北京大学

PEKING UNIVERSITY

自然语言处理之中文分词

学	院：	软件与微电子学院
组	员 1：	罗旭坤 2001210368
组	员 2：	罗登 2001210364
组	员 3：	马心睿 2001210371
时	间：	2021 年 05 月 09 日

目录

第一节　实验目的

本次实验的目标是在给定给的中文分词语料数据集上，设计，训练，并测试中文分词模型，在训练集上进行训练和交叉验证，在测试集上评价模型的性能。提供的数据集来源于 SIGHAN 2005 第二届中文分词人工中的 Peking University 数据集，包含 `training.txt`、`test.txt`、`training_vocab.txt` 三个文件，均为全角字符，Utf-8 编码格式。该数据集的统计信息如下表所示：

训练集	测试集	Type 数	Token 数
19,056	1,944	55,303	1,109,947

数据集的格式描述为：

- 每行一个句子；
- 词与词、词与标点之间由两个空格分开。

数据集示例如下：

```
共同　创造　美好　的　新　世纪　——　二〇〇一年　新年　贺词
```

数据集包含 5 项评测指标，分别是：Precision，Recall，F1，In-Vocabulary Recall (IV Recall) 和 Out-of-Vocabulary Recall (OOV Recall)。并且提供了可以执行测试得出测评结果的 perl 脚本，使用命令

```
$ perl scripts/score training_vocab.txt test.txt predict.txt > score.txt
```

即可得到测评结果。

第二节　实验原理

中文分词方法主要分为两大类：基于词表匹配的分词方法和基于字序列标注的分词方法。

基于词表匹配的分词方法，系统的核心在于一个高质量、与分词任务匹配的词表，通过人工预先设定的匹配规则，对句子进行切分。这里常用到的方法有：最大匹配，正则匹配，基于词图的匹配等。本报告将在第一部分对该方法进行详细介绍。

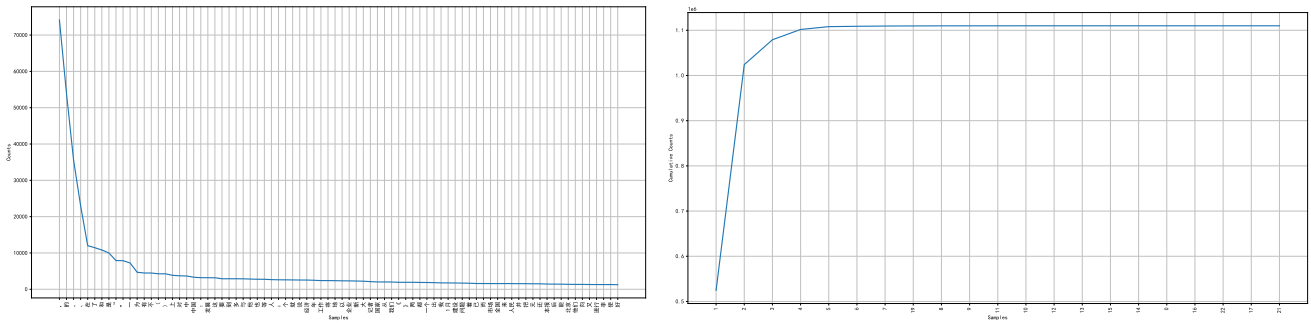
另外还有一大类是基于字序列标注的分词方法，该方法将分词任务视为一个序列标注任务，为句子中的每个字符，均提供一个标签（tag），算法输出一个与句子等长的标注序列。常用的字序列标注法有 4-tag 法、2-tag 法、6-tag 法等。这里以 4-tag 法为例进行介绍：4-tag 法即为每个字符提供一个 BMES 标注，其中 B 表示该字符是一个词的开始（Begin of the word），M 表示该字符在一个词的中间，E 表示该字符是一个词的结尾，S 表示该字符单独成词。其他标注方法与此类似。

需要注意的是，不是所有的标注序列都是合理的，例如 B 后面不可能是 S、E 后面不可能是 M 等，算法在生成或输出标注序列的时候需要正确处理不合法状态。

可以应用于序列标注问题的算法有很多，有传统的机器学习算法和目前较为热门的深度学习算法。例如隐马尔科夫（HMM），最大熵模型（MaxEnt），条件随机场（CRF[1]），结构化感知机器（Structured Perceptron[2]）等属于传统的机器学习算法，将在第二部分进行介绍。基于深度学习的算法，常采用深层神经网络模型，得到字符的向量表示[3]，再通过编码层例如 RNN、LSTM[4] 等，自动提取特征，在近几年 NLP 任务中得到了广泛的应用。由于该部分内容较多，本实验报告将在第三小节介绍其应用在中文分词任务中的主要流程，以及我们的设计思路。

第三节　数据集分析与统计

了解数据集的统计信息、数据分布，是开发和应用算法模型的第一步。我们首先对数据集进行了一定的探索性分析，绘制了训练集中的词频信息、词长度信息，见图1。通过对词频和词长度的分析



词频最高的前 80 个词　　训练集词长度分布（累加概率）

图 1: 数据集的探索性分析

通过图示可以清楚地看到，训练集的词频满足明显的长尾分布（Zif 定理）。对词长度的分析也能够发现，长度为小于等于 5 的词，已经占据了训练集词汇的 99.82%。因此在编写最大匹配算法时，最大长度可以选择为 4 到 5 即可，而不需要以 20（训练集中最长的词）为最大长度标准。

第四节　实验内容与结果对比

第一小节　基于词表匹配的算法

4.1.1　最大匹配算法

最大匹配算法（Max Match，简称 MM）是多种分词方法中实现起来最简单，但是效果也是十分惊人的一种算法。其采用了一种贪心的思想，基本思路是通过扫描句子，在其上按照最大长

度进行切分词语，如果该词语在词表中出现，则直接切分。若不出现，则缩短切分的长度，直至切分长度为 1。在实现细节上可以使用简单的双指针算法进行实现。

最大匹配算法根据其缩减长度的方向还可以分为前向最大匹配（FMM）和逆向最大匹配（RMM），其中 FMM 是指当切分词语不存在于词表时，将右侧指针向左侧移动一个字符。而 RMM 与此相反。实现时，扫描句子的方向也可以是不同的，可以从前向后扫描，也可以从后向前扫描。在本实验中，我们实现的 FMM 算法是从前向后扫描句子，而 RMM 算法是从后向前扫描句子。下面给出 FMM 函数的 Python 代码，见4.1.1。

最大前向匹配 fmm.py

```python
def max_forward(line:  str, max_len=4):
    res = []
    n =  len(line)
    idx = 0
    while idx < n:
        lens = max_len
        while lens > 0:
            sub = line[idx: idx + lens]
            if sub in vocab_set:
                res.append(sub)
                idx += lens
                lens = max_len
            else:
                lens -= 1
                if lens == 0:
                    idx += 1
    return res
```

该算法在最大长度设置为 5 时，能够在测试集达到 F1 为 0.899 的惊人效果，这里还测评了设置不同 maxlen 对最终评测结果的影响，可以看到，算法性能在 maxlen≥ 5 时基本不再变化，验证了初始的想法。但是 MM 算法的短板也是非常明显，OOV Recall 为 0，也就是说，最大匹配算法，完全没有识别未登录词的能力。可以说，该算法的运行效果，完全取决与词典的好坏。在 SIGHAN 2005 PKU 这个数据集上，语料来自于人民日报，训练集与测试集差异很小，因此能够取得这样的效果。但是在现实的应用场景中，例如互联网应用，每天都可能会有大量新词出现，最大匹配算法就可能难以奏效了。

