《机器学习》课程系列

线性链条件随机场*

武汉纺织大学数学与计算机学院

杜小勤

2020/05/15

Contents

*本系列文档属于讲义性质，仅用于学习目的。Last updated on: June 22, 2020。

1 概率图模型

概率图模型 (Probabilistic Graphical Model, PGM) 是一类使用图结构来简洁紧凑地表达随机变量之间关系的概率模型，它充分利用了随机变量之间的依赖关系 (因果关系) 和关联关系 (相关关系)，大大降低了联合概率求解的复杂度。

在概率图中，节点表示一个或一组随机变量，节点之间的边表示随机变量 (组) 之间的概率关系，它们构成所谓的“随机变量关系图”[1]。根据边是否具有方向性，可以把概率图模型分为 2 类：

- 概率有向图模型或贝叶斯网络 (Bayesian Network)。它使用有向边表示随机变量之间的因果关系，形成了有向无环图；
- 概率无向图模型或马尔科夫网络 (Markov Network)。它使用无向边表示随机变量之间的相关关系，形成了无向图；

一些经典的浅层模型均属于概率图模型，例如，概率有向图的典型代表有：朴素贝叶斯、HMM 模型、Kalman 滤波模型等；概率无向图的典型代表有：条件随机场等。

1.1 概率有向图——HMM 模型

HMM 模型是一种简单的概率有向图模型或动态贝叶斯网络 (Dynamic Bayesian Network, DBN)，其模型示意图如图1-1所示。

在 HMM 模型中，随机变量分为 2 类：(隐) 状态随机变量和观测随机变量。在图1-1中，前者使用灰色实心方块表示，并依时序关系形成状态序列

[1]实际上，可以把概率图模型简单地理解为具有概率关系的结构图。

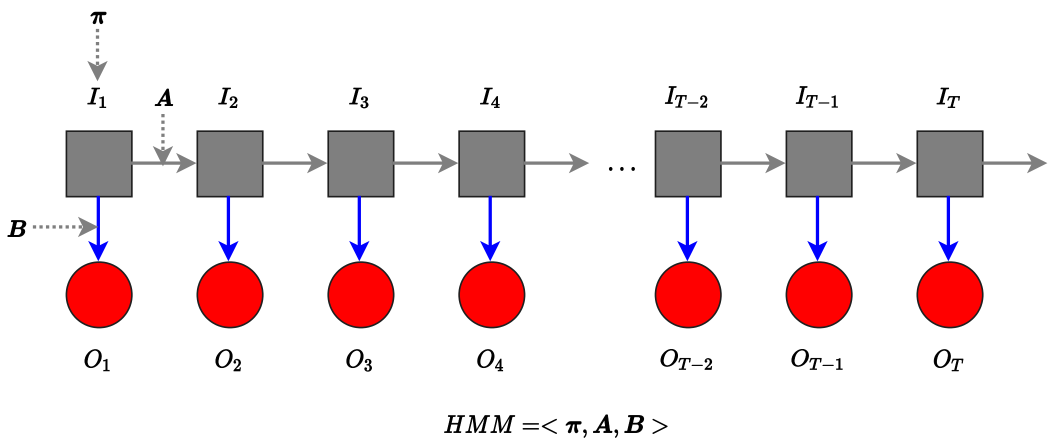

图 1-1: 一种简单的概率有向图模型-HMM

$\boldsymbol{I} = (I_1, I_2, \ldots, I_T)$(其中 I_t 为第 t 个状态，T 表示时间步数)，状态与状态之间的边具有方向性；后者使用红色实心圆表示，每个观测随机变量都有 1 个与之对应的状态随机变量，它们之间的边也是有方向的：由状态变量指向观测变量。这些观测也依时序关系形成观测序列 $\boldsymbol{O} = (O_1, O_2, \ldots, O_T)$(其中 O_t 为第 t 个观测)。

在 HMM 中，存在 2 种 Markov 性质：状态转移的 Markov 链 (齐次 Markov) 与观测的独立性。前者表达为状态转移概率矩阵 $\boldsymbol{A}$，其中：

$$\begin{aligned} &\boldsymbol{A} = [a_{q_i}(q_j)]_{N\times N} \\ &a_{q_i}(q_j) = P(I_{t+1} = q_j | I_t = q_i) \quad (= P(I_{t+1} = q_j | I_t = q_i, O_t, \ldots, I_1, O_1)) \end{aligned} \tag{1}$$

其中 N 表示状态个数，$i, j = 1, 2, \ldots, N$，$t = 1, 2, \ldots, T-1$。后者表达为观测矩阵 $\boldsymbol{B}$，其中：

$$\begin{aligned} &\boldsymbol{B} = \left[b_{q_j}(v_k)\right]_{N\times M} \\ &b_{q_j}(v_k) = P(O_t = v_k | I_t = q_j) \quad (= P(O_t = v_k | I_t = q_j, O_{t-1} \ldots, I_1, O_1)) \end{aligned} \tag{2}$$

其中 M 表示观测个数，$j = 1, 2, \ldots, N$，$k = 1, 2, \ldots, M$，$t = 1, 2, \ldots, T$。

上述的状态转移矩阵 $\boldsymbol{A}$、观测矩阵 $\boldsymbol{B}$ 与初始状态概率向量 $\boldsymbol{\pi}$ 一起构成了 HMM 模型 $\boldsymbol{\lambda} =< \boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{B} >$[2]。

HMM 模型实际上是一种简化的概率有向图模型，它当然也能够表达联合分布，因而属于生成式模型。利用随机变量之间的条件独立性，所有随机变量的联

[2]关于 HMM 模型的概率计算、预测与学习算法，详见《机器学习》课程系列之 HMM 模型，Chapter11-CN.pdf。

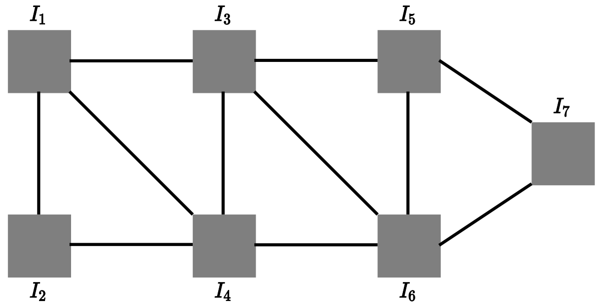

图 1-2: 一个简单的 Markov 随机场

合概率分布可以表示为[3]：

$$\begin{aligned} P(I_1, O_1, I_2, O_2, \ldots, I_T, O_T) &= P(I_1)P(O_1|I_1)\prod_{t=2}^{T} P(I_t|I_{t-1})P(O_t|I_t) \\ &= \pi(I_1)b_{I_1}(O_1)\prod_{t=2}^{T} a_{I_{t-1}}(I_t)b_{I_t}(O_t) \end{aligned} \tag{3}$$

