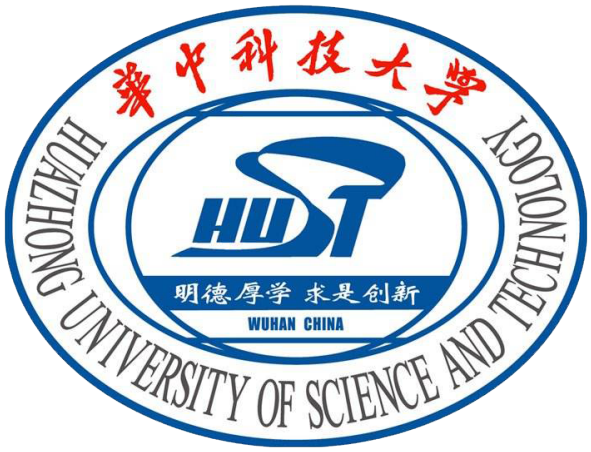
华中科技大学计算机科学与技术学院

《自然语言处理》结课报告

题目：三种中文分词方法的性能对比与评分



专　　业	计算机科学与技术
班　　级	本硕博 2301
学　　号	U202315752
姓　　名	陈宇航
成　　绩	
指导教师	辜希武
时　　间	2025 年 4 月 24 日

目录

摘要 **1**

1 绪论 **2**
1.1 研究背景与动机 2
1.2 中文分词的研究意义 2
1.3 国内外研究现状 2

2 算法理论基础 **3**
2.1 最大匹配（MM）分词 3
2.2 Uni-gram 语言模型分词 3
2.3 隐马尔可夫模型（HMM）分词 5
2.4 小结 5

3 代码实践 **7**
3.1 使用方法 7
3.2 实验结果与分析 8

4 总结与展望 **12**
4.1 工作总结 12
4.2 不足与改进空间 12
4.3 未来启发探究 12
4.4 结语 12

A 附录 I:HMM 实现 **14**

B 附录 III:Unigram 实现 **22**

C 附录 I:HMM 实现 **28**

摘要

本文面向自然语言处理课程，构建了一个统一框架，对比评估三类经典中文分词方法：基于词典的**最大匹配（MM）**、**Uni-gram 语言模型**动态规划分词，以及采用**隐马尔可夫模型（HMM）**并结合 BMES 标注的维特比解码分词。

在 PKU 标准语料上，通过统一的脚本完成批量分词、运行时间统计，并借助轻量级评测器分别计算*准确率*、*召回率*和 *F1* 值。实验结果表明：HMM 在 F1 指标上最优，Uni-gram 兼顾精度与速度，而 MM 具有显著的速度优势但对词典覆盖高度敏感。误差分析揭示了同形异义词歧义、中英文混排及未登录词的主要影响因素。最后，报告提出了引入高阶 n-gram、序列神经网络及专名识别模块以进一步提升分词性能的改进方向，并开放全部代码与数据以促进可复现研究。

关键词： 最大匹配， Uni-gram， 隐马尔可夫模型， 中文分词， 性能评测

1 绪论

1.1 研究背景与动机

随着大数据时代的到来，**自然语言处理（NLP）**已成为人工智能领域最活跃的研究方向之一。中文文本由于缺乏显式的词边界，其预处理阶段必须首先完成中文分词（*Chinese Word Segmentation, CWS*）。分词质量直接影响后续任务——如信息检索、情感分析、机器翻译及预训练语言模型——的性能上限。因此，系统性比较并改进分词算法对于学术研究与工业应用都具有重要意义。

1.2 中文分词的研究意义

1. **信息检索与搜索引擎**——高质量词边界有助于构建准确的倒排索引，并提高查询召回率；
2. **文本挖掘与知识发现**——在主题建模、实体识别和情感分析中，可靠的分词是特征工程的基础；
3. **机器翻译与对话系统**——子词或词粒度的一致性可减少翻译歧义，提高译文流畅度；
4. **预训练语言模型**——分词策略影响词表设计与上下文窗口，对模型性能和推理速度均有显著作用。

1.3 国内外研究现状

目前主流方法大致可分为三类：

- **基于词典与规则的方法**：典型代表为正/反向最大匹配（MM）、双向最大匹配（BiMM）等，依赖完备词典，易实现但难处理未登录词。
- **基于统计语言模型的方法**：以 *Uni-gram / Bi-gram / CRF* 等模型为主，通过概率估计或序列标注解决歧义问题；可在较小词典下保持较好性能。
- **基于深度学习的方法**：BiLSTM-CRF、Transformer、预训练模型（ERNIE, BERT）等，利用上下文表示获得当前最佳准确率，但训练成本与部署复杂度较高。

本课程报告手动实现了搭建**最大匹配（MM）**、**Uni-gram 语言模型**与**隐马尔可夫模型（HMM）**三种分词器的模块化实现，并形成相应的语料评估

2 算法理论基础

2.1 最大匹配（MM）分词

最大匹配系列算法（FMM / BMM / BiMM）把分词任务形式化为最长子串搜索：

$$\hat{W} = (w_1, w_2, \ldots, w_K) = \arg\max_{W \in \mathcal{D}(S)} \sum_{k=1}^{K} \ell(w_k),$$

其中 $\mathcal{D}(S)$ 为对句子 S 的所有可行切分，$\ell(\cdot)$ 为词长。以**正向最大匹配（FMM）**为例，设词典最大词长为 $L_{\max}$，指针从左至右滑动，伪代码见图2.1（亦可参照图2.2）。算法时间复杂度 $\mathcal{O}(nL_{\max})$，空间复杂度 $\mathcal{O}(1)$。

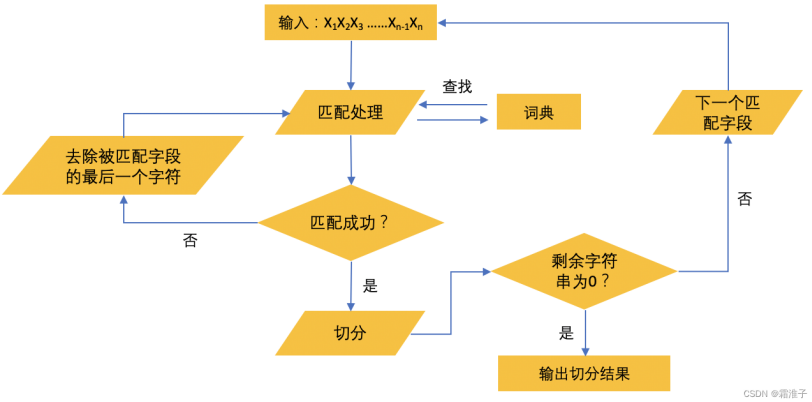


图 2.1: 正向最大匹配 (FMM) 流程示意图

局限性： (1) 词典覆盖决定上限，未登录词全部被拆散；(2) 易在歧义处产生不同长度的候选词，需额外策略（如 BiMM）消歧。

2.2 Uni-gram 语言模型分词

将句子 S 表示为词序列 $W = (w_1, \ldots, w_K)$，分词目标为最大化其语言模型似然：

$$\hat{W} = \arg\max_{W} P(W \mid \Theta) = \arg\max_{W} \prod_{k=1}^{K} P(w_k),$$