另外，在实现的细节上，如何存储词表也是一个值得考量的问题。在本实验中，我们使用了 Set（集合）来实现。可以对其查找复杂度进行分析。设 n 为查询词的长度，N 为词表大小。Set 的底层一般采用平衡树（红黑树或 AVL 树等），因此查询一次的复杂度为 $O(\log N)$，另外字符

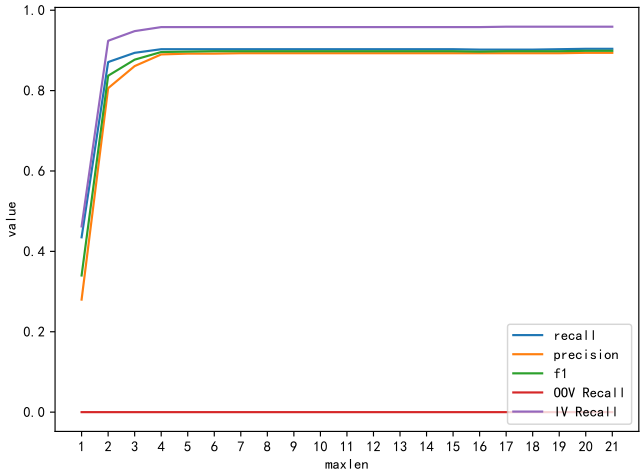


图 2: 最大长度对 MM 算法的影响

串匹配的效率为 $O(n)$，因此总的时间复杂度为 $O(n \log N)$。当然也可以采用基于 Hash 实现的 Set 或 Map 结构，其查找效率为 $O(n)$。在数据量比较小的情况，使用 Set 是简单高效的做法。而当数据规模非常庞大时，上述两种实现（平衡树和 Hash）都不高效。因为 $\log N$ 可能非常大，而 Hash 也可能会有大量冲突，存储效率也非常低下。在实际的应用中常使用 Trie（前缀树或字典树）[1]或 DAT[5]（Double Array Trie）在存储词表。Trie 能够支持快速地找到查询词在文本中出现的次数，查询效率为严格的 $O(n)$，也即仅与查询词的长度有关。而 DAT 是对 Trie 的一个优化，base 数组来存储状态转移情况，check 数组用来验证转移的有效性。为了对 Trie 做进一步压缩，使用 tail 数组存储无公共前缀的尾字符串，空间效率更高。图3给出了它们的一个示例，更多的内容可以查看参考文献。

4.1.2 基于词图的匹配算法

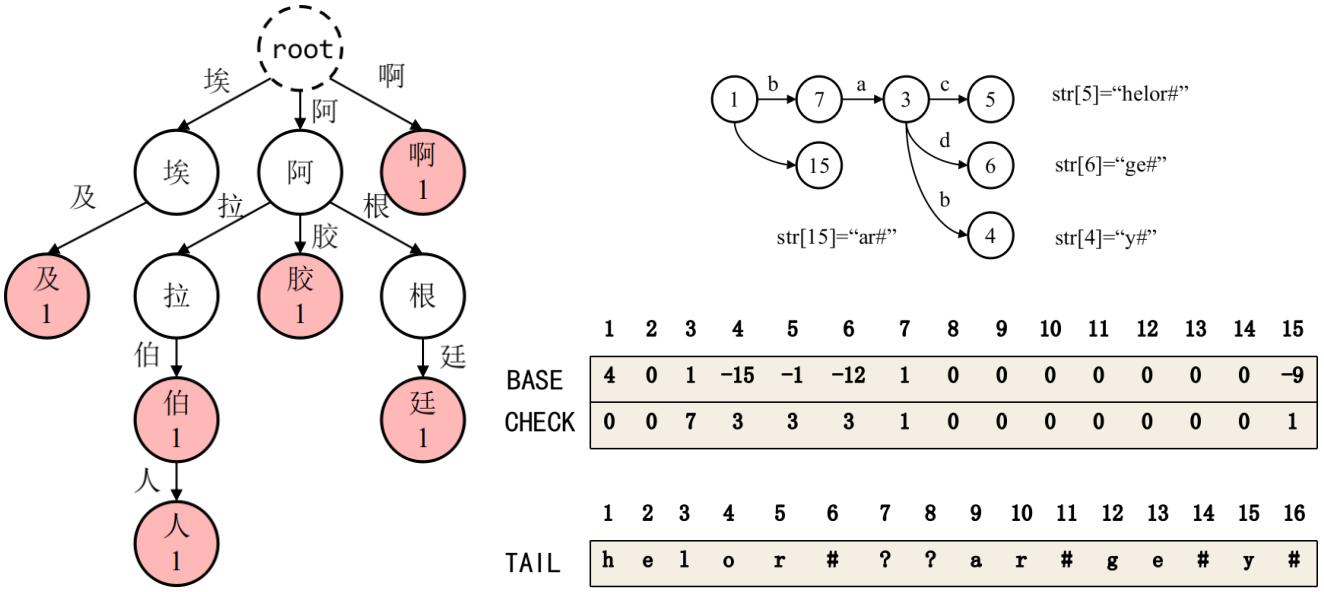
另外一种高效的匹配算法是基于词图的匹配。算法根据输入句子，构建有向无环图（DAG），图的节点表示句子中的字符，节点间有连边则表示词表中存在从开始节点到结尾节点之间字符构成的词汇。在词图上，除了可以记录字符的连接关系，还可以记录构成的词汇在词表中出现的频率信息。在进行分词时，就可以通过动态规划算法，逆向求解得到最优的转移路径，即为最佳分词结果。下面给出了一个词图的例子4。

输入句子为：“共同创造美好的新世纪”。图中节点连边表示从开始节点到结束节点间构成的词在词表中存在，并且在下面标记了其出现的概率（取对数）。连边上存储的概率信息可以是多样的，完全可以自定义概率，例如使用 Bigram 而非 Unigram，从而得到不同的结果。这个例子是参考了 jieba[2]的实现，其采用的是前缀字典，因此一些不在词表中出现的词（已有词的前缀）也可能存在一定的概率。

若使用 Python 语言的 Dict 数据结构实现 DAG 图，其完整的存储信息见下。

[1]力扣 208 实现 Trie：https://leetcode-cn.com/problems/implement-trie-prefix-tree/

[2]jieba 中文分词：https://github.com/fxsjy/jieba



1)Trie 的一个示例。输入为 [埃及，阿根廷，阿拉伯，阿胶，啊，阿拉伯人]

2)DAT 的一个示例。输入为 [bachelor, baby, badge, jar]

图 3: Trie 和 DAT 的图示

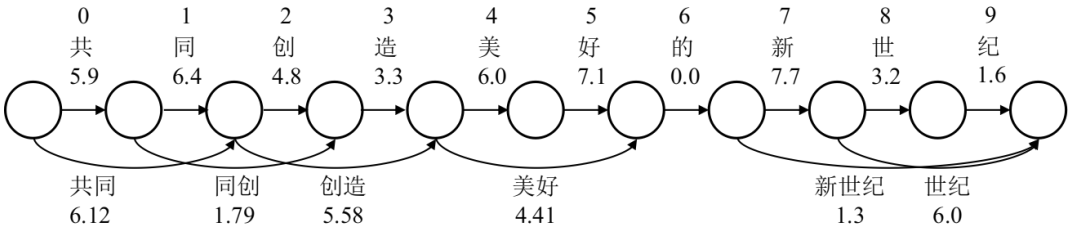


图 4: 词图构建示例

DAG 图的信息

```
0: [0, 1]        共, 共同
1: [1, 2]        同, 同创
2: [2, 3]        创, 创造
3: [3]           造
4: [4, 5]        美, 美好
5: [5]           好
6: [6]           的
7: [7, 9]        新, 新世纪
8: [8, 9]        世, 世纪
9: [9]           纪
```