关于 HMM 的概率计算、预测与学习算法，请阅读相关资料，在此不再赘述。

1.2 概率无向图——Markov 随机场

Markov 随机场 (Markov Random Field, MRF) 是一种典型的概率无向图或 Markov 网络。与概率有向图模型一样，图中的每个节点表示随机变量。

但是，在概率有向图模型中，节点与节点之间具有明确的因果关系，可以直接使用先验概率或条件概率直接建模这种关系，因而也就可以直接使用这些概率计算联合概率。

而在概率无向图模型中，节点之间的边表示两个随机变量之间存在某种相关关系，而非明确的因果关系——这意味着节点之间失去了用于表达因果关系的条件分布，而只能通过局部参数来建模它们之间的相关关系——需要一组局部势函数 (Potential Function) 或因子 (Factor) 来表示联合概率分布。

为了简洁地表示概率无向图中的联合概率，需要引入团 (Clique) 和极大团 (Maximal Clique) 的概念。图1-2展示了一个简单的 Markov 随机场。

定义 1.1（团与极大团）对于图中的任一子集，如果该子集中的任意 *2* 个节点之

[3]注意，一般的概率有向图模型，也充分利用了随机变量之间的条件独立性，能够对联合概率分布的表示进行简化。

间都有边连接，则称该子集形成一个团。如果在一个团中，无法再加入新节点，使之成为一个更大的团，那么称该团为极大团。

在图1-2中，任意 2 个相邻节点形成一个最小团，例如 $\{I_1, I_2\}$ 和 $\{I_3, I_4\}$ 等。向团 $\{I_1, I_2\}$ 中添加 1 个节点 I_4，使之成为一个更大的团 $\{I_1, I_2, I_4\}$。但是，继续添加 I_3，无法使 $\{I_1, I_2, I_4, I_3\}$ 成为更大的团——I_2 与 I_3 之间没有直接边连接。因此，$\{I_1, I_2, I_4\}$ 是一个极大团。显然，每个节点至少出现在一个极大团中，且极大团不是任何其它团的子集。

为了更加简洁高效地定义联合概率，需要将局部势函数定义在极大团之上，即将极大团作为局部势函数的定义域[4]。

于是，在概率无向图中，定义所有随机变量的联合概率分布为所有极大团的局部势函数 (因子) 的乘积 (需要配上归一化项)[5]：

$$P(\boldsymbol{I}) = P(I_1, I_2, \ldots, I_N) = \frac{\prod\limits_{Q \in \boldsymbol{\mathcal{C}}} \psi_Q(\boldsymbol{I}_Q)}{\sum\limits_{\boldsymbol{I}} \prod\limits_{Q \in \boldsymbol{\mathcal{C}}} \psi_Q(\boldsymbol{I}_Q)} = \frac{\prod\limits_{Q \in \boldsymbol{\mathcal{C}}} \psi_Q(\boldsymbol{I}_Q)}{Z} \tag{4}$$

其中，$\boldsymbol{\mathcal{C}}$ 表示所有极大团 Q 构成的集合，$\boldsymbol{I}_Q$ 表示极大团 Q 内的随机变量集合，ψ_Q 表示极大团 Q 所对应的局部势函数。上式中，分母项 Z 为归一化项，确保 $P(\boldsymbol{I})$ 为正确定义的概率。在实际应用中，精确计算归一化项 Z 通常较困难。但是，许多任务往往并不需要获得归一化项 Z 的精确值。

对于势函数 ψ_Q，需要满足以下约束条件：

- $\psi_Q(\boldsymbol{I}_Q)$ 是 Q 上定义的严格正函数；
- 确保归一化项 Z 关于 $\boldsymbol{I}$ 的求和或积分在给定的定义域内收敛，而不能发散；
- 确保模型求解的高效性；

为了满足以上约束条件，一般考虑 Boltzmann 分布。于是，将势函数定义为：

$$\psi_Q(\boldsymbol{I}_Q) = \exp(-H_Q(\boldsymbol{I}_Q)) \tag{5}$$

[4]否则，在随机变量较多的情况下，团的数目也较多，如果不使用最大团，那么势必会造成局部势函数个数过多，这将会影响到联合概率的表达简洁性与计算效率。

[5]在概率无向图中，联合概率分布的因子分解形式，由 Hammersley-Clifford 定理保证，详见相关文献。

其中，$H_Q(\boldsymbol{I}_Q)$ 是一个定义在随机变量组 $\boldsymbol{I}_Q$ 上的实值函数，常见的形式为：

$$H_Q(\boldsymbol{I}_Q) = \sum_{u,v\in Q,u\neq v} \alpha_{uv} I_u I_v + \sum_{v\in Q} \beta_v I_v \tag{6}$$

其中，α_{uv} 和 β_v 为参数；上式第 1 项考虑两个节点 u 和 v 之间的关系；上式第 2 项仅考虑单个节点 v。于是，联合概率分布可以写为：

$$\begin{aligned} P(\boldsymbol{I}) &= \frac{\prod\limits_{Q\in\boldsymbol{\mathcal{C}}} \psi_Q(\boldsymbol{I}_Q)}{Z} = \frac{\prod\limits_{Q\in\boldsymbol{\mathcal{C}}} \exp(-H_Q(\boldsymbol{I}_Q))}{Z} \\ &= \frac{\exp\left(-\sum\limits_{Q\in\boldsymbol{\mathcal{C}}} H_Q(\boldsymbol{I}_Q)\right)}{Z} = \frac{\exp\left(-H(\boldsymbol{I})\right)}{Z} \end{aligned} \tag{7}$$

其中，$H(\boldsymbol{I}) = \sum\limits_{Q\in\boldsymbol{\mathcal{C}}} H_Q(\boldsymbol{I}_Q)$，$Z = \sum\limits_{\boldsymbol{I}} \exp\left(-H(\boldsymbol{I})\right)$。

在概率无向图中，如何得到“条件独立性”呢？这可以借助于团与分离集的概念来实现。

定义 1.2（分离集）如果团 A 的节点必须经过团 C 才能到达团 B，那么我们称团 A 与团 B 被团 C 分离，团 C 被称为分离集(*Separating Set*)。

对于 Markov 随机场，在定义了分离集的概念后，可以定义“全局 Markov 性”。

定义 1.3（全局 *Markov* 性）设团 A、团 B 与团 C 对应的随机变量集分别为 $\boldsymbol{I}_A$、$\boldsymbol{I}_B$ 与 $\boldsymbol{I}_C$，如果团 C 是团 A 与团 B 的分离集，那么 $\boldsymbol{I}_A$ 和 $\boldsymbol{I}_B$ 在给定 $\boldsymbol{I}_C$ 的条件下，条件独立，即 $P(\boldsymbol{I}_A, \boldsymbol{I}_B|\boldsymbol{I}_C) = P(\boldsymbol{I}_A|\boldsymbol{I}_C)\cdot P(\boldsymbol{I}_B|\boldsymbol{I}_C)$ 成立，记为 $\boldsymbol{I}_A \perp \boldsymbol{I}_B|\boldsymbol{I}_C$。这个性质被称为“全局 *Markov* 性”(*Global Markov Property*)。