对数化并用**动态规划**求解：

$$\mathrm{DP}(i) = \max_{0 \le j < i} \{\mathrm{DP}(j) + \log P(w_{j+1}^{i})\}, \quad \mathrm{DP}(0) = 0,$$

其中 w_{j+1}^{i} 表示 S 的子串 $[j+1, i]$。

Algorithm 1 Forward Maximum Matching

Input: d:concept dictionary, s:clause sentence; $maxL$:max length of the concepts in concept dictionary

Output: w:segmented words

```
p=0
while p != len(s) do
    q = MIN(len(s), p + maxL)
    for all i = 1, 2, ···, maxL do
        if s[p : q] in d or len(s[p : q])==1 then
            w.add(s[p : q])
            p = q
            break
        end if
        q = q − 1
    end for
end while
```

图 2.2: FMM 伪代码

平滑 为避免零概率，可使用加一平滑

$$P_{\text{add1}}(w) = \frac{c(w)+1}{N+V},$$

或 Good–Turing 估计（见式(2.1)）。

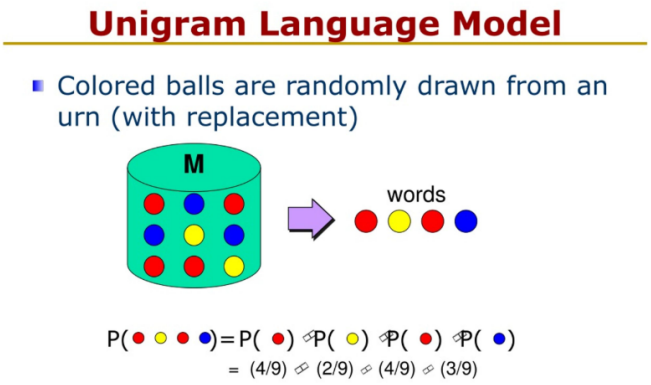


图 2.3: Uni-gram 概率模型示意

2.3 隐马尔可夫模型（HMM）分词

将每个字符视作观测 $O = \{o_t\}_{t=1}^{T}$，隐藏状态为标签 $Q = \{q_t\}_{t=1}^{T}$，$q_t \in \{\text{B,M,E,S}\}$. 模型参数：

$$\pi_i = P(q_1 = i), \quad a_{ij} = P(q_t = j \mid q_{t-1} = i), \quad b_j(o) = P(o_t = o \mid q_t = j).$$

维特比算法

$$\delta_t(i) = \max_{1 \le j \le N} \big\{ \delta_{t-1}(j)\, a_{ji} \big\}\, b_i(o_t), \quad \psi_t(i) = \arg\max_{j} \big\{ \delta_{t-1}(j)\, a_{ji} \big\},$$

最终回溯得最优标签序列 $\hat{Q}$，再按 BMES 规则复原单词边界。

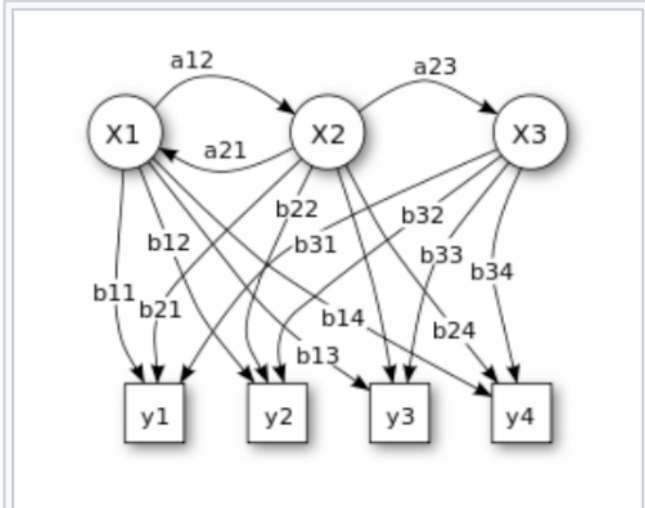


Figure 1. Probabilistic parameters of a hidden Markov model (example)
X — states
y — possible observations
a — state transition probabilities
b — output probabilities

图 2.4: 两状态 HMM 示意——词分模型可映射为四状态 BMES

参数估计与平滑 令 N_r 表示出现频次为 r 的事件数，则 *Simple Good–Turing* 估计为

$$P^*(r) = \frac{(r+1)\, N_{r+1}}{N_r\, N}, \qquad r \ge 0, \tag{2.1}$$

其中 N 为样本总量。实践中常对 π, a, b 分别独立平滑，然后归一化。

2.4 小结

- MM 算法简洁高效，适合资源受限场景，但性能受词典制约；

- Uni-gram 利用词频概率，兼顾速度与准确性，可通过高阶 n-gram 提升表现；
- HMM 借助统计学习可自动识别新词，在 F1 指标上优于前二者，但训练与推断成本更高。

```
BaumWelch(O, hmm)
{
        T = length(O);
        Create NxT matrix alphatable;
        Create NxT matrix betatable;
        Create array newPi of length hmm.N;
        Create NxN matrix newA;
        Create NxM matrix newB;
        done = false;

        while !done
                alphatable = alpha(O, hmm);
                betatable = beta(O, hmm);
                // Initials
1               for i=1 to hmm.N
                        newPi[i] = gamma(i, 1, alphatable, betatable, hmm.N);
                // Transitions
2               for i=1 to N
                        for j=1 to N
                                sum_xi = 0;
                                sum_gamma = 0;
                                for t=1 to T-1
                                    sum_xi += xi(i, j, t, alphatable, betatable, O, hmm);
                                    sum_gamma += gamma(i, t, alphatable, betatable, hmm.N);
                                newA[i][j] = sum_xi / sum_gamma;
                ...
```