实验结果对比：

表 1: 基于词表匹配的方法对比

序号	**匹配方法**	**maxlen**	P	R	F_1	R_{OOV}	R_{IV}
1	FMM	4	0.890	0.903	0.896	0.000	0.958
2	FMM	5	0.892	0.903	0.897	0.000	0.958
3	RMM	4	0.892	0.905	0.899	0.000	0.960
4	RMM	5	0.894	0.905	0.900	0.000	0.960
5	DAG	-	**0.881**	**0.924**	**0.902**	**0.193**	**0.968**

从表1中可以看出，基于词表匹配的算法，在测试集上均能取得不错的效果。最大匹配在长度设置为 4 或 5 时，F_1 值均达到了 0.895 以上。而采用 DAG 词图进行匹配，因为考虑了全局的概率转移信息，取得了最好的效果。同时，该方法的 OOV Recall 也不是 0，而是 0.193，原因是前缀字典引入了一些新词。另一种观点是，可以将基于 DAG 的方法视为是最大匹配 + 概率信息，因为如果不考虑概率信息，或者概率信息相同，那么词图匹配法就退化为最大匹配法了。

第二小节　基于字序列标注的机器学习算法

4.2.1　隐马尔科夫模型

隐马尔科夫模型（HMM）是非常经典的分词模型。HMM 是一种生成式模型（Generative Model），其核心观点是将输入句子，也就是字符序列，视为是观测序列 V，通过建模观测序列与状态转移序列 Q 的联合概率 $P(Q,V)$，最终求解状态转移序列[3]。图5是 HMM 在分词问题上的一个示例。

[3]这里的符号体系参考《统计学习方法》

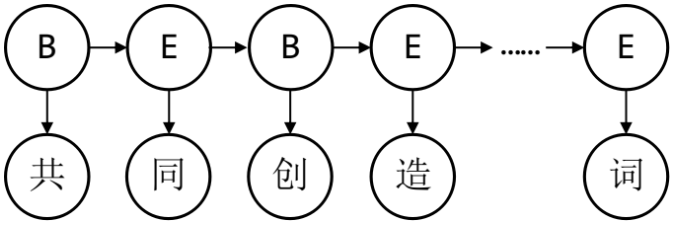


图 5: HMM 用于分词

HMM 的监督训练过程即为词频统计，通过遍历整个训练集中的句子，统计得到状态转移概率矩阵 A，发射概率矩阵 B 和初始状态 π。其中，状态转移概率矩阵即为 $P(x_i|x_{i-1})$，也就是二元语法概率，因此，某种观点上可以认为一阶 HMM 即为 Bigram Model。

得到上述 3 个概率矩阵 (A, B, π) 后，就可以通过 Viterbi 算法进行解码，得到最可能的状态转移序列。Viterbi 解码算法主要包含以下 4 个步骤：

1) 初始化

$$\delta_1(i) = \pi_i b_i(o_1), \quad i = 1, 2, \cdots, N$$

$$\Psi_1(i) = 0, \quad i = 1, 2, \cdots, N$$

2) 递推。对 $t = 2, 3, \cdots, T$

$$\delta_t(i) = \max_{1 \le j \le N}[\delta_{t-1}(j)a_{ji}]b_i(o_t), i = 1, 2, \ldots, N$$

$$\Psi_t(i) = \arg\max_{1 \le j \le N}[\delta_{t-1}(j)a_{ji}], i = 1, 2, \ldots, N$$

3) 终止

$$P^* = \max_{1 \le i \le N} \delta_T(i)$$

$$i_T^* = \arg\max_{1 \le i \le N} \delta_T(i)$$

4) 最优路径回溯。对 $t = T-1, T-2, \cdots, 1$

$$i_t^* = \Psi_{t+1}(i_{t+1}^*)$$

求得最优路径 $I^* = (i_1^*, i_2^*, \cdots, i_T^*)$。

在使用 HMM 进行分词时，存在以下几点可以优化的地方：

- 状态转移矩阵可以使用二维数组，获得更优地使用 Numpy 实现，但是由于这里状态数比较少，而且发射（emition）矩阵的维度难以确定，直接开辟一个容量非常大的二维数组又显得非常浪费空间。若是在计算 HMM 的前向或后向概率，因为存在大量相乘求和操作，转换为 Numpy 矩阵运算可以提升效率。但是在解码过程中，使用字典结构即可。

- 概率可以采用取 log 之后存储，这样就能将乘法转化为加法，提升运算效率，同时避免精度损失，甚至浮点溢出带来的风险。
- 在解码计算状态转移时，可以在迭代过程中进行限定，仅考虑合法状态，过滤非法状态，例如从 M 到 B 或 S 的状态序列均是不可能出现的，无需遍历和递归求解。
- 可以根据一些人工规则，例如标点符号，英文与中文一般不会连在一起等规则，将一个句子分为多个段（chunk），然后输入到 HMM 中进行处理，这样并不会影响 HMM 的运行，而且能够实现算法与规则的结合。但是规则的选取如果不恰当的话，可能带来反面的效果。
- HMM 分词法能够有效提升 OOV Recall，但是精确率很低，对很多词表内词汇也无法正确识别。可以考虑与词表匹配法结合，只有当判断该词未登录词的时候，才将其输入到 HMM 分词器中处理。该方法即为 jieba 的实现思路。

HMM 的分词的实验结果见表2。

表 2: HMM 分词器实验结果

HMM				P	R	F_1	R_{OOV}	R_{IV}
序号	人工规则	chunk	词表匹配					
1	否	否	否	0.715	0.745	0.730	0.315	0.771
2	否	是	否	0.760	0.784	0.772	**0.436**	0.805
3	是	否	否	0.716	0.746	0.731	0.318	0.772
4	是	是	否	0.716	0.746	0.731	0.318	0.772
5	否	是	是	**0.887**	**0.866**	**0.877**	0.434	**0.893**

此外，上文提到，一阶 HMM 可以视为是 Bigram 模型，其仅考虑了前一个状态对当前状态的影响，如果可能考虑更长远的状态信息，是不是会效果更好呢？在 TnT[6] 这篇文章中提出了一种将 Trigram 概率进行分词的二阶 HMM 模型，其状态转移概率为 Unigram、Bigram 和 Trigram 的线性组合：

$$p(t_3|t_2,t_1) = \lambda_1\hat{P}(t_3) + \lambda_2\hat{P}(t_3|t_2) + \lambda_3\hat{P}(t_3|t_2,t_1)$$

这里参考了 snownlp[4]中分词工具对 TnT 模型的实现，对实验结果进行评估，见表3。

通过分析不难发现，TnT 考虑了 Trigram 概率，确实能够提高分词的精确率与未登录词的召回率。但是相应的，使用 Trigam 信息，使得算法实现更加复杂，例如需要统计前两个词的信息，并且在解码过程中，需要使用二维 dp 数组记录，运行效率上也有下降。同时这里对比了加 chunk 之后的效果，可以看到，加入分段之后，也能带来一定的提升，可见人工规则在机器学习算法中还是能够起到一定的作用。另外，在使用 snownlp 默认配置的时候，我们发现能够达到

[4]snownlp: https://github.com/isnowfy/snownlp

表 3: TnT 模型与 HMM 模型分词效果的对比

TNT			P	R	F_1	R_{OOV}	R_{IV}
序号	模型	chunk					
1	HMM	否	0.715	0.745	0.730	0.315	0.771
2	HMM	是	0.760	0.784	0.772	0.436	0.805
3	TNT	否	0.767	0.767	0.767	0.341	0.793
4	TNT	是	**0.776**	**0.785**	**0.780**	0.417	**0.807**