图1-3展示了 Markov 随机场的“条件独立性”。下面，我们将依据基于势函数的联合概率分布，来证明 Markov 随机场确实满足全局 Markov 性。首先，写出联合概率分布：

$$\begin{aligned} P(\boldsymbol{I}) &= P(I_1, I_2, \ldots, I_7) = P(\boldsymbol{I}_A, \boldsymbol{I}_B, \boldsymbol{I}_C) \\ &= \frac{\psi_A(\boldsymbol{I}_A)\psi_B(\boldsymbol{I}_B)\psi_C(\boldsymbol{I}_C)}{\sum\limits_{\boldsymbol{I}} \psi_A(\boldsymbol{I}_A)\psi_B(\boldsymbol{I}_B)\psi_C(\boldsymbol{I}_C)} \\ &= \frac{\psi_A(\boldsymbol{I}_A)\psi_B(\boldsymbol{I}_B)\psi_C(\boldsymbol{I}_C)}{Z} \end{aligned} \tag{8}$$

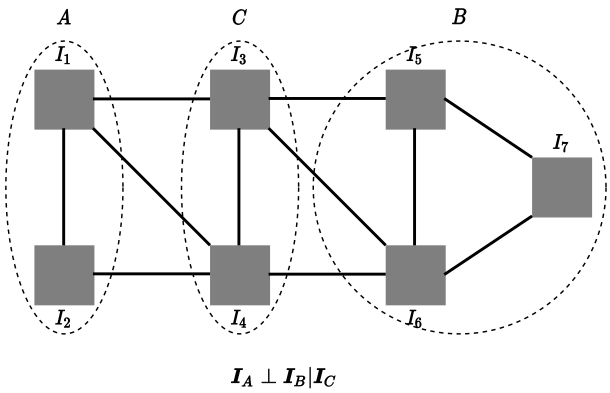

图 1-3: Markov 随机场的“条件独立性”

在给定团 C 的情况下，团 A 与团 B 关于团 C 的条件概率为：

$$P(\boldsymbol{I}_A, \boldsymbol{I}_B | \boldsymbol{I}_C) = \frac{P(\boldsymbol{I}_A, \boldsymbol{I}_B, \boldsymbol{I}_C)}{P(\boldsymbol{I}_C)} = \frac{P(\boldsymbol{I}_A, \boldsymbol{I}_B, \boldsymbol{I}_C)}{\sum\limits_{\boldsymbol{I}_A}\sum\limits_{\boldsymbol{I}_B} P(\boldsymbol{I}_A, \boldsymbol{I}_B, \boldsymbol{I}_C)} \tag{9}$$

继续变换：

$$\begin{aligned}
P(\boldsymbol{I}_A, \boldsymbol{I}_B | \boldsymbol{I}_C) &= \frac{P(\boldsymbol{I}_A, \boldsymbol{I}_B, \boldsymbol{I}_C)}{\sum\limits_{\boldsymbol{I}_A}\sum\limits_{\boldsymbol{I}_B} P(\boldsymbol{I}_A, \boldsymbol{I}_B, \boldsymbol{I}_C)} \\
&= \frac{\frac{1}{Z}\psi_A(\boldsymbol{I}_A)\psi_B(\boldsymbol{I}_B)\psi_C(\boldsymbol{I}_C)}{\sum\limits_{\boldsymbol{I}_A}\sum\limits_{\boldsymbol{I}_B} \frac{1}{Z}\psi_A(\boldsymbol{I}_A)\psi_B(\boldsymbol{I}_B)\psi_C(\boldsymbol{I}_C)} \\
&= \frac{\psi_A(\boldsymbol{I}_A)\psi_B(\boldsymbol{I}_B)\psi_C(\boldsymbol{I}_C)}{\psi_C(\boldsymbol{I}_C)\sum\limits_{\boldsymbol{I}_A}\psi_A(\boldsymbol{I}_A)\sum\limits_{\boldsymbol{I}_B}\psi_B(\boldsymbol{I}_B)} \\
&= \frac{\psi_A(\boldsymbol{I}_A)}{\sum\limits_{\boldsymbol{I}_A}\psi_A(\boldsymbol{I}_A)} \cdot \frac{\psi_B(\boldsymbol{I}_B)}{\sum\limits_{\boldsymbol{I}_B}\psi_B(\boldsymbol{I}_B)}
\end{aligned} \tag{10}$$

在给定团 C 的情况下，团 A 关于团 C 的条件概率为：

$$\begin{aligned}
P(\boldsymbol{I}_A | \boldsymbol{I}_C) &= \frac{P(\boldsymbol{I}_A, \boldsymbol{I}_C)}{P(\boldsymbol{I}_C)} = \frac{\sum\limits_{\boldsymbol{I}_B} P(\boldsymbol{I}_A, \boldsymbol{I}_B, \boldsymbol{I}_C)}{\sum\limits_{\boldsymbol{I}_A}\sum\limits_{\boldsymbol{I}_B} P(\boldsymbol{I}_A, \boldsymbol{I}_B, \boldsymbol{I}_C)} \\
&= \frac{\sum\limits_{\boldsymbol{I}_B} \frac{1}{Z}\psi_A(\boldsymbol{I}_A)\psi_B(\boldsymbol{I}_B)\psi_C(\boldsymbol{I}_C)}{\sum\limits_{\boldsymbol{I}_A}\sum\limits_{\boldsymbol{I}_B} \frac{1}{Z}\psi_A(\boldsymbol{I}_A)\psi_B(\boldsymbol{I}_B)\psi_C(\boldsymbol{I}_C)} \\
&= \frac{\psi_A(\boldsymbol{I}_A)\psi_C(\boldsymbol{I}_C)\sum\limits_{\boldsymbol{I}_B}\psi_B(\boldsymbol{I}_B)}{\psi_C(\boldsymbol{I}_C)\sum\limits_{\boldsymbol{I}_A}\psi_A(\boldsymbol{I}_A)\sum\limits_{\boldsymbol{I}_B}\psi_B(\boldsymbol{I}_B)} \\
&= \frac{\psi_A(\boldsymbol{I}_A)}{\sum\limits_{\boldsymbol{I}_A}\psi_A(\boldsymbol{I}_A)}
\end{aligned} \tag{11}$$

同理，可得：

$$P(\boldsymbol{I}_B|\boldsymbol{I}_C) = \frac{\psi_B(\boldsymbol{I}_B)}{\sum\limits_{\boldsymbol{I}_B} \psi_B(\boldsymbol{I}_B)} \tag{12}$$

于是，得到：

$$P(\boldsymbol{I}_A, \boldsymbol{I}_B|\boldsymbol{I}_C) = P(\boldsymbol{I}_A|\boldsymbol{I}_C) \cdot P(\boldsymbol{I}_B|\boldsymbol{I}_C) \tag{13}$$