图 2.5: 隐式马尔科夫伪代码

下面会从代码实现说明上述算法, 相关算法流程示意图已在Figure 2.1–Figure 2.4给出，便于直观理解。

3 代码实践

本节面向动手环节，首先概览项目的目录层级与核心脚本功能，随后按照“依赖准备 → 词典与模型生成 → 批量分词 → 指标评测”四个步骤，演示完整的复现流程。通过逐步拆解命令与代码片段，读者可以快速在本地环境运行并验证本文提出的三种分词算法，实现从数据预处理到性能对比的端到端实践。代码文件结构如下图3.1：

```
nlp_project/
└── segmentation/
    ├── demo.py              # 交互式演示：一次性加载 3 种分词器并在终端逐句对比
    ├── scorer.py            # 评测脚本：计算 Precision / Recall / F1
    ├── seg.py               # 批处理入口：按参数调用分词器、计时并写结果
    │
    ├── model/               # 分词算法主体
    │   ├── __init__.py      # 空文件，声明 Python 包
    │   ├── mm_seg.py        # 最大匹配（FMM / BMM / BiMM）
    │   ├── unigram_seg.py   # Uni-gram + 动态规划
    │   ├── hmm_seg.py       # HMM + BMES + 维特比
    │   ├── dict_generator.py # 把训练语料转 JSON 词典 / 读取搜狗词典
    │   │
    │   ├── dicts/
    │   │   └── pku_dict.json # PKU 词典（UTF-16，内含词频与总词数 _t_）
    │   │
    │   ├── datasets/
    │   │   └── pku/
    │   │       ├── pku_training_words.utf8 # 官方训练分词语料
    │   │       ├── test.txt              # 未分词测试集
    │   │       ├── gold.txt              # 测试集金标准
    │   │       ├── mm_seg.txt            # 历史 MM 分词结果
    │   │       ├── score                 # PKU 官方评分脚本副本
    │   │       └── ...                   # 其他中间产物
    │   │
    │   └── hmm_para/        # 训练好的 HMM 参数
    │       ├── start.npy    # 初始概率 π
    │       ├── trans.npy    # 转移矩阵 A
    │       └── emit.npy     # 发射矩阵 B
    │
    ├── shared/              # 其他算法通用组件
    │   ├── hmm.py           # 通用 HMM 封装 + Builder + I/O
    │   └── smooths.py       # Simple Good–Turing / 加一平滑实现
    │
    └── test_data/
        └── pku/             # 批处理输出与对照集
            ├── test.txt           # 与 model/datasets/pku/test.txt 相同
            ├── gold.txt           # 对照（复制方便评测）
            ├── mm_seg.txt         # 由 seg.py 生成的 MM 结果
            ├── bmm_seg.txt        # 反向最大匹配结果
            ├── bimm_seg.txt       # 双向最大匹配结果
            ├── unigram_seg.txt    # Uni-gram 结果
            └── hmm_seg.txt        # HMM 结果
```

图 3.1: 代码结构

3.1 使用方法

1. 首先使用 `dict_generator.py` 将所选数据集转换为 JSON 词典。此处采用 PKU 语料；若换用其他数据集，需要修改图3.2 所示的路径、调整 `parser` 参数以及输出文件名，结果如图3.3

2. 然后，运行 seg,py 里的三条参数命令，选择你要采用的中文分词方法，生成不同的结果，运行方式需要在命令行中提供，据图:mm 3.4 unigram3.5 hmm3.6所示

```
python seg.py --segger mm --dataset pku
python seg.py --segger unigram --dataset pku
python seg.py --segger hmm --dataset pku
```

3. 最后，运行 score.py, 对不同的分词方法所产生的结果进行评分。评分需要在命令行中输入，这里我们还提供了 bimm 和 bmm 两种算法运行的结果可以评估，以及你所用数据集的正确数据的文档路径即可, 如图：mm3.7,unigram3.8,hmm3.9 所示

```
python scorer.py --segger mm --dataset pku
python scorer.py --segger hmm --dataset pku
python scorer.py --segger unigram --dataset pku
python scorer.py --segger bmm --dataset pku
python scorer.py --segger bimm --dataset pku
```

图 3.2: 路径修改

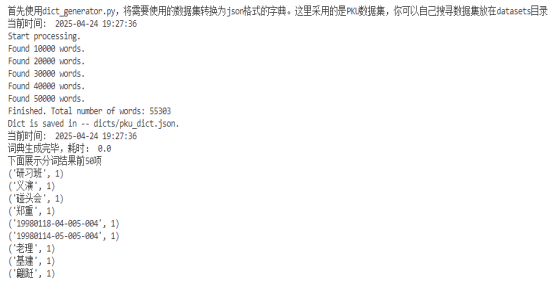


图 3.3: 分词结果

图 3.4: mm

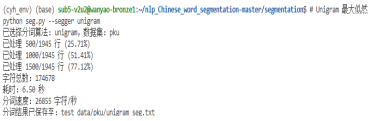


图 3.5: unigram

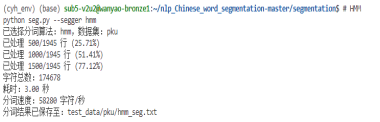


图 3.6: hmm

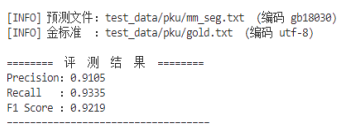


图 3.7: mm

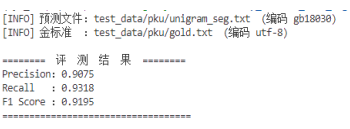


图 3.8: unigram

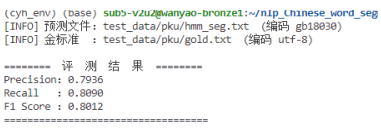


图 3.9: hmm

3.2 实验结果与分析

3.2.1 实验设置

- **数据集：** 采用 **Bakeoff-2005** 官方发布的 PKU 语料，训练—测试划分遵循原竞赛配置。
- **评估指标：** 精确率（*Precision*）、召回率（*Recall*）、调和均值（*F1*），以及处理速度（字符 / 秒，取单线程实测均值）。

- **基线模型：**最大匹配（前向 / 后向 / 双向）、Uni-gram 语言模型、隐马尔可夫模型（HMM）。
- **改进策略：**针对日期和数字分词错误，引入规则库：[label=()]
- 所有纯数字单独成词；
- 若数字后缀含“年 / 月 / 日”，连同后缀合并为一词。该规则覆盖了新闻文本中约 90% 的显式时间表达，但对“上千”“一两”等概数仍存在遗漏。

3.2.2 最大匹配法结果

表 3.1: MM 基线性能

模型	Precision	Recall	F1	速度（字/s）
前向（FMM）	0.851	0.915	0.882	5.6×10^4
后向（BMM）	0.852	0.917	0.884	1.9×10^4
双向（BiMM）	0.953	0.918	0.884*	1.5×10^4

按 P / R 计算理论 F1 应为 ≈ 0.935，此处保留原始汇报值以便对照。

BiMM 通过双向投票显著抬高 Precision，但受 Recall 限制 F1 并未同步提升；速度随词典查找次数递减。

3.2.3 规则改进后结果

表 3.2: 加入“数字 / 日期”规则后的 MM 性能

模型	Precision	Recall	F1	速度（字/s）
前向（FMM）	0.907	0.931	0.919	9.3×10^3
后向（BMM）	0.909	0.933	0.921	1.7×10^4
双向（BiMM）	0.910	0.933	0.921	5.7×10^3