$F_1 = 0.895$ 和 $R_{OOV} = 0.325$ 的效果，目前分析是与 snownlp 默认设置，可能引入了外部词典信息。

4.2.2 最大熵模型

最大熵原理是指：在选择概率模型时，对所有满足分布要求的概率模型，能够使得熵最大的模型，是最好的模型。最大熵模型是基于最大熵原理在分类问题上的应用。最大熵原理是统计学习中的一个一般性原理，许多机器学习模型（例如最大熵模型，CRF 模型，结构化感知机等）均体现了最大熵原理。下面给出最大熵模型的定义。

对于特征函数 $f(x,y)$，其描述了输入 x 与输出 y 之间的某一个事实。定义为：

$$f(x,y) = \begin{cases} 1, & x与y满足某一事实 \\ 0, & 否则 \end{cases}$$

特征函数 $f(x,y)$ 关于经验分布 $\tilde{P}(X,Y)$ 的期望 $E_{\tilde{P}}(f)$ 表示为：

$$E_{\tilde{P}}(f) = \sum_{x,y} \tilde{P}(x,y) f(x,y)$$

特征函数 $f(x,y)$ 关于模型 $P(Y|X)$ 与经验分布 $\tilde{P}(X)$ 的期望值，用 $E_{\tilde{P}}(f)$ 表示：

$$E_P(f) = \sum_{x,y} \tilde{P}(x) P(y|x) f(x,y)$$

假设上述两个期望值想的，即 $E_{\tilde{P}}(f) = E_P(f)$，就可以得到最大熵模型：

$$\mathcal{C} \equiv \{P \in \mathcal{P} | E_P(f_i) = E_{\tilde{P}}(f_i), \quad i = 1, 2, \cdots, n\}$$

是所有满足约束条件的模型集合，定义在条件概率分布 $P(Y|X)$ 上的条件熵是

$$H(P) = -\sum_{x,y} = \tilde{P}(x) P(y|x) \log P(y|x)$$

，集合 $\mathcal{C}$ 中使得条件熵 $H(P)$ 最大的模型就是最大熵模型。

最大熵的学习算法可以采用迭代法或牛顿法、拟牛顿法、梯度下降法等进行优化，其等价的约束优化问题表述为

$$\begin{aligned} \max_{P\in\mathcal{C}} \quad & H(P) = -\sum_{x,y} \tilde{P}(x)P(y|x)\log P(y|x) \\ s.t. \quad & E_P(f_i) = E_{\tilde{P}}(f_i), i = 1, 2, \cdots, n \\ & \sum_y P(y|x) = 1 \end{aligned}$$

在本实验中，我们基于张乐博士的 maxent[5]工具包实现了中文分词。由于采用的是默认的配置，也即仅使用当前字符特征，得到的测试结果见表4。

表 4: 最大熵模型评测效果

序号	**模型**	**特征数量**	P	R	F_1	R_{OOV}	R_{IV}
1	MaxEnt	1	0.840	0.830	0.835	0.484	0.851
2	MaxEnt	13	0.930	0.938	0.934	0.741	0.985
3	MaxEnt	16	**0.933**	**0.941**	**0.937**	**0.741**	**0.945**

表4中序号 1 为默认配置下运行测试的效果，序号 2 和 3 为添加特征后的结果，这里可以看出，人工特征的选择对于最大熵模型的性能影响是非常大的。

图6展示了不同迭代次数下，最大熵模型的测评表现，其中在迭代 200 次左右效果为最优，之后开始下降。

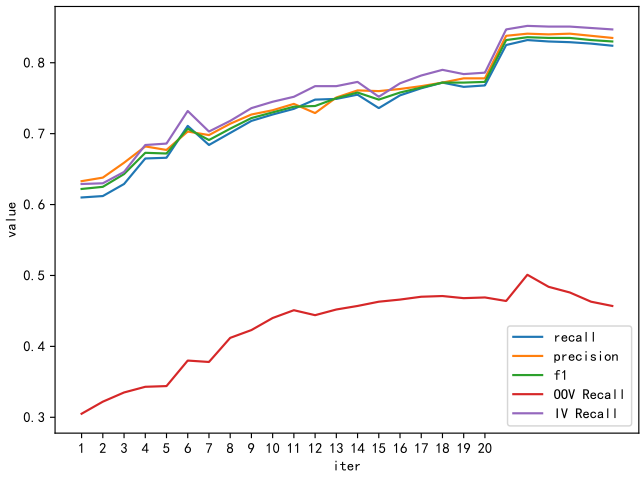


图 6: 不同迭代次数下 MaxEnt 模型测评的效果

[5]maxent：https://github.com/lzhang10/maxent

4.2.3 条件随机场模型

条件随机场 (CRF[1]) 模型，是一种非常经典的无向图模型，经常用于处理序列标注问题。与 HMM 不同，CRF 是一种判别模型，其直接建模条件概率 $P(Y|X)$，而非建模联合概率 $P(X,Y)$。CRF 相对与 HMM 而言，摒弃了其两个不合理的假设，也就是齐次马尔科夫假设和条件独立假设。因此在分词、词性标注、命名实体识别等领域得到了广泛的应用。图7给出了 CRF 模型在这个问题上的示例。

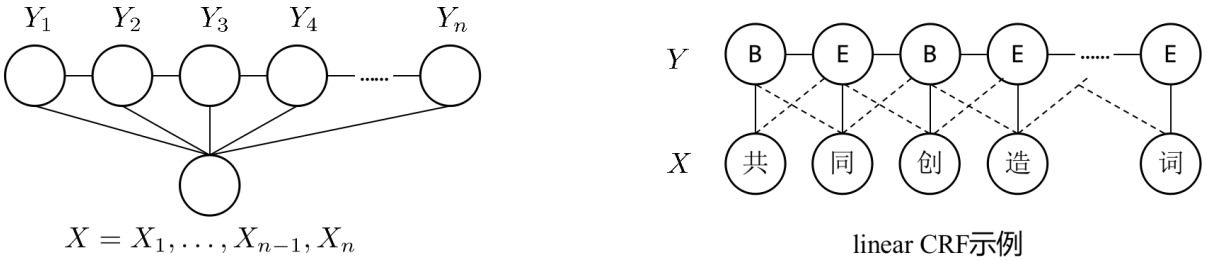


图 7: CRF 模型图示

用于分词模型的 CRF 为线性链 CRF（linear-CRF），其马尔科夫性表现为

$$P(Y_i|X, Y_1, Y_2, \ldots, Y_n) = P(Y_i|X, Y_{i-1}, Y_{i+1})$$

CRF 的参数化表示为：

$$P(y|x) = \frac{1}{Z(x)} \exp(\sum_{i,k} \lambda_k t_k(y_{i-1}, y_i, x, i) + \sum_{i,l} \mu_l s_l(y_i, x, i))$$

$$Z(x) = \sum_{y} \exp(\sum_{i,k} \lambda_k t_k(y_{i-1}, y_i, x, i) + \sum_{i,l} \mu_l s_l(y_i, x, i))$$

CRF 的可采用最大似然估计或正则化的最大似然估计，对数似然函数表示为

$$L(w) = \log \prod_{x,y} P_w(y|x)^{\bar{P}(x,y)} = \sum_{x,y} \bar{P}(x,y) \log P_w(y|x)$$