根据全局 Markov 性，可以得到 2 个有用的推论，即成对 Markov 性与局部 Markov 性。

定义 1.4（成对 *Markov* 性）在概率无向图中，任意 *2* 个非邻接的节点 u 和 v，在所有其它节点 $\mathcal{O}$ 给定的情况下，它们是条件独立的，即 $P(I_u, I_v|\boldsymbol{I}_{\mathcal{O}}) = P(I_u|\boldsymbol{I}_{\mathcal{O}}) \cdot P(I_v|\boldsymbol{I}_{\mathcal{O}})$，记为 $I_u \perp I_v|\boldsymbol{I}_{\mathcal{O}}$。这个性质被称为“成对 *Markov* 性”（*Pairwise Markov Property*）。

定义 1.5（局部 *Markov* 性）在概率无向图中，节点 u 是任一节点，$n(u)$ 是节点 u 的邻域节点集，$\mathcal{O}$ 是除节点 u 和 $n(u)$ 之外的所有其它节点，那么在 $n(u)$ 给定的情况下，节点 u 与 $\mathcal{O}$ 是条件独立的，即 $P(I_u, \boldsymbol{I}_{\mathcal{O}}|\boldsymbol{I}_{n(u)}) = P(I_u|\boldsymbol{I}_{n(u)}) \cdot P(\boldsymbol{I}_{\mathcal{O}}|\boldsymbol{I}_{n(u)})$，记为 $I_u \perp \boldsymbol{I}_{\mathcal{O}}|\boldsymbol{I}_{n(u)}$。这个性质被称为“局部 *Markov* 性”（*Local Markov Property*）。

全局 Markov 性、局部 Markov 性与成对 Markov 性是等价的。

2 条件随机场

前面介绍的 HMM 模型和 Markov 随机场，分别属于概率有向图模型和概率无向图模型。它们均直接对随机变量的联合概率分布进行建模，属于生成式模型。

下面介绍的条件随机场 (Conditional Random Field, CRF) 虽然也属于概率无向图模型，但是它直接对条件分布建模，属于判别式模型。

与 HMM 模型一样，条件随机场定义了观测序列 $\boldsymbol{O}$ 和状态序列 $\boldsymbol{I}$。但是，条件随机场的建模与求解目标是条件概率分布 $P(\boldsymbol{I}|\boldsymbol{O})$，并且状态序列可以是非线性结构。例如，在自然语言处理 (Natural Language Processing, NLP) 的词性标注任务中，观测数据为单词序列 (语句)，状态序列或标记序列为具有线性结构的词性序列；但是，在语法分析任务中，状态序列为具有树形结构的语法树。

下面给出条件随机场的定义。

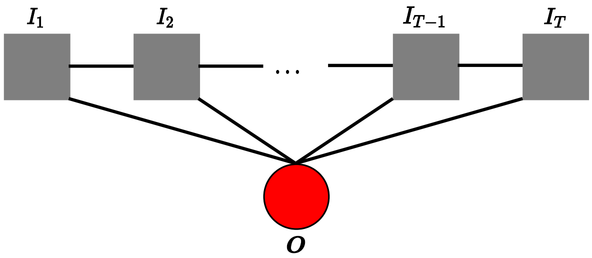

图 2-4: 线性链条件随机场

定义 2.1 （条件随机场）设概率无向图 G 中的节点表示状态 I，如果图 G 中的每个节点 I_u 都满足 *Markov* 性，即图 G 构成一个 *Markov* 随机场：

$$P(I_u|\boldsymbol{O}, \boldsymbol{I}_{V\backslash\{u\}}) = P(I_u|\boldsymbol{O}, I_{n(u)}) \tag{14}$$

其中，$\boldsymbol{O}$ 表示观测序列；$V\backslash\{u\}$ 表示 V 中除 u 之外的所有节点；$n(u)$ 表示 u 的所有邻接节点。那么，我们称图 G 形成一个条件随机场，或者称条件概率分布 $P(\boldsymbol{I}|\boldsymbol{O})$ 形成一个条件随机场，其中 $\boldsymbol{I}$ 是图结构 G 中的节点。

从上述定义可以看出，条件随机场相当于在 Markov 随机场的基础上，给每个节点附加了关于条件 $\boldsymbol{O}$ 的概率分布。我们使用图 G 或 $P(\boldsymbol{I}|\boldsymbol{O})$ 表示条件随机场。

2.1 线性链条件随机场

理论上，条件随机场的图结构可以是任意的。但是，在一些应用中，通常使用所谓的线性链条件随机场 (Linear Chain Conditional Random Field, LC-CRF)。

定义 2.2 （线性链条件随机场）如果条件随机场 $P(\boldsymbol{I}|\boldsymbol{O})$ 满足如下的 *Markov* 性：

$$P(I_t|\boldsymbol{O}, I_1, I_2, \ldots, I_{t-1}, I_{t+1}, \ldots, I_T) = P(I_t|\boldsymbol{O}, I_{t-1}, I_{t+1}) \tag{15}$$

其中，$t = 1, 2, \ldots, T$，T 表示状态序列的长度；当 $t = 1$ 时，左端节点 I_{t-1} 不存在，忽略；当 $t = T$ 时，右端节点 I_{t+1} 不存在，忽略。那么，我们称 $P(\boldsymbol{I}|\boldsymbol{O})$ 为线性链条件随机场。

图2-4展示了线性链条件随机场的示意图。

本质上，线性链条件随机场是一种 Markov 随机场，也需要定义合适的、基于团的局部势函数。但是，与一般的 Markov 随机场不同的是，线性链条件随机场建模的是条件概率 $P(\boldsymbol{I}|\boldsymbol{O})$，而不是联合概率 $P(\boldsymbol{I}, \boldsymbol{O})$。

下面，介绍线性链条件随机场的三种概率表达方式。

2.1.1 基本形式——局部特征函数

在线性链条件随机场中，使用 2 种团，即相邻 2 节点形成的极大团 $\{I_{i-1}, I_i\}$(下文称为双节点团) 和单节点形成的团 $\{I_i\}$(下文称为单节点团)。因此，需要分别基于上述双节点团和单节点团，定义相应的势函数。

在前文，我们已经描述了势函数的选取标准，并给出了 Markov 随机场的联合概率分布 (公式 (7))。对于线性链条件随机场，我们依然选择指数势函数，并分别为双节点团与单节点团引入相应的特征函数 (Feature Function)。于是，条件概率被定义为:

$$P(\boldsymbol{I}|\boldsymbol{O}) = \frac{1}{Z}\exp\left(\sum_k\sum_{\tau=1}^{T-1}\lambda_k t_k(I_\tau, I_{\tau+1}, \boldsymbol{O}, \tau) + \sum_l\sum_{\tau=1}^{T}\mu_l s_l(I_\tau, \boldsymbol{O}, \tau)\right) \tag{16}$$