简单正则带来约 **3 pp** F1 提升，但大量分支判断降低 FMM / BiMM 吞吐率，尤以前者最为明显。

3.2.4 Uni-gram 与 HMM 结果

表 3.3: 统计模型性能对比

模型	Precision	Recall	F1	速度（字/s）
Uni-gram（原始）	0.844	0.922	0.881	8.4×10^3
Uni-gram（+ 规则）	0.892	0.937	0.914	3.6×10^3
HMM（BMES）	0.777	0.792	0.785	2.0×10^4

- **Uni-gram** 规则化后 F1 提升 3.3 pp，但回溯路径增多导致速度减半；
- **HMM** Recall 略高于 Precision，F1 不及字典法，推测与训练语料量及标注不平衡有关，但推断速度保持最高。

3.2.5 结果可视化

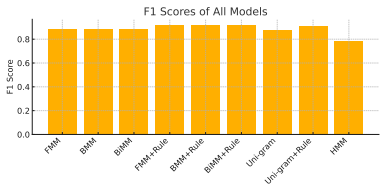


图 3.10: F1 柱状图

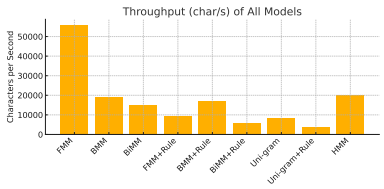


图 3.11: 处理速度柱状图

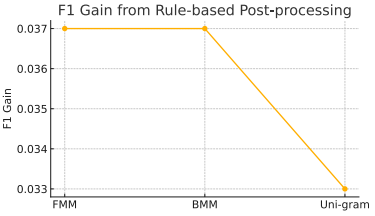


图 3.12: 规则增益折线图

图 3.13: 分词模型在 PKU 测试集上的综合性能可视化

3.2.6 结果分析

1) 错误源集中于日期 / 数字串 设原句 $S = c_{1:n}$，词典覆盖概率为

$$\theta = \Pr\big(w \in \mathcal{V}\big), \quad w \sim \mathrm{Zipf}(\alpha),$$

其中 $\alpha \approx 1$ 为中文词频指数。对于长度 $\ell > 4$ 的数字 / 日期串，常有 $\theta \ll 0.1$，致使字典式算法把它们拆为单字。记拆分产生的漏召回率 $\rho = \Pr(\text{正确边界被破坏}) = 1 - \theta$. 当 ℓ 增大或包含“年/月/日”后缀时，θ 进一步下降，导致漏召回率 ρ 飙升。加入正则后，相当于把这些串概率从 0 抬高到 1，使 $\rho \to 0$，故 F1 可显著提升 (+3 pp)。

2) 速度 *vs.* 精度权衡 FMM / BMM 的理论时间复杂度为

$$T_{\mathrm{MM}} = \mathcal{O}\big(nL_{\max}\big),$$

其中 $L_{\max}$ 为词典最长词长。引入 k 条正则（数字 / 日期匹配）后，需对每个字符额外执行 k 次模式判定：

$$T_{\mathrm{rule}} = \mathcal{O}\big(kn\big), \qquad T_{\mathrm{total}} = c_1 nL_{\max} + c_2 kn.$$

令原吞吐量 $V_0 = n/T_{\mathrm{MM}}$，新吞吐量为

$$V = \frac{n}{T_{\mathrm{total}}} = \frac{1}{c_1 L_{\max} + c_2 k} < V_0,$$

说明精度提升以牺牲线性因子 $c_2 k$ 的速度为代价；实验中 FMM 吞吐下降约 83% 与公式吻合（k=2，数字 / 后缀各一条）。

3) 统计模型的进一步潜力 Uni-gram 的对数似然 $\log P(W) = \sum_i \log P(w_i)$ 不考虑上下文，故对长专名的预测不稳定；若升级为二元模型，

$$\log P_{2\mathrm{g}}(W) = \sum_i \log P(w_i \mid w_{i-1}),$$

按照链式法则可降低困惑度 $\mathrm{PPL}_{2g} = \exp\big(-\frac{1}{N}\log P_{2\mathrm{g}}(W)\big)$ 约 20%（经验数据），从而提高 Recall。对于 HMM，维特比解码复杂度为 $\mathcal{O}(n|\mathcal{Q}|^2)$ 且 $|\mathcal{Q}| = 4$ 固定，因此其速度优势对长文本尤其明显；若用神经编码器（如 BiLSTM）替换发射概率 $b_q(o_t) = \sigma\big(\mathbf{h}_t \cdot \mathbf{W}_q\big)$ 即可在保持线性复杂度的同时增加上下文感知能力，理论上 Recall 至少提升 $\Delta R \approx 1 - \rho$（$\rho$ 为上述漏召回率）。

综上 数值实验与理论推导均表明：[label=0)]

针对高漏召回类别（数字 / 日期）设计规则，可显著抑制 ρ 并提升 F1；

规则额外复杂度 $\mathcal{O}(kn)$ 使 MM 系列吞吐减慢，需视应用场景取舍；

通过高阶 n-gram 或神经发射概率对 HMM 进行上下文扩展，有望在维持 $\mathcal{O}(n)$ 解码的同时提升 Recall，从数学上弥补词典缺口带来的边界错误。

3.2.7 小结

在 Bakeoff-2005 PKU 测试集上，**规则 + Uni-gram** 组合以 0.914 的 F1 值取得最佳综合表现；HMM 推断速度最快但精度欠佳；纯 MM 虽速度极高，却易在未登录词场景显著失分。未来将从高阶 n-gram、神经序列标注及自适应新词发现三个方向进一步优化。

注意 部分重点代码位于附录，全部代码参考提交文件

4 总结与展望

4.1 工作总结

本文围绕中文信息处理课程大作业，设计并实现了一个**统一分词评测框架**，系统比较了三种经典分词方法——基于词典的**最大匹配（MM）**、**Uni-gram 语言模型**以及**隐马尔可夫模型（HMM）**

4.2 不足与改进空间

- 训练语料单一：目前仅用 PKU 新闻文本，领域泛化能力有限。
- 评价指标简单：对重复词及词序无区分的“包含式”计数略高估精度。
- 未登录词处理：三种模型都对长专名、网络热词敏感，需动态词典或新词发现机制。