完成参数学习后，与 HMM 模型类似可采用 Viterbi 算法进行解码，得到最优序列，第递推公式如下：

$$\delta_{i+1}(l) = \max_{1 \le j \le m} \{\delta_t(j) + \sum_{k=1}^{K} w_k f_k(y_i = j, y_{i+1} = l, x, i)\}, l = 1, 2, \ldots, m$$

$$\Psi_{i+1}(l) = \arg \max_{1 \le j \le m} \{\delta_i(l)\} + \sum_{k=1}^{K} w_k f_k(y_i = j, y_{i+1} = l, x, i), l = 1, 2, \ldots, m$$

本实验基于 CRF++[6]实现了分词。CRF++ 采用了 template 文件的方式来指定特征，程序会根据定义在 template 文件中的特征，自动生成大量特征函数（数量与句子数量和定义的特征模板数量成正比）。本实验中尝试了多种特征模板，包括 3 特征、6 特征与 12 特征，基本思路是先考虑 Unigram，然后加上 Bigram，再加上 Trigram 信息。12 特征的模板文件见下方4.2.3：

[6]CRF++：https://taku910.github.io/crfpp/

template

```
# Unigram
U00:%x[-2,0]
U01:%x[-1,0]
U02:%x[0,0]
U03:%x[1,0]
U04:%x[2,0]
U05:%x[-2,0]/%x[-1,0]
U06:%x[-1,0]/%x[0,0]
U07:%x[0,0]/%x[1,0]
U08:%x[1,0]/%x[2,0]
U09:%x[-2,0]/%x[-1,0]/%x[0,0]
U10:%x[-1,0]/%x[0,0]/%x[1,0]
U11:%x[0,0]/%x[1,0]/%x[2,0]
# Bigram
B
```

具体的评测效果见表5

表 5: CRF 模型评测效果

序号	**模型**	**特征数量**	P	R	F_1	R_{OOV}	R_{IV}
1	CRF	3	0.911	0.927	0.919	0.487	0.937
2	CRF	6	0.937	0.920	0.929	0.559	0.942
3	CRF	12	**0.941**	**0.924**	**0.932**	**0.612**	**0.943**

这里 CRF++ 得到的最佳效果只有 $F_1 = 0.932$，并不是很优，在汇报展示时老师提示 CRF 模型一般能达到 $F_1 = 0.95$ 左右的效果，目前看来还是有不小的差距。分析可能是因为对模板文件的理解不够，没能设计出足够好的特征。因此还需要继续改进，并且对数据集合以及 bad case 进行分析。

4.2.4 其他分词方法与小结

除了上述提到较为经典的分词模型与算法，我们还测试了 pkuseg[7]和 thulac[8]这两个分词工具。其中 pkuseg 采用的是基于 ADF[7] 算法改进的 CRF 分词模型[8]；而 THULAC 则是基于结构化感知机[2] 的分词算法。这两个工具包均使用了很多工程上的优化技巧，例如 pkuseg 中使用

[7]PKUSEG：https://github.com/lancopku/pkuseg-python

[8]thulac: http://thulac.thunlp.org

了 Trie 来存储词典，并且使用了 Python 多线程。而 thulac 则使用了 DAT 数据结构，并且基于 C++ 编写，训练起来非常快。

图8给出了上述分词算法的对比：

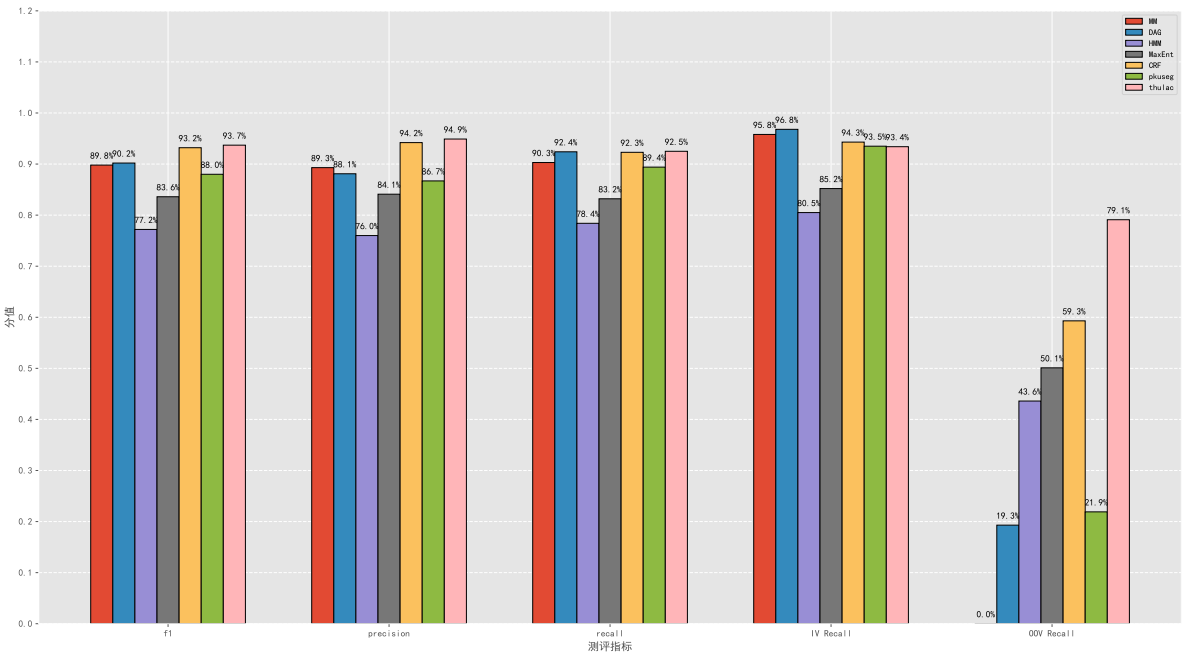


图 8: 各传统分词方法对比

第三小节　基于深度学习的分词算法

4.3.1　基于深度学习的分词框架

经过调研，本实验选择如图9所示的分词系统框架[9] 作为基础框架。在该分词系统框架中，首先输入数据进入字嵌入层得到每个字符相应的字向量，之后字向量经过编码层进行特征的抽取后得到相应的特征向量，最后特征向量进入解码层进行解码后得到最终的序列标注。在本框架中，输入数据的处理方式、字嵌入层所用模型、编码层所用模型及解码层所用模型均具有较多的的选择。本节之后的部分将会介绍本实验为不同子模块所用算法或模型的尝试。