其中，归一化因子 $Z = \sum_{\boldsymbol{I}}\exp\left(\sum_k\sum_{\tau=1}^{T-1}\lambda_k t_k(I_\tau, I_{\tau+1}, \boldsymbol{O}, \tau) + \sum_l\sum_{\tau=1}^{T}\mu_l s_l(I_\tau, \boldsymbol{O}, \tau)\right)$，确保上式满足概率约束；$t_k(I_\tau, I_{\tau+1}, \boldsymbol{O}, \tau)$ 是定义在相邻节点上的转移特征函数 (Transition Feature Function)，用于刻画相邻节点的相关关系以及观测序列 $\boldsymbol{O}$ 对它们的影响，λ_k 为其参数；$s_l(I_\tau, \boldsymbol{O}, \tau)$ 是定义在单节点上的状态特征函数 (Status Feature Function)，用于刻画单节点的状态以及观测序列 $\boldsymbol{O}$ 对它的影响，μ_l 为其参数。

对照公式 (7)，可得[6]:

$$\begin{aligned}H(\boldsymbol{I}, \boldsymbol{O}) &= \sum_Q H_Q(\boldsymbol{I}_Q, \boldsymbol{O}_Q) \\ &= \sum_k\sum_{\tau=1}^{T-1}\lambda_k t_k(I_\tau, I_{\tau+1}, \boldsymbol{O}, \tau) + \sum_l\sum_{\tau=1}^{T}\mu_l s_l(I_\tau, \boldsymbol{O}, \tau)\end{aligned} \tag{17}$$

其中 Q 表示双节点团与单节点团。这就是线性链条件随机场概率公式 (16) 中指数部分的来源——各团势函数的简单叠加[7]。转移特征函数 t_k 和状态特征函数 s_l

[6] 注意到，公式 (7) 中指数部分的负号，可以融入参数中，可以去掉，不影响功能。

[7] 回想起，在最大熵模型中，条件概率公式为 $P_{\boldsymbol{w}}(y|x) = \frac{1}{Z_{\boldsymbol{w}}(x)}\exp\left(\sum_{i=1}^{n} w_i f_i(x, y)\right)$，其中 $Z_{\boldsymbol{w}}(x)$ 为规范化因子。其形式与线性链条件随机场的概率公式 (16) 非常相似。确实如此，线性链条件随机场结合了 HMM 模型和最大熵模型的特点。与最大熵模型一样，线性链条件随机场也是对数线性模型 (Log Linear Model)。

都依赖于状态序列中的位置，都是局部特征函数。线性链条件随机场完全由特征函数及其相应的参数确定。

显然，要正确地使用和训练线性链条件随机场，就需要定义合适的特征函数。这可以从训练数据集中提取相应的特征来做到这一点。通常，特征函数是实值函数，用来表示数据集中指定特征的经验特性。更特别地，可以简单地使用二值函数，即函数取值 1(条件满足时) 和 0(条件不满足时)。例如，在自然语言处理中，为了完成词性标注任务，首先需要对语料数据集进行预处理，以获取单词序列及其词性的上下文特征 (函数)，然后使用这些预处理数据对线性链条件随机场进行训练，以便学习到合适的模型参数[8]。

2.1.2 简化形式——全局特征函数

注意到，在条件概率的基本形式 (公式 (16)) 中，指数项中的第 1 项和第 2 项，都存在着 2 种类型的求和——特征函数的种类 k 和 l、时序步 τ，并且 t_k 和 s_l 都是局部特征函数。

对此，可以考虑将局部特征函数转换为全局特征函数：对局部特征函数按时序步 τ 求和，从而消去了时序变量 τ，并将指数部分改写为通用的权值向量与特征向量的内积形式。于是，便得到了线性链条件随机场的条件概率简化形式[9]。

对基本形式的指数部分 (公式 (17)) 继续变形：

$$
\begin{aligned}
H(\boldsymbol{I},\boldsymbol{O}) &= \sum_k \sum_{\tau=1}^{T-1} \lambda_k t_k(I_\tau, I_{\tau+1}, \boldsymbol{O}, \tau) + \sum_l \sum_{\tau=1}^{T} \mu_l s_l(I_\tau, \boldsymbol{O}, \tau) \\
&= \sum_k \lambda_k \sum_{\tau=1}^{T-1} t_k(I_\tau, I_{\tau+1}, \boldsymbol{O}, \tau) + \sum_l \mu_l \sum_{\tau=1}^{T} s_l(I_\tau, \boldsymbol{O}, \tau) \\
&= \sum_k \lambda_k t'_k(\boldsymbol{I},\boldsymbol{O}) + \sum_l \mu_l s'_l(\boldsymbol{I},\boldsymbol{O}) \\
&= \sum_{k'=1}^{K+L} w_{k'} f_{k'}(\boldsymbol{I},\boldsymbol{O}) \\
&= \boldsymbol{w}_{k'}^T \boldsymbol{f}_{k'}(\boldsymbol{I},\boldsymbol{O})
\end{aligned}
\tag{18}
$$

其中，转移特征函数 t_k 的种类数为 K，状态特征函数 s_l 的种类数为 L。全局权

[8]具体示例，详见附录“线性链条件随机场实验”。

[9]此时，线性链条件随机场的条件概率简化公式，在形式上，更加接近于最大熵模型的条件概率公式。

值参数为：

$$w_{k'} = \begin{cases} \lambda_k & k' = k = 1, 2, \dots, K \\ \mu_l & k' = K + l,\ l = 1, 2, \dots, L \end{cases} \tag{19}$$

其向量形式为 $\boldsymbol{w}_{k'}$。全局特征函数为：

$$f_{k'}(\boldsymbol{I}, \boldsymbol{O}) = \begin{cases} t'_k(\boldsymbol{I}, \boldsymbol{O}) = \sum_{\tau=1}^{T-1} t_k(I_\tau, I_{\tau+1}, \boldsymbol{O}, \tau) & k' = k = 1, 2, \dots, K \\ s'_l(\boldsymbol{I}, \boldsymbol{O}) = \sum_{\tau=1}^{T} s_l(I_\tau, \boldsymbol{O}, \tau) & k' = K + l,\ l = 1, 2, \dots, L \end{cases} \tag{20}$$

其向量形式为 $\boldsymbol{f}_{k'}(\boldsymbol{I}, \boldsymbol{O})$。

于是，得到线性链条件随机场的简化形式：

$$\begin{aligned} P(\boldsymbol{I}|\boldsymbol{O}) &= \frac{1}{Z} \exp H(\boldsymbol{I}, \boldsymbol{O}) \\ &= \frac{1}{Z} \exp \left(\sum_{k'=1}^{K+L} w_{k'} f_{k'}(\boldsymbol{I}, \boldsymbol{O}) \right) \\ &= \frac{1}{Z} \exp \left(\boldsymbol{w}_{k'}^T \boldsymbol{f}_{k'}(\boldsymbol{I}, \boldsymbol{O}) \right) \end{aligned} \tag{21}$$