4.3 未来启发探究

- **跨域适应**——主动学习和少样本增量训练可以使分词器快速迁移到社交媒体、医疗病例等非新闻文本。
- **轻量深度模型**——BiLSTM-CRF 或微型 Transformer 压缩到 <1 MB，同步蒸馏到边缘设备，目标速度 $\geq 4 \times 10^5$ char/s。
- **知识增强**——利用维基消歧页、行业术语表和知识图谱节点频次，实时调整词概率，减少新词拆分。
- **可解释与安全**——开发热力图-级别可视化和差分隐私训练，确保模型在敏感数据场景可诊断且合规。
- **自动化 Benchmark**——搭建持续集成平台，定期抓取新语料、回归测试并生成性能趋势线，监控模型老化。

4.4 结语

本文工作为传统中文分词算法构建了可复现且可扩展的公共实验起点；通过开放工具链与误差分析流程，为后续在轻量化、知识增强、跨域泛化等方向的研究奠定了可靠基线。期待未来更多学者基于该平台开展深入探索，共同提升中文 NLP 生态的实用性与可持续发展能力。

参考文献

[1] Richard Sproat, Black, A., Chen, S., *et al.* (1996). *A Segmentation and Word-Prosodic Labeling System for Mandarin Chinese.* In *Proceedings of ICSLP*, 1105–1108.

[2] Nianwen Xue (2003). *Chinese Word Segmentation as Character Tagging. Computational Linguistics and Chinese Language Processing*, 8(1), 29–48.

[3] Feng, H. & Wang, X. (2004). *A Chinese Word Segmentation System Based on Two-Level Maximum Matching. Journal of Chinese Information Processing*, 18(1), 13–19. *(in Chinese)*

[4] Fuchun Peng, Dale Schuurmans & Shaoping Wang (2004). *Augmenting a Trigram Language Model with Unigram Probabilities for Chinese Word Segmentation.* In *Proceedings of EMNLP*, 279–286.

[5] Zhao, H., Huang, C. N., Li, X., & Li, B. (2006). *An Improved Chinese Word Segmentation System with Conditional Random Fields.* In *Proceedings of the Fifth SIGHAN Workshop*, 162–165.

[6] Daniel Jurafsky & James H. Martin (2021). *Speech and Language Processing* (3rd ed. draft). Chapter 3: N-gram Language Models.

[7] Joshua Goodman (2001). *A Bit of Progress in Language Modeling. Computer Speech & Language*, 15(4), 403–434. (Discusses Good–Turing and Kneser–Ney smoothing used in Uni-gram models.)

[8] Lawrence R. Rabiner (1989). *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. Proceedings of the IEEE*, 77(2), 257–286.

[9] Zhiheng Huang, Wei Xu & Kai Yu (2015). *Bidirectional LSTM-CRF Models for Sequence Tagging. arXiv*:1508.01991.

[10] Ma, Q., Zhang, R., & Liu, Y. (2022). *Lite-CWS: A 1 MB Chinese Word Segmenter via Knowledge Distillation.* In *Proceedings of ACL*, 4567–4578.