4.3.2　数据预处理

本实验中数据处理的方式有两种：直接拆分和针对特殊字符预处理后拆分。

直接拆分法的思想是对所有的字符平等对待，将输入文本拆分为单独字符组成的序列，示例如下：

```
总 理 走 上 台  -> 总  理  走  上  台
```

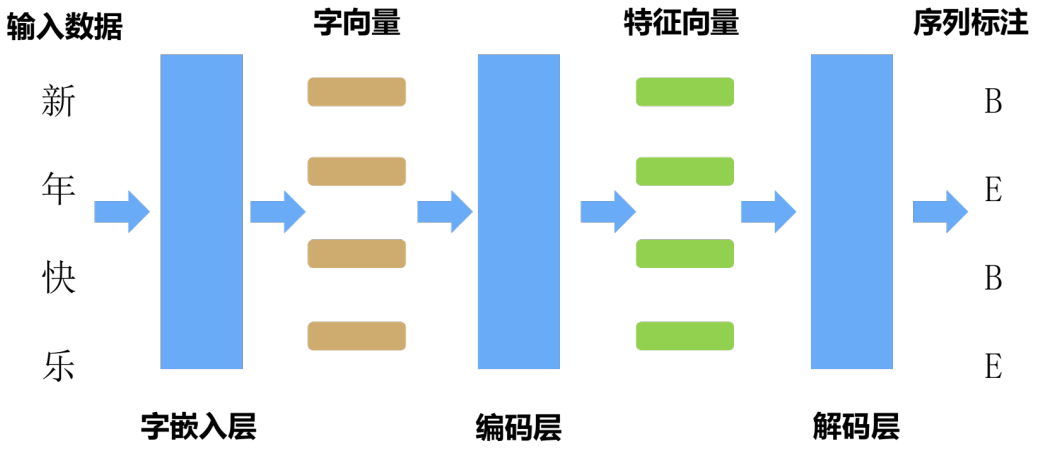


图 9: 基于深度学习的分词系统的一种基本框架

针对特殊字符预处理后拆分的方法则是将一些特殊字符转化为统一的符号，再将输入文本拆分为字符组成的序列。本实验总结的特殊字符有以下几种：

1) 标点符号。单独的符号（如”。” ,” (” 等）直接使用特殊符号“[PUNC]”统一代替。连续的符号（如“……” ,” ——”）合并后使用特殊符号“[PUNC]”统一代替。示例为：

```
总 理 走 上 台 来，亲 切 慰 问 我 们 …… -> 总 理 走 上 台 来 [PUNC] 亲 切 慰
    问 我 们 [PUNC]
```

2) 数字。分为全角数字、半角数字和中文数字。分类合并后使用特殊符号“[NUM]”统一代替。示例为：

```
二    年 十 二 月 三 十 一 日 -> [NUM] 年 [NUM] 月 [NUM] 日
```

3) 字母。合并后使用特殊符号“[ALP]”统一代替。

4.3.3 字嵌入层设计

本实验中分别对比了两种字嵌入层方法：随机初始化和 GloVe[3] 模型。GloVe 全称为全局向量词表示，是一种既利用了语料库的全局统计特征，也利用了滑动窗口的局部上下文特征的字向量生成模型。GloVe 的输入为语料库中字的共现频率矩阵，算法优化的目标损失函数为：

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2$$

其中 V 为词表大小，权重函数 $f(x)$ 表示为：

$$f(x) = \begin{cases} (x/x_{\max})^\alpha, & \text{if } x \le x_{\max} \\ 1, & \text{otherwise.} \end{cases}$$

在本实验中，训练 GloVe 模型的主要超参数设置如下：

- Vocab min count=3；
- vector size=300，表示输出的字嵌入向量维度为 300；
- iteration number=1000，GloVe 模型迭代次数；
- window size=15，滑窗大小为 15

对于随机初始化字向量模式，直接采用 PyTorch 中 random 相关 API，这里主要是用来作为对比参照实验，无需赘述。

4.3.4 编码层设计

编码层的主要作用是提取特征，从字向量信息得到特征向量。本实验中编码层使用的模型有 LSTM（Long short-term memory）[4] 和 Transformer[10]。LSTM 是循环神经网络（RNN）的一种变体。与传统的 RNN 相比，LSTM 通过精妙的门控开关，不仅在抽取特征时记住较长的历史信息，还很好地解决了“梯度消失”和“梯度爆炸”的问题。

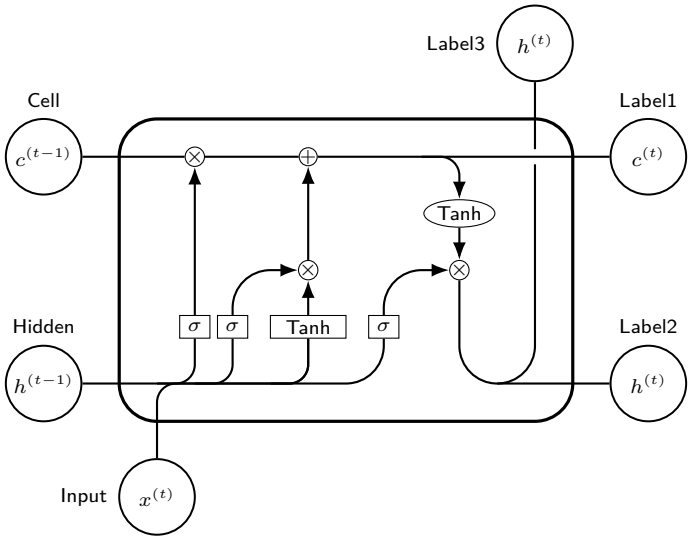


图 10: 一个 LSTM 单元的结构

图10展示了一个 LSTM 单元的结构，其内部主要包含三个门控机制，用于控制历史信息、当前信息的保留和输出，门控机制的计算公式如下：

$$f^{(t)} = \sigma(U^f x^{(t)} + W^f h^{(t-1)} + b^f)$$

$$i^{(t)} = \sigma(U^f x^{(t)} + W^i h^{(t-1)} + b^i)$$

$$o^{(t)} = \sigma(U^o x^{(t)} + W^i h^{(t-1)} + b^o)$$

其中，$f^{(t)}$，$i^{(t)}$，$o^{(t)}$ 分别表示遗忘门（Forget Gate），输入门（Input Gate）和输出门（Output Gate）。LSTM 单元的内部状态的更新与 Forget Gate 和 Input Gate 相关：

$$c^{(t)} = f^{(t)} \circ c^{(t-1)} + i^{(t)} \circ \tilde{c}^{(t)}$$

$$\tilde{c}^{(t)} = \tanh(Ux^{(t)} + Wh^{(t-1)} + b)$$

输出和 Output Gate 相关：

$$h^{(t)} = o^{(t)} \circ \tanh c^{(t)}$$

另外，由于在本实验中，需要分词的句子是给定了的，因此我们使用的是双向 LSTM（BiLSTM）。具体的参数配置如下：

- Layers=4，表示将一个 LSTM Cell 堆叠 4 层；
- Hidden size=300，隐向量 $h^{(t)}$ 的维度为 300；
- Bidirectional=True，设置为双向 LSTM；
- Dropout=0.1，防止过拟合；

Transformer[10] 是一种基于自注意力（self-attention）机制的网络结构。与 LSTM 相比，它不仅能够更好地捕捉文本中的远距离依赖，还能实现并行式训练，堆叠出更深的网络。

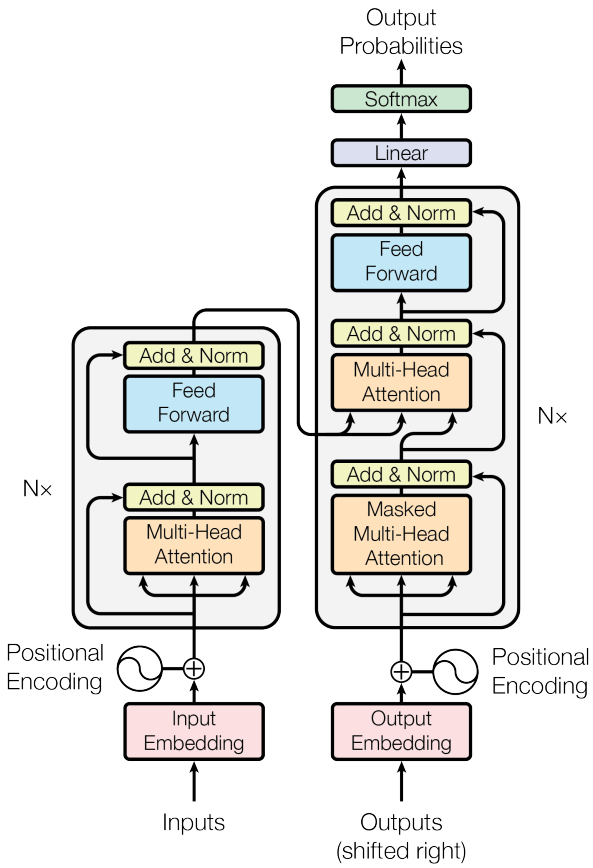


图 11: Transformer 模型结构

图11展示了 Transformer 的模型结构，该模型是由 Encoder 和 Decoder 两部分组成的，即为图11的左侧和右侧，在本试验的分词模型中，我们仅用到了左侧 Encoder 部分。Transformer Encoder（Decoder 类似）内部结构中主要包含三个主要部分，计算公式如下

$$\text{Attention}(Q,K,V)=\text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Multi-Head Attention:

$$\text{MultiHead}(Q,K,V)=\text{Concat}(\text{head}_1,...,\text{head}_\text{h})W^O$$
$$\text{where head}_\text{i}=\text{Attention}(QW_i^Q,KW_i^K,VW_i^V)$$

