成最终的分词结果
├─── data
│   ├─── answers.txt
│   ├─── hmm              # HMM 训练结果
│   │   ├─── emit_matrix .json
│   │   ├─── init_status_prob.txt
│   │   └─── trans_matrix.txt
│   ├─── n_gram           # bigram 训练结果
│   │   ├─── 199801-bigram.txt
│   │   ├─── bigram_prob_tab_eval.json
│   │   ├─── bigram_prob_tab.json
│   │   └─── word_dict_bigram.json
│   ├─── origin_data      # 原始数据集
│   │   └─── 199801.txt
│   ├─── test.txt
│   ├─── training.txt
│   └─── word_dict.json
├─── data_helper.py       # 数据预处理脚本
├─── evaluation.py        # 分词程序接口 1，用于生成评估报告
├─── LICENSE
├─── README.md
├─── settings.py          # 配置文件
├─── test.txt             # 测试数据
└─── util.py
```

使用说明

分词程序设置了两个接口，一个用于生成模型的评估报告，一个用于切分 test.txt 生成最终的分词结果 2018140546.txt。

数据处理接口

$ python data_helper.py

生成 training.txt，test.txt，answers.txt，word_dict.json 训练集与测试集比例 9:1

分词接口 1 | evaluation.py

FMM 测试报告

将 settings.py 中的 IS_BACK 设置为 False，IS_COMBINE 设置为 False

$ python evaluation.py ms

BMM 测试报告

将 settings.py 中的 IS_BACK 设置为 True，IS_COMBINE 设置为 False

$ python evaluation.py ms

HMM 测试报告

$ python evaluation.py hmm

Bigram 测试报告

将 settings.py 中的 IS_EVALUATION 设置为 True

$ python evaluation.py bigram

FMM+BMM+Bigram 测试报告

将 settings.py 中的 IS_EVALUATION 设置为 True，IS_COMBINE 设置为 True

$ python evaluation.py

分词接口 2 | cut.py

将 settings.py 中的 IS_EVALUATION 设置为 False，IS_COMBINE 设置为 True

$ python cut.py