其中，归一化因子：

$$\begin{aligned} Z &= \sum_{\boldsymbol{I}} \exp \left(\sum_{k'=1}^{K+L} w_{k'} f_{k'}(\boldsymbol{I}, \boldsymbol{O}) \right) \\ &= \sum_{\boldsymbol{I}} \exp \left(\boldsymbol{w}_{k'}^T \boldsymbol{f}_{k'}(\boldsymbol{I}, \boldsymbol{O}) \right) \end{aligned} \tag{22}$$

2.1.3 矩阵形式

在条件概率的基本形式 (公式 (16)) 中，指数项中的第 1 项和第 2 项，都存在着 2 种类型的求和——特征函数的种类 k 和 l、时序步 τ。如果对特征函数先按时序步 τ 求和 (消去 τ)，就得到了简化形式，上一节已经对此进行了讨论。实际上，也可以对特征函数先按种类数求和，并进行适当的变形，从而得到矩阵形式。

观察到，转移特征函数 t_k 与状态特征函数 s_l 在 τ 范围上，具有差异性，前者 $\tau = 1, 2 \dots T - 1$，后者 $\tau = 1, 2 \dots T$。因此，为了统一 τ 的范围，需要对状态节点进行扩展：为每个状态序列 $\boldsymbol{I}$ 引入 1 个特殊的虚拟节点 I_{T+1}[10]。

[10]下文将表明，它是一个真的虚拟节点——它的转移特征函数可以取任何实值，甚至可以是 0。因此，该节点的引入，不会改变线性链条件随机场的功能，即它的存在只是形式上需要而已。在算法实现时，完全可以被忽略掉。例如，当 $\tau = T$ 时，下文中的矩阵 $M_{\boldsymbol{O},\tau}$，将只包含状态特征函数 s_l 的信息，而状态转移特征函数 t_k 的信息将被全部忽略。

在此条件下，考虑对下面的基本形式进行恒等变换：

$$\begin{aligned}
P(\boldsymbol{I}|\boldsymbol{O}) &= \frac{1}{Z}\exp H(\boldsymbol{I},\boldsymbol{O}) \\
&= \frac{1}{Z}\exp\left(\sum_k\sum_{\tau=1}^{T-1}\lambda_k t_k(I_\tau, I_{\tau+1}, \boldsymbol{O}, \tau) + \sum_l\sum_{\tau=1}^{T}\mu_l s_l(I_\tau, \boldsymbol{O}, \tau)\right) \\
&= \frac{\exp\left(\sum\limits_k\sum\limits_{\tau=1}^{T-1}\lambda_k t_k(I_\tau, I_{\tau+1}, \boldsymbol{O}, \tau) + \sum\limits_l\sum\limits_{\tau=1}^{T}\mu_l s_l(I_\tau, \boldsymbol{O}, \tau)\right)}{\sum\limits_{\boldsymbol{I}}\exp\left(\sum\limits_k\sum\limits_{\tau=1}^{T-1}\lambda_k t_k(I_\tau, I_{\tau+1}, \boldsymbol{O}, \tau) + \sum\limits_l\sum\limits_{\tau=1}^{T}\mu_l s_l(I_\tau, \boldsymbol{O}, \tau)\right)}
\end{aligned} \tag{23}$$

上式中，Z 为归一化项，$\boldsymbol{I}$ 为原始未扩展的状态序列。下面，使用扩展状态序列，并令 $t_k(I_T, I_{T+1}, \boldsymbol{O}, \tau) = 1(\forall k = 1, 2, \ldots, K)$[11]。于是，针对扩展后的状态序列：

$$\begin{aligned}
P(\boldsymbol{I}|\boldsymbol{O}) &= \frac{\exp\left(\sum\limits_k\sum\limits_{\tau=1}^{T-1}\lambda_k t_k(I_\tau, I_{\tau+1}, \boldsymbol{O}, \tau) + \sum\limits_l\sum\limits_{\tau=1}^{T}\mu_l s_l(I_\tau, \boldsymbol{O}, \tau)\right)}{\sum\limits_{\boldsymbol{I}}\exp\left(\sum\limits_k\sum\limits_{\tau=1}^{T-1}\lambda_k t_k(I_\tau, I_{\tau+1}, \boldsymbol{O}, \tau) + \sum\limits_l\sum\limits_{\tau=1}^{T}\mu_l s_l(I_\tau, \boldsymbol{O}, \tau)\right)} \\
&= \frac{\exp\left(\sum\limits_k\sum\limits_{\tau=1}^{T-1}\lambda_k t_k(I_\tau, I_{\tau+1}, \boldsymbol{O}, \tau) + \sum\limits_l\sum\limits_{\tau=1}^{T}\mu_l s_l(I_\tau, \boldsymbol{O}, \tau)\right)}{\sum\limits_{\boldsymbol{I}}\exp\left(\sum\limits_k\sum\limits_{\tau=1}^{T-1}\lambda_k t_k(I_\tau, I_{\tau+1}, \boldsymbol{O}, \tau) + \sum\limits_l\sum\limits_{\tau=1}^{T}\mu_l s_l(I_\tau, \boldsymbol{O}, \tau)\right)} \cdot \frac{\exp\left(\sum\limits_k\lambda_k\right)}{\exp\left(\sum\limits_k\lambda_k\right)} \\
&= \frac{\exp\left(\sum\limits_k\sum\limits_{\tau=1}^{T-1}\lambda_k t_k(I_\tau, I_{\tau+1}, \boldsymbol{O}, \tau) + \sum\limits_l\sum\limits_{\tau=1}^{T}\mu_l s_l(I_\tau, \boldsymbol{O}, \tau)\right)}{\sum\limits_{\boldsymbol{I}}\exp\left(\sum\limits_k\sum\limits_{\tau=1}^{T-1}\lambda_k t_k(I_\tau, I_{\tau+1}, \boldsymbol{O}, \tau) + \sum\limits_l\sum\limits_{\tau=1}^{T}\mu_l s_l(I_\tau, \boldsymbol{O}, \tau)\right)} \cdot \\
&\quad \frac{\exp\left(\sum\limits_k\lambda_k t_k(I_T, I_{T+1}, \boldsymbol{O}, \tau)\right)}{\exp\left(\sum\limits_k\lambda_k t_k(I_T, I_{T+1}, \boldsymbol{O}, \tau)\right)} \\
&= \frac{\exp\left(\sum\limits_k\sum\limits_{\tau=1}^{T}\lambda_k t_k(I_\tau, I_{\tau+1}, \boldsymbol{O}, \tau) + \sum\limits_l\sum\limits_{\tau=1}^{T}\mu_l s_l(I_\tau, \boldsymbol{O}, \tau)\right)}{\sum\limits_{\boldsymbol{I}}\exp\left(\sum\limits_k\sum\limits_{\tau=1}^{T}\lambda_k t_k(I_\tau, I_{\tau+1}, \boldsymbol{O}, \tau) + \sum\limits_l\sum\limits_{\tau=1}^{T}\mu_l s_l(I_\tau, \boldsymbol{O}, \tau)\right)} \\
&= \frac{1}{Z}\exp\left(\sum_k\sum_{\tau=1}^{T}\lambda_k t_k(I_\tau, I_{\tau+1}, \boldsymbol{O}, \tau) + \sum_l\sum_{\tau=1}^{T}\mu_l s_l(I_\tau, \boldsymbol{O}, \tau)\right)
\end{aligned} \tag{24}$$