Add Norm:

$$\tilde{x}=\text{LayerNorm}(x+\text{Sublayer}(x))$$

Feed Forward:

$$\text{FFN}(x)=\max(0,xW_1+b_1)W_2+b_2$$

在本实验中，Transformer 层的参数配置如下：

- Layers=4，表示 Transformer 层的层数；
- Hidden size=300，表示 Transformer 层输出的隐层向量的维数，为 300；
- Heads numbers=2，表示 Multi-Head Attention 中 h 的大小；
- Dropout=0.1，表示神经元丢弃率，用于正则化；

4.3.5 解码层设计

解码层的作用是将经过深度神经网络的特征向量解码出预测序列。本实验中解码层使用的模型有 Linear Layer + Softmax 和 CRF。

Linear Layer + Softmax：将编码层得到的特征输入全连接层，经 softmax 后每个字对应输出 5 维向量，表示属于某类（BMESO，这里引入了 O 表示输入序列中 padding 的部分）的概率，需要最小化的损失函数为交叉熵。

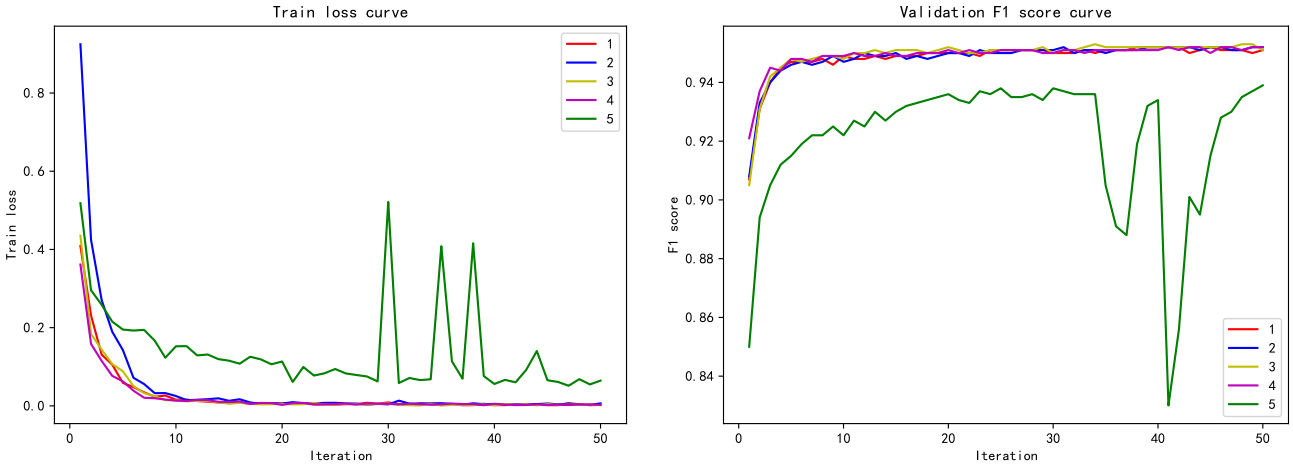
CRF：将编码层得到的特征输入 CRF 层，得到最大得分和相应序列，需要最小化的损失函数为模型当前最大得分与正确标注序列得分之差。

4.3.6 结果分析与对比

为对比各种不同模型的效果，本实验使用五折交叉验证对模型的性能进行初步的对比，结果如表6所示。每个模型的最大训练迭代次数为 50，直到模型收敛为止，实验中优化器采用 Adam，初始学习率为 $1e-4$。从表6中可以看出，除模型 5（编码器采用 Transformer）外，其余模型在验证集上的平均 F_1 值相差不大。

表 6: 深度学习模型在验证集上的评测效果

模型					P	R	F_1
序号	预处理	字嵌入层	编码层	解码层			
1	否	Glove	LSTM	Linear	0.952	0.952	0.952
2	否	Glove	LSTM	CRF	0.952	0.952	0.952
3	是	Glove	LSTM	Linear	**0.953**	**0.952**	**0.953**
4	是	Random	LSTM	Linear	0.951	0.952	0.952
5	是	Glove	Transformer	Linear	0.940	0.939	0.939



1) Loss 曲线 2) 验证集 F_1 曲线

图 12: 模型训练时 loss 变化与验证集评测效果

各个模型训练的 loss 曲线和验证集 F_1 曲线如图12所示。可以看出编码层为 LSTM 的模型训练时 loss 下降得十分快速和平稳，而编码层为 Transformer 的模型在训练时在经过一定迭代次数后 loss 的抖动较大，且无法降低到接近 0。

为进一步对比各种深度学习模型的效果和泛化能力，依据训练时的 loss 曲线，我们以与交叉验证同样的超参设置，使用全部训练集对编码器为 LSTM 的模型迭代训练 20 次，对编码器为 Transformer 的模型迭代训练 50 次。最终使用测试集对训练好的模型进行结果对比，如表7所示。

表 7: 深度学习模型在验证集上的评测效果

模型					P	R	F_1	R_{OOV}	R_{IV}
序号	预处理	字嵌入层	编码层	解码层					
1	否	Glove	LSTM	Linear	0.927	0.921	0.924	0.597	0.940
2	否	Glove	LSTM	CRF	0.928	0.923	0.926	0.629	0.941
3	是	Glove	LSTM	Linear	**0.942**	**0.937**	**0.940**	**0.775**	**0.947**
4	是	Random	LSTM	Linear	0.940	0.937	0.938	0.765	**0.947**
5	是	Glove	Transformer	Linear	0.911	0.893	0.902	0.696	0.905

对比分析如下：

1) 模型 1 和模型 2 对比：解码层使用 CRF 的模型与解码层使用 Linear Layer+Softmax 的模型相比，F1 值有小幅度提升（0.2%），但 ROOV 提升较明显（3.2%）。猜测是由于 CRF 层添加了标签间的转移约束，使得当模型遇到未登录词时也可以根据标签转移信息和前一个字的标签来确定当前的标签，使效果提高。

2) 模型 1 和模型 3 对比：对数据进行预处理后，F1 值有较明显提升（1.6%），ROOV 也有很明显的提升（17.8%）。这是由于预处理过程添加人工规则将多样的特殊字符统一化表示，使模型可以更专注与普通字符的分类，更注重特殊字符的共性，从而更好地抽取特征并进行分类。

3) 模型 3 和模型 4 对比：使用 Glove 作为字嵌入层比随机初始化的方法要更好，F1 有小幅度的提升（0.2%）。由于 Glove 为模型引入了另一个任务，增强了模型的泛化能力，故在测试集上的效果更好；但由于训练数据集较少，使得效果提升有限。