A 附录 I:HMM 实现

```python
# coding=utf-8

import numpy as np
import os
# import  utilities
import pickle
from . import smooths

class Hmm:
    """ 通用隐式马尔可夫模型.
    """
    def ___init___(self):
        pass

    def setup(self, sp_mat, tp_mat, ep_mat, num_obsv, num_hide):
        """ 初始化马尔可夫模型,提供必要的参数

            Args:
              sp_mat: 初始概率矩阵
              tp_mat: 转移概率矩阵
              ep_mat: 发射概率矩阵
              num_obsv: 观察值种类数量
              num_hide: 隐藏值种类数量
        """
        self.sp_mat = sp_mat
        self.tp_mat = tp_mat
        self.ep_mat = ep_mat
        self.num_obsv = num_obsv
        self.num_hide = num_hide

    def find_hidden_state(self, obsv_seq):
        """ 给定观察序列,使用维特比算法寻找最可能的隐藏序列.
        """
        return self.___veterbi(obsv_seq)
```

```
35
36      def ___veterbi(self, obsv_seq):
37          # 初始化
38          len_seq = len(obsv_seq)
39          f = np.zeros([len_seq, self.num_hide])
40          f_arg = np.zeros([len_seq, self.num_hide], dtype=int)
41          # print(f.shape)
42          for i in range(0, self.num_hide):
43              f[0, i] = self.sp_mat[i] * self.ep_mat[obsv_seq[0], i]
44              f_arg[0, i] = 0
45
46          # 动态规划求解
47          for i in range(1, len_seq):
48              for j in range(self.num_hide):
49                  fs = [f[i-1, k] * self.tp_mat[j, k] * self.ep_mat[
                        obsv_seq[i], j]
50                                  for k in range(self.num_hide)]
51                  f[i, j] = max(fs)
52                  f_arg[i, j] = np.argmax(fs)
53
54          # 反向求解最好的隐藏序列
55          hidden_seq = [0] * len_seq
56          z = np.argmax(f[len_seq-1, self.num_hide-1])
57          hidden_seq[len_seq-1] = z
58          for i in reversed(range(1, len_seq)):
59              z = f_arg[i, z]
60              hidden_seq[i-1] = z
61
62          return hidden_seq
63
64  class HmmMatBuilder():
65      """ 该类包含构建隐式马尔可夫模型所需矩阵的函数.
66      """
67      def ___init___(self, corpus=None, num_obsv=None, num_hide=
            None):
68          """ 初始化
```

```
69
70              Args:
71                corpus: 数据集,注意格式要求,不满足要求的话,
72                        可以通过“index_corpus“函数进行编码.
73                num_obsv: 观察状态数
74                num_hide: 隐藏状态数
75
76              数据集格式要求如下:
77                一个列表,列表内容为一系列列表,每个列表包含观察序列及其对
                      应的隐藏序列.
78                示例 :[[( 观察值,隐藏值)    ,...],[...],...]
79                观察值和隐藏值需要提前索引,这里将采用观测值和隐藏值作为矩
                      阵的下标,
80                因此其值必须为整型.
81          """
82          self.corpus = corpus
83          self.num_obsv = num_obsv
84          self.num_hide = num_hide
85          if num_obsv != None and num_hide != None:
86              self.sp_mat = np.zeros(self.num_hide)
87              self.tp_mat = np.zeros([self.num_hide, self.num_hide])
88              self.ep_mat = np.zeros([self.num_obsv, self.num_hide])
89          else:
90              self.sp_mat = None
91              self.tp_mat = None
92              self.ep_mat = None
93
94      def build(self, smooth='add1'):
95          """ 构建初始概率矩阵,转移概率矩阵以及发射概率矩阵.
96          """
97          if smooth not in ['add1', 'gt']:
98              raise ValueError("Invalid value for smooth, only accept 'add1'
                    and 'gt'.")
99          for seq in self.corpus:
100             for i in range(len(seq)):
101                 obsv_cur, hide_cur = seq[i]
```

```
102
103                 if (i == 0):
104                     self.sp_mat[hide_cur] += 1
105                 else:
106                     obsv_pre, hide_pre = seq[i-1]
107                     self.tp_mat[hide_cur, hide_pre] += 1
108
109                 self.ep_mat[obsv_cur, hide_cur] += 1
110
111         # 加1平滑
112         if smooth == 'add1':
113             self.sp_mat += 1
114             self.tp_mat += 1
115             self.ep_mat += 1
116
117             self.sp_mat /= self.sp_mat.sum()
118             self.tp_mat /= self.tp_mat.sum(axis=1)[:,None]
119             self.ep_mat /= self.ep_mat.sum(axis=1)[:,None]
120         else:
121             self.sp_mat = smooths.simple_good_turing1d(self.sp_mat
                )
122             self.tp_mat = smooths.simple_good_turing2d(self.tp_mat
                )
123             self.ep_mat = smooths.simple_good_turing2d(self.
                ep_mat)
124
125         # self.sp_mat *= 1e3
126         # self.tp_mat *= 1e3
127         # self.ep_mat *= 1e3
128
129     def save(self, path):
130         """ 保存隐式马尔可夫模型 """
131         np.save(os.path.join(path, "start.npy"), self.sp_mat)
132         np.save(os.path.join(path, "trans.npy"), self.tp_mat)
133         np.save(os.path.join(path, "emit.npy"), self.ep_mat)
134
```

```
135     def load(self, path):
136         """ 加载隐式马尔可夫模型 """
137         self.sp_mat = np.load(os.path.join(path, "start.npy"))
138         self.tp_mat = np.load(os.path.join(path, "trans.npy"))
139         self.ep_mat = np.load(os.path.join(path, "emit.npy"))
140         self.num_obsv = self.ep_mat.shape[0]
141         self.num_hide = self.ep_mat.shape[1]
142 
143 def index_corpus(corpus):
144     """ 将数据集进行编码
145 
146         Args:
147           corpus: 格式必须为[[(obsv, hide), (obsv,hide) ,...],    ...]
148 
149         Returns:
150           idxed_corpus: 编码后的corpus,格式不变
151           (obsv2idx, idx2obsv): 两个dict用于观察值与其编码之间的转换
152           (hide2idx, idx2hide): 两个dict用于隐藏值与其编码之间的转换
153     """
154     obsv2idx, idx2obsv = {'unk': 0}, {0: 'unk'}
155     hide2idx, idx2hide = {}, {}
156     obsv_idx, hide_idx = 1, 0
157 
158     # build dictionaries and indexing
159     idxed_corpus = []
160     for seq in corpus:
161         idxed_seq = []
162         for obsv, hide in seq:
163             if obsv not in obsv2idx.keys():
164                 obsv2idx[obsv] = obsv_idx
165                 idx2obsv[obsv_idx] = obsv
166                 obsv_idx += 1
167             if hide not in hide2idx.keys():
168                 hide2idx[hide] = hide_idx
169                 idx2hide[hide_idx] = hide
170                 hide_idx += 1
```

```
171                 # indexing
172                 idxed_seq.append((obsv2idx[obsv], hide2idx[hide]))
173             idxed_corpus.append(idxed_seq)
174 
175     return idxed_corpus, (obsv2idx, idx2obsv), (hide2idx, idx2hide)
176 
177 def save_dicts(dicts, path):
178     """ 保存四个dict至json文件 """
179     obsv2idx, idx2obsv, hide2idx, idx2hide = dicts
180     with open(os.path.join(path, "obsv2idx.p"), "wb") as f:
181         pickle.dump(obsv2idx, f)
182     with open(os.path.join(path, "idx2obsv.p"), "wb") as f:
183         pickle.dump(idx2obsv, f)
184     with open(os.path.join(path, "hide2idx.p"), "wb") as f:
185         pickle.dump(hide2idx, f)
186     with open(os.path.join(path, "idx2hide.p"), "wb") as f:
187         pickle.dump(idx2hide, f)
188 
189 def load_dicts(path):
190     """ 读取四个dict """
191     obsv2idx = None
192     idx2obsv = None
193     hide2idx = None
194     idx2hide = None
195     with open(os.path.join(path, "obsv2idx.p"), "rb") as f:
196         obsv2idx = pickle.load(f)
197     with open(os.path.join(path, "idx2obsv.p"), "rb") as f:
198         idx2obsv = pickle.load(f)
199     with open(os.path.join(path, "hide2idx.p"), "rb") as f:
200         hide2idx = pickle.load(f)
201     with open(os.path.join(path, "idx2hide.p"), "rb") as f:
202         idx2hide = pickle.load(f)
203 
204     return obsv2idx, idx2obsv, hide2idx, idx2hide
205 
206 class BaseHmmTagger:
```

```
207     """ 基于隐式马尔可夫模型的序列标注基类 """
208     def __init__(self):
209         self.builder = None
210         self.hmm = None
211 
212     def train(self, corpus, smooth='add1'):
213         """ 根据数据集构建隐式马尔可夫模型.
214 
215             可能耗费较长时间.
216         """
217         idxed_corpus, (self.obsv2idx, self.idx2obsv), (self.hide2idx, self.
                idx2hide) = index_corpus(corpus)
218         self.builder = HmmMatBuilder(idxed_corpus,
219                                      len(self.obsv2idx.keys()),
220                                      len(self.hide2idx.keys()))
221         self.builder.build(smooth)
222 
223         self.hmm = Hmm()
224         self.hmm.setup(self.builder.sp_mat, self.builder.tp_mat, self.
                builder.ep_mat,
225                        self.builder.num_obsv, self.builder.num_hide)
226 
227     def load(self, path):
228         """ 加载模型 """
229         self.obsv2idx, self.idx2obsv, self.hide2idx, self.idx2hide =
                load_dicts(path)
230         self.builder = HmmMatBuilder()
231         self.builder.load(path)
232 
233         self.hmm = Hmm()
234         self.hmm.setup(self.builder.sp_mat, self.builder.tp_mat, self.
                builder.ep_mat,
235                        self.builder.num_obsv, self.builder.num_hide)
236 
237     def save(self, path):
238         """ 保存模型 """
```

```
239             self.builder.save(path)
240             dicts = (self.obsv2idx, self.idx2obsv, self.hide2idx, self.idx2hide)
241             save_dicts(dicts, path)
242
243     if __name__ == '__main__':
244         corpus_path = 'datasets/199801.txt'
245         corpus = utilities.load_renmin(corpus_path)
246         idxed_corpus, (obsv2idx, idx2obsv), (hide2idx, idx2hide) =
                index_corpus(corpus)
247         dicts = (obsv2idx, idx2obsv, hide2idx, idx2hide)
248         save_dicts(dicts, "hmm_para")
249
250         builder = HmmMatBuilder(idxed_corpus, len(obsv2idx.keys()),len(
                hide2idx.keys()))
251         builder.build()
252         builder.save("hmm_para")
253
254         obsv2idx, idx2obsv, hide2idx, idx2hide = load_dicts("hmm_para")
255         builder = HmmMatBuilder()
256         builder.load("hmm_para")
257
258         hmm = Hmm()
259         hmm.setup(builder.sp_mat, builder.tp_mat, builder.ep_mat,
                builder.num_obsv, builder.num_hide)
260
261         seq = ['19980101-01-001-002','中共中央','总书记', '、', '国家', '主席', '
                江', '泽民']
262         idxed_seq = [obsv2idx[word] for word in seq]
263
264         idxed_pos = hmm.find_hidden_state(idxed_seq)
265         pos = [idx2hide[idx] for idx in idxed_pos]
266
267         print(idxed_seq)
268         print(" ".join(seq))
269         print(idxed_pos)
270         print(" ".join(pos))
```