[11] 可以取任何实值，为方便，令其等于 1。

上式就是时序步 τ 统一之后的条件概率[12]。下面，对它进行进一步的变换：

$$
\begin{aligned}
P(\boldsymbol{I}|\boldsymbol{O}) &= \frac{1}{Z}\exp\left(\sum_k\sum_{\tau=1}^{T}\lambda_k t_k(I_\tau, I_{\tau+1}, \boldsymbol{O}, \tau) + \sum_l\sum_{\tau=1}^{T}\mu_l s_l(I_\tau, \boldsymbol{O}, \tau)\right) \\
&= \frac{1}{Z}\exp\sum_{\tau=1}^{T}\left(\sum_k\lambda_k t_k(I_\tau, I_{\tau+1}, \boldsymbol{O}, \tau) + \sum_l\mu_l s_l(I_\tau, \boldsymbol{O}, \tau)\right) \\
&= \frac{1}{Z}\prod_{\tau=1}^{T}\exp\left(\sum_k\lambda_k t_k(I_\tau, I_{\tau+1}, \boldsymbol{O}, \tau) + \sum_l\mu_l s_l(I_\tau, \boldsymbol{O}, \tau)\right) \\
&= \frac{1}{Z}\prod_{\tau=1}^{T}\exp\left(\sum_{k'=1}^{K+L} w_{k'} f_{k'}(I_\tau, I_{\tau+1}, \boldsymbol{O}, \tau)\right)
\end{aligned} \tag{25}
$$

其中，转移特征函数 t_k 的种类数为 K，状态特征函数 s_l 的种类数为 L。权值参数被定义为：

$$
w_{k'} = \begin{cases} \lambda_k & k' = k = 1, 2, \ldots, K \\ \mu_l & k' = K + l,\ l = 1, 2, \ldots, L \end{cases} \tag{26}
$$

特征函数 $f_{k'}(I_\tau, I_{\tau+1}, \boldsymbol{O}, \tau)$ 被定义为：

$$
f_{k'}(I_\tau, I_{\tau+1}, \boldsymbol{O}, \tau) = \begin{cases} t_k(I_\tau, I_{\tau+1}, \boldsymbol{O}, \tau) & k' = k = 1, 2, \ldots, K \\ s_l(I_\tau, \boldsymbol{O}, \tau) & k' = K + l,\ l = 1, 2, \ldots, L \end{cases} \tag{27}
$$

为了便于使用矩阵形式来表示条件概率 $P(\boldsymbol{I}|\boldsymbol{O})$，对状态 I_τ 的 N 个可能取值，使用索引 (从 1 开始编号) 来表示。图2-5展示了一个按值展开的状态序列 $\boldsymbol{I}$。其中，水平方向为时序步，垂直方向为状态的取值 (索引)；状态的取值个数为 $N = 3$，原始时序步数为 $T = 3$。

于是，令矩阵 $M_{\boldsymbol{O},\tau}$ 为：

$$
M_{\boldsymbol{O},\tau} = \left[\exp\left(\sum_{k'=1}^{K+L} w_{k'} f_{k'}(I_\tau, I_{\tau+1}, \boldsymbol{O}, \tau)\right)\right]_{N\times N} \tag{28}
$$

可以看出，矩阵 $M_{\boldsymbol{O},\tau}$ 的维度为 $N \times N$，其中 N 表示 I_τ 和 $I_{\tau+1}$ 的取值个数。例如，$M_{\boldsymbol{O},\tau}\left[I_\tau = 1, I_{\tau+1} = 2\right]$ 表示矩阵 $M_{\boldsymbol{O},\tau}$ 的第 1 行第 2 列的元素：

$$
M_{\boldsymbol{O},\tau}\left[I_\tau = 1, I_{\tau+1} = 2\right] = \exp\left(\sum_{k'=1}^{K+L} w_{k'} f_{k'}(I_\tau = 1, I_{\tau+1} = 2, \boldsymbol{O}, \tau)\right) \tag{29}
$$

[12]为方便，仍然使用 Z 表示新的归一化项。

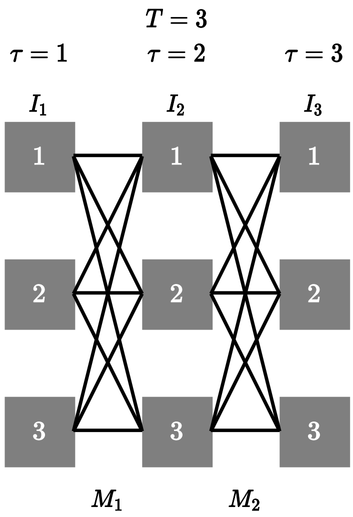

图 2-5: 一个按值展开的状态序列示意图

因此，继续对 $P(\boldsymbol{I}|\boldsymbol{O})$ 进行变形，得到：

$$\begin{aligned}P(\boldsymbol{I}|\boldsymbol{O}) &= \frac{1}{Z}\prod_{\tau=1}^{T}\exp\left(\sum_{k'=1}^{K+L} w_{k'}f_{k'}(I_\tau, I_{\tau+1}, \boldsymbol{O}, \tau)\right) \\ &= \frac{1}{Z}\prod_{\tau=1}^{T} M_{\boldsymbol{O},\tau}\left[I_\tau, I_{\tau+1}\right]\end{aligned} \tag{30}$$

其中：

$$Z = \sum_{\boldsymbol{I}}\prod_{\tau=1}^{T} M_{\boldsymbol{O},\tau}\left[I_\tau, I_{\tau+1}\right] \tag{31}$$

归一化项 Z 的计算公式还不够简洁，需要进一步处理。

考虑任意 2 个矩阵 M_τ 与 $M_{\tau+1}$ 的乘积，如图2-6所示。在该图左上角，有 2 个矩阵 M_τ 和 $M_{\tau+1}$，它们的乘积结果记为矩阵 $M_\tau M_{\tau+1}$。于是，根据矩阵乘法，矩阵元素 $M_\tau M_{\tau+1}(11)$ 为：

$$M_\tau M_{\tau+1}(11) = M_\tau(11)M_{\tau+1}(11) + M_\tau(12)M_{\tau+1}(21) + M_\tau(13)M_{\tau+1}(31) \tag{32}$$

其几何意义为，从时刻 τ 取值为 1 的节点到时刻 $\tau+2$ 取值为 1 的节点，按所有路径上的代价乘积之和作为后面这个节点的值。进一步地，如果需要统计从时刻 τ 取值为 1 的节点到时刻 $\tau+2$ 的所有取值或节点的总值，则需要在时刻 $\tau+2$ 之后添加 1 个汇出总节点 $stop$，且汇总到节点 $stop$ 的所有路径代价均为 1，它们形成一个全 1 列向量；同理，如果需要统计从时刻 τ 的所有取值或节点到时刻 $\tau+2$