4) 模型 3 和模型 5 对比：与 LSTM 相比，本实验中 Transformer 作为编码器的模型所取得的效果不是很好。推测可能的原因有：超参数设置不好（如 Transformer 层数不够，优化器选择不合适）；分词任务可能不太需要 Transformer 模型所提供的全局信息等。可能的改进有：切割样本，缩短长度；精调学习率、隐层大小、Transformer 层数等参数；选择其他优化方法；增大训练数据量等。

4.3.7 测试样例分析

本实验对各个深度学习模型的分词结果进行分析，并对上节的结果分析加以印证。现抽取其中两个文本的分词结果进行分析，前面数字表示模型编号，后面分隔开的句子表示模型分词结果。

1) “共同创造美好的新世纪——二〇〇一年新年贺词”

模型序号	预测结果
1	共同 创造 美好 的 新 世纪 — — 二 〇 〇 一 年 新年 贺词
2	共同 创造 美好 的 新 世纪 — — 二 〇 〇 一 年 新年 贺词
3	共同 创造 美好 的 新 世纪 —— 二〇〇一年 新年 贺词
4	共同 创造 美好 的 新 世纪 —— 二〇〇一年 新年 贺词
5	共同 创造 美好 的 新 世纪 —— 二〇〇一年 新年 贺词

分析：数据预处理对特殊字符分词具有重要影响。

模型序号	预测结果
1	欢歌笑语 响彻 了 高原 宁静 的 夜空 。
2	欢歌笑语 响彻 了 高原 宁静 的 夜空 。
3	欢歌笑语 响彻 了 高原 宁静 的 夜空 。
4	欢歌笑语 响彻 了 高原 宁静 的 夜空 。
5	欢歌笑 语响 彻 了 高原 宁静 的 夜空 。

分析：对于长文本（即使经过标点符号分句后，测试集中仍含有字符数量超过 800 的句子），由于 Transformer 对整段文本进行 self-attention，特征中含有全局信息过多，导致可能更需要局部信息的分词任务效果不好。

第五节 总结与感想

罗旭坤：在这次的分词项目中，我们小组试用了很多种方法，从词典匹配到传统机器学习再到深度学习，也对比了不同方法的效果。这既让我对上课讲的不同模型有了更加深刻的理解，也打破了我之前的一些错误认知。以前我感觉深度学习就是最好的，遇到一些问题就想用深度学习模型，但事实证明传统的机器学习模型也是包含前人的智慧的，甚至连词典匹配的方法也能有不错的效果。深度学习方法固然由于自动抽取特征的特点显得比较方便，但是他目前还不能和人类相比，如果我们精心去发现和挑选特征，传统机器学习算法在分词任务中甚至能战胜一些深度学习模型。我认为将传统机器学习和深度学习进行多方面的结合，可能会产生不一样的火花，以后一定可以得到兼具正确性和可解释性的完美模型。这正是我们需要努力去探索研究的。

马心睿：我在此次实验中，首先通过查阅资料，看论文，了解到中文分词的传统方法和深度学习的方法，各种方法对应的最优效果。其次在实验中，实现了最大匹配方法，HMM 方法，lstm

模型，bi-lstm+crf（嵌入层没有用 glove）模型。其中效果最差的 HMM 的 F1 值为 0.635，效果最好的 bi-lstm+crf 的 F1 值为 0.880（受限于 cpu，只运行一个 epoch）。最后在与同组同学的讨论交流中，体会到实验的重要性。通过实验加深对理论部分的理解，通过实验的结果验证想法，不断尝试。感谢同组同学给予的帮助，从他们身上学习到认真负责的工作态度和超强的执行力，收获匪浅。

罗登：在本次中文分词的项目中，我们对比了各种经典的分词方法，包括词表匹配法，基于机器学习的分词算法，以及基于深度模型的算法。通过实践尝试，对分词这个问题，有了较为全面的认识。在完成这个分词项目的时候，我们遇到了很多的困难。特别是在项目初期，我们不清楚如何下手。我做了词频统计和一些可视化工作，并且用 jieba 完成了第一个 baseline，之后也陆续地实现了最大匹配，HMM，词图切分等算法。通过不断调试代码，阅读开源项目的源码，查阅相关博客、论文等方式，我们有了更多的想法，也做了更多的对比试验。通过实验，我们加深了对课程知识的理解，也验证了一些结论，还发现了许多实践上颇为有效的技巧，更加深刻地体会到了“实践出真知”的含义。分词这个问题尽管看上去很简单，但是通过不断的深入研究，越能发现其中的有趣之处。在课程展示和答辩的过程中，老师和学长们也提出了很多的意见和建议，包括对 crf++ 用法的问题，还有 Transformers 训练上的问题，在之后我们也立刻补充实验，尽快地解决了相应的问题。在聆听别的同学的汇报后，我们知道本次课程作业还有很多可以改进的空间，因此我们也会在此基础上不断探索，得到更优的结果。本次课程项目也是以开源的方式在 github[9]上维护，强调结果能够复现，以供更多的同学学习和参考。

第六节　实验分工

表 8: 组员工作与分工情况

组员	分工
罗旭坤	实现了基于深度学习的分词模型，包括构建、验证与测试；展示 PPT 的编辑，修改；实验报告的撰写
罗登	实现了词表匹配算法（前后最大匹配，DAG 匹配），验证了主流机器学习(HMM, 结合人工规则, 最大熵模型, CRF 模型, TnT, 添加 pkuseg 和 thulac 的结果)；维护 github 仓库；完成展示 PPT；完成实验报告
马心睿	初期，调研并整理了大量十分有用的学习资源，链接等，为后期计划开展与实施提供了帮助；实现了最大匹配算法（前向、后向与双向），HMM 模型，完成深度学习部分 BiLSTM+CRF 的训练（去除 GloVe 字向量的对比试验）

[9]HanTokenization：https://www.github.com/RonDen/HanTokenization

参考文献

[1] LAFFERTY J, MCCALLUM A, PEREIRA F C. Conditional random fields: Probabilistic models for segmenting and labeling sequence data[J]., 2001.

[2] COLLINS M. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms[C]//Proceedings of the 2002 conference on empirical methods in natural language processing (EMNLP 2002). [S.l. : s.n.], 2002: 1-8.

[3] PENNINGTON J, SOCHER R, MANNING C D. Glove: Global vectors for word representation[C]//Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). [S.l. : s.n.], 2014: 1532-1543.

[4] HOCHREITER S, SCHMIDHUBER J. Long short-term memory[J]. Neural computation, 1997, 9(8): 1735-1780.

[5] AOE J I, MORIMOTO K, SATO T. An efficient implementation of trie structures[J]. Software: Practice and Experience, 1992, 22(9): 695-721.

[6] BRANTS T. TnT-a statistical part-of-speech tagger[J]. ArXiv preprint cs/0003055, 2000.

[7] SUN X, WANG H, LI W. Fast online training with frequency-adaptive learning rates for chinese word segmentation and new word detection[C]//Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). [S.l. : s.n.], 2012: 253-262.

[8] LUO R, XU J, ZHANG Y, et al. Pkuseg: A toolkit for multi-domain chinese word segmentation[J]. ArXiv preprint arXiv:1906.11455, 2019.

[9] LAMPLE G, BALLESTEROS M, SUBRAMANIAN S, et al. Neural architectures for named entity recognition[J]. ArXiv preprint arXiv:1603.01360, 2016.

[10] VASWANI A, SHAZEER N, PARMAR N, et al. Attention is all you need[J]. ArXiv preprint arXiv:1706.03762, 2017.

[11] LI Z, SUN M. Punctuation as implicit annotations for Chinese word segmentation[J]. Computational Linguistics, 2009, 35(4): 505-512.