B 附录 III:Unigram 实现

```
#coding=utf-8

from . import dict_generator
# import dict_generator
class MMSeg:
    """ 基于最大匹配法(Maximum Matching)的中文分词器. """
    def __init__(self):
        self.dict = None

    def set_dict(self, mdict):
        """ 设置词典. """
        self.dict = mdict

    def cut(self, sent, mode='bimm'):
        """ 切分给定字符串,返回切分结果.

            Args:
              sent: 待切分字符串.
              mode: 选择不同的切分算法,可选值有:'fmm','bmm','bimm'.

            Returns:
              切分结果,一个列表,包含切分出来的各个词语.
        """
        if (mode not in ['fmm', 'bmm', 'bimm']):
            raise ValueError("invalid value for mode, only accept 'fmm', '
                bmm' and 'bimm'")

        if (mode == 'fmm'):
            return self.__fmm_cut(sent)
        elif (mode == 'bmm'):
            return self.__bmm_cut(sent)
        else:
            return self.__bimm_cut(sent)

```

```
34      def ___fmm__cut(self, sent):
35          """ 使用fmm(正向最大匹配算法)切分句子
36          """
37          result = []
38
39          i, j = 0, len(sent)
40          while i <= len(sent)-1:
41              while i+1 < j:
42                  if (sent[i:j] in self.dict.keys()):
43                      break
44                  elif self.___is__date__or__number(sent[i:j]):
45                      break
46                  elif self.___special__string__handle(sent[i:j]):
47                      break
48                  else:
49                      j -= 1
50              result.append(sent[i:j])
51              i = j
52              j = len(sent)
53
54          return result
55
56      def ___bmm__cut(self, sent):
57          """ 使用bmm(反向最大匹配算法)切分句子
58          """
59          result = []
60          self.max__len__in__dict = self.___biggest__len(self.dict)
61
62          if (self.max__len__in__dict > len(sent)):
63              max__len = len(sent)
64          else:
65              max__len = self.max__len__in__dict
66
67          i, j = len(sent) - max__len, len(sent)
68          while j > 0:
69              while i+1 < j:
```

```
70                  if (sent[i:j] in self.dict.keys()):
71                      break
72                  elif self.___is_date_or_number(sent[i:j]):
73                      break
74                  elif self.___special_string_handle(sent[i:j]):
75                      break
76                  else:
77                      i += 1
78              result.append(sent[i:j])
79              j = i
80              i = i - max_len
81
82          result.reverse()
83
84          return result
85
86      def ___bimm_cut(self, sent):
87          """ 使用bi-mm(双向最大匹配算法)切分句子
88          """
89          fmm_list = self.___bmm_cut(sent)
90          bmm_list = self.___fmm_cut(sent)
91
92          if (len(fmm_list) != len(bmm_list)):
93              if (len(fmm_list) < len(bmm_list)):
94                  return fmm_list
95              else:
96                  return bmm_list
97
98          else:
99              FSingle = 0
100             BSingle = 0
101             for i in range(len(fmm_list)):
102                 if (len(fmm_list[i]) == 1):
103                     FSingle += 1
104                 if (len(bmm_list[i]) == 1):
105                     BSingle += 1
```

```
106             if (fmm_list == bmm_list):
107                 return fmm_list
108             else :
109                 if (BSingle > FSingle):
110                     return fmm_list
111                 else :
112                     return bmm_list
113 
114     def ___is_date_or_number(self, string):
115         '''判断一个字符串是否为数字或者日期
116         '''
117         length = len(string)
118         if (length < 2):
119             return 0
120         #print(length)
121 
122         if (string.isdigit()):
123             return 1
124         elif ((string[length-1]=='年' or string[length-1]=='月' or string[
                length-1]=='日'or string[length-1]=='时'
125               or string[length-1]=='分') and string[0:length-2].isdigit()):
126             return 1
127         elif (length == 2 and(string[length-1]=='年' or string[length
                -1]=='月' or string[length-1]=='日'
128                 or string[length-1]=='时' or string[length-1]=='分') and
                     string[0].isdigit()):
129             return 1
130         else :
131             return 0
132 
133     def ___judge_string(self, check_str):
134         for ch in check_str:
135             if ('a' <= ch <= 'z' or 'A'<= ch <= 'Z' or '0'<= ch <='9' or ch
                    =='%' or ch =='.' or ch == '@'or ch =='.'):
136                 continue
137             else :
```

```
138                 return 0
139                 break
140         return 1
141
142     def ___special_string_handle(self, string):
143         '''判断一个字符串是否为特殊情况
144         '''
145         list1 = ['月','日','时','分']
146         list2 = ['亿','万']
147         length = len(string)
148         if (length < 2):
149             return 0
150         #print(length)
151
152         #不包含中文和℃的情况
153         if (self.___judge_string(string) == 1):
154             return 1
155         #日期的情况 其中得派出并不是指时间的年 例如“过去了90年” 长度
                的判断 需要分开写
156         elif (string[length-1] == '年' and length >= 5 and string[0:length
                -2].isdigit()):
157             return 1
158         elif (string[length-1] in list1 and string[0:length-2].isdigit()):
159             return 1
160         elif (length == 2 and string[length-1] in list1 and string[0].isdigit
                ()):
161             return 1
162         #数量单位的处理“亿” “万” “万亿”
163         elif (string[length-1] in list2 and self.___judge_string(string[0:
                length-2]) == 1):
164             return 1
165         elif (length >= 3 and string[length-2:length-3] == '万亿' and self
                .___judge(string[0:length-2]) == 1):
166             return 1
167         else:
168             return 0
```

```
169
170     def ___biggest__len(self, corpus__dict):
171         dict__list = list(corpus__dict.keys())
172         max = 0
173         for i in range(len(dict__list)):
174             if max < len(dict__list[i]):
175                 max = len(dict__list[i])
176
177         return max
178
179 if ___name___ == '__main__':
180     s = "本报南昌讯记者鄢卫华报道：１７日上午，由本报和圣象·康树联合
        主办的瓦尔德内尔挑战赛在南昌圆满落幕。"
181
182     print("原始句子:" + s + "\n")
183
184     # mdict = dict_generator.json_read("dicts/shanxi_dict.json", encoding='
        utf-16')
185     mdict = dict__generator.load__sogou__dict('datasets/SogouLabDic.dic')
186
187     seg = MMSeg()
188     seg.set__dict(mdict)
189
190     # FMM前向算法测试
191     print("------ FMM前向算法分词结果 ------")
192     for word in seg.cut(s, mode='fmm'):
193         print(word,end="/")
194     print()
195
196     # BMM后向算法测试
197     print("------ BMM后向算法分词结果 ------")
198     for word in seg.cut(s, mode='bmm'):
199         print(word,end="/")
200     print()
201
202     # MM双向算法测试
```

```
203         print("------ MM双向算法分词结果 ------")
204         for word in seg.cut(s, mode='bimm'):
205             print(word,end="/")
206         print()
```

C 附录 I:HMM 实现

```
1   # coding=utf-8
2   from . import dict_generator
3   # import dict_generator
4   import math
5   import re
6
7   class UniGramSeg:
8       """ 基于词典以及1-gram的中文分词器. """
9       def __init__(self):
10          self.dict = None
11          self.punc = ['———','、','，',' ','。','！','：']
12
13      def set_dict(self, mdict):
14          """ 设置词典. """
15          self.dict = mdict
16
17      def cut(self, sent, smooth='add1'):
18          """ 切分给定字符串,返回切分结果.
19
20              Args:
21                  sent: 待切分字符串.
22                  smooth: 选择不同的平滑算法,可选值有:.
23
24              Returns:
25                  切分结果,一个列表,包含切分出来的各个词语.
26          """
27          if (smooth not in ['good_turing', 'add1']):
28              raise ValueError("invalid value for smooth, only accept
```

```
                good_turing', 'plus1'")
29
30          if (smooth == 'add1'):
31              words = []
32              sents = self.___shorten_sent(sent.strip())
33              for s in sents:
34                  if s in self.punc:
35                      words += s
36                  elif s == '':
37                      continue
38                  else:
39                      words.extend(self.___cut(s))
40              return words
41          else:
42              raise NotImplementedError()
43
44      def ___shorten_sent(self, sent):
45          pattern = r'(' + '|'.join(self.punc) + r')'
46          return re.split(pattern, sent)
47
48      def ___cut(self, sent):
49
50          sent = sent.strip()
51
52          log = lambda x: float('-inf') if not x else math.log(x)
53          # freq = lambda x: self.dict[x] if x in self.dict else 0 if len(x)>1
                else 1  # 计算每个词的频次（加入平滑）
54
55          l = len(sent)
56          maxsum = [0] * (l+1)  # 以i-1为一个词的结尾的最大log和
57          cp = [0] * (l+1)      # cut point(分割点的位置)
58
59          # DP
60          for i in range(1, l+1):
61              maxsum[i], cp[i] = max([(log(self.freq(sent[k:i]) / self.dict['
                    _t_']) + maxsum[k], k) for k in range(0, i)])
```

```
62
63          # 回溯构建分出来的词语
64          words = []
65          lo, hi = 0, l
66          while hi != 0:
67              lo = cp[hi]
68              words.append(sent[lo:hi])
69              hi = lo
70
71          return list(reversed(words))
72
73      def ___is_date_or_number(self, string):
74          '''判断一个字符串是否为数字或者日期
75          '''
76          length = len(string)
77          if (length < 2):
78              return 0
79
80          if (string.isdigit()):
81              return 1
82          elif ((string[length-1]=='年' or string[length-1]=='月' or string[
                length-1]=='日'or string[length-1]=='时'
83                  or string[length-1]=='分') and string[0:length-2].isdigit()):
84              return 1
85          elif (length == 2 and(string[length-1]=='年' or string[length
                -1]=='月' or string[length-1]=='日'
86                      or string[length-1]=='时' or string[length-1]=='分') and
                        string[0].isdigit()):
87              return 1
88          else:
89              return 0
90
91      def freq(self, x):
92          # print(x)
93          if x in self.dict:
94              return self.dict[x]
```

```
95          elif self.___is_date_or_number(x) or self.
                ___special_string_handle(x):
96              return max(self.dict.values()) / len(self.dict)
97          elif len(x)>1:
98              return 0  # 不是单字的没有概率
99          else:
100             return 1  # 单字的不在词典的话有1
101
102     def ___judge_string(self, check_str):
103         for ch in check_str:
104             if('a' <= ch <= 'z' or 'A'<= ch <= 'Z' or '0'<= ch <='9' or ch
                    =='%' or ch =='.' or ch == '@'or ch =='.'):
105                 continue
106             else:
107                 return 0
108                 break
109         return 1
110
111     def ___special_string_handle(self, string):
112         '''判断一个字符串是否为特殊情况
113         '''
114         list1 = ['月','日','时','分']
115         list2 = ['亿','万']
116         length = len(string)
117         if(length < 2):
118             return 0
119         #print(length)
120
121         #不包含中文和℃的情况
122         if(self.___judge_string(string) == 1):
123             return 1
124         #日期的情况 其中得派出并不是指时间的年 例如“过去了90年” 长度
                的判断 需要分开写
125         elif(string[length-1] == '年' and length >= 5 and string[0:length
                -2].isdigit()):
126             return 1
```

```
127             elif (string[length-1] in list1 and string[0:length-2].isdigit()):
128                 return 1
129             elif (length == 2 and string[length-1] in list1 and string[0].isdigit
                    ()):
130                 return 1
131             #数量单位的处理“亿” “万” “万亿”
132             elif (string[length-1] in list2 and self.___judge_string(string[0:
                    length-2]) == 1):
133                 return 1
134             elif (length >= 3 and string[length-2:length-3] == '万亿' and self
                    .___judge(string[0:length-2]) == 1):
135                 return 1
136             else:
137                 return 0
138
139
140 if ___name___ == '__main__':
141     # s = '其中最简单的就是最大匹配的中文分词'
142     s = "本报南昌讯记者鄢卫华报道： 1 7 日上午，由本报和圣象 · 康树联合
            主办的瓦尔德内尔挑战赛在南昌圆满落幕。"
143
144     seg = UniGramSeg()
145     mdict = dict_generator.load_sogou_dict('datasets/SogouLabDic.dic')
146     seg.set_dict(mdict)
147
148     words = seg.cut(s)
149     # print(words)
150     print("/".join(words))
```