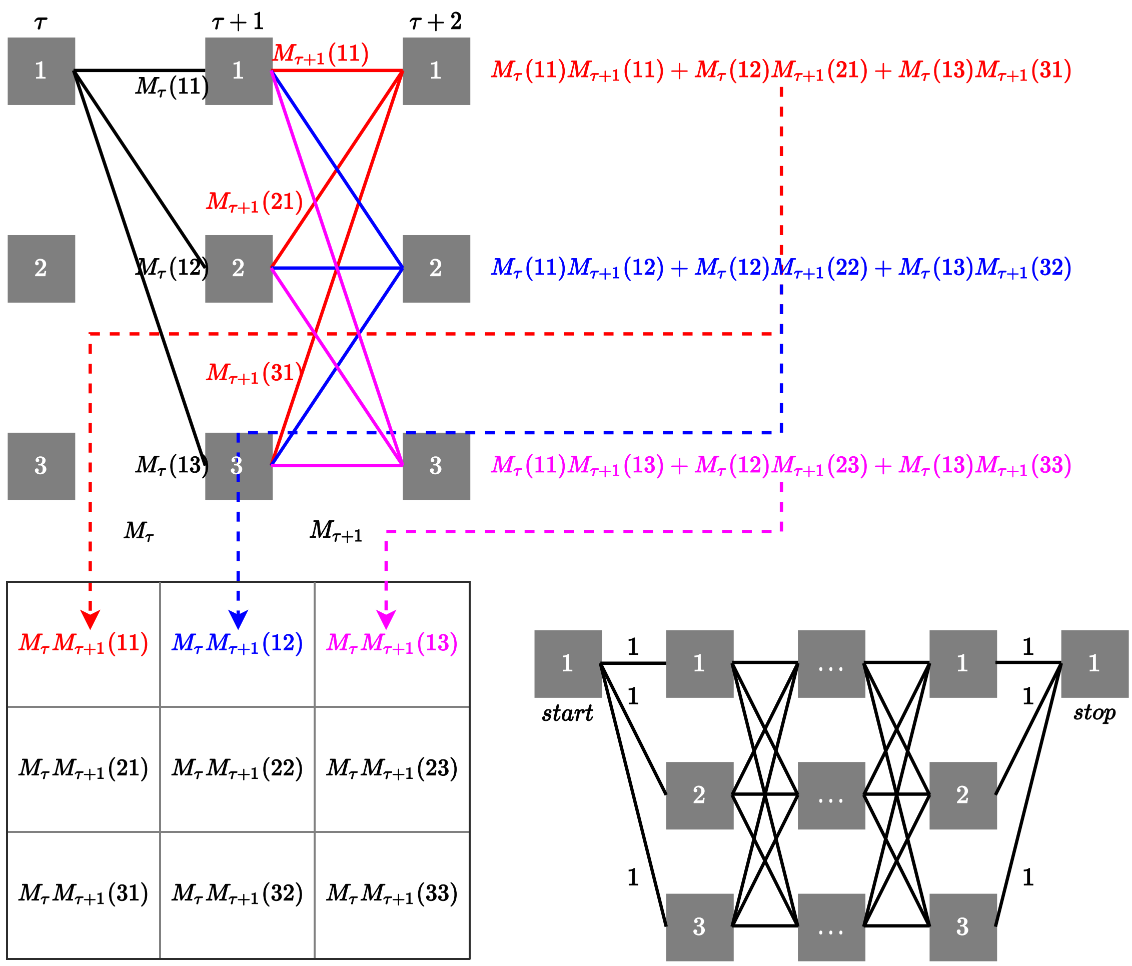

图 2-6: 2 个矩阵乘积的几何意义

的所有取值或节点的总值 (即统计所有路径的代价乘积之和)，则还需要在时刻 τ 之前添加 1 个汇入总节点 $start$，且汇总到节点 $start$ 的所有路径代价均为 1，它们形成一个全 1 行向量。示意图如图2-6右下角所示。

对于任意多个矩阵的乘积，也可以得出相同的结论。于是，归一化项 Z 可以表示为：

$$\begin{aligned} Z &= \sum_{\boldsymbol{I}} \prod_{\tau=1}^{T} M_{\boldsymbol{O},\tau}\left[I_\tau, I_{\tau+1}\right] \\ &= \mathbf{1}^T \left(\prod_{\tau=1}^{T} M_{\boldsymbol{O},\tau} \right) \mathbf{1} \end{aligned} \tag{33}$$

或者：

$$Z = \left(M_{start} \left(\prod_{\tau=1}^{T} M_{\boldsymbol{O},\tau} \right) M_{stop} \right) [1,1] \tag{34}$$

其中，M_{start} 为 $N \times N$ 矩阵：

$$M_{start} = \begin{bmatrix} 1 & 1 & \dots & 1 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 \end{bmatrix} \tag{35}$$

M_{stop} 为 $N \times N$ 矩阵：

$$M_{stop} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & \dots & 0 \end{bmatrix} \tag{36}$$

3 概率计算

在实际应用中，也需要计算线性链条件随机场的各种概率。例如，计算 $P(I_\tau = q_i|\boldsymbol{O})$ 和 $P(I_\tau = q_i, I_{\tau+1} = q_j|\boldsymbol{O})$(其中，$i,j = 1,2,\dots,N$，$N$ 表示状态个数或取值个数) 等。与 HMM 模型一样，不使用低效的直接计算方法，而是使用前向-后向算法：构造递推迭代公式，高效地进行计算。

3.1 前向-后向算法

实际上，从归一化项 Z 的计算公式 (33)，可以分别得到前向与后向递推迭代公式及其前向-后向算法。下面，首先考察前向向量。

定义前向向量 $\boldsymbol{\alpha}_\tau(\boldsymbol{O})$ 为 N 维列向量，它的分量 $\boldsymbol{\alpha}_{\tau,i}(\boldsymbol{O})$ 表示时刻 τ 处于状态 q_i 或状态取值为 i(其中 $i=1,2,\ldots,N$，N 为状态个数或取值个数) 且观测序列为 $O_1,O_2,\ldots,O_\tau$ 的非规范概率[13]，其中 $\tau=1,2\ldots,T+1$。

从公式 (33) 出发，初始化前向向量 ($\tau=1$)：

$$\boldsymbol{\alpha}_1(\boldsymbol{O})=\mathbf{1} \tag{37}$$

为其它时序构造递推公式：

$$\boldsymbol{\alpha}_{\tau+1}^{\mathrm{T}}(\boldsymbol{O})=\boldsymbol{\alpha}_\tau^{\mathrm{T}}(\boldsymbol{O})M_{\boldsymbol{O},\tau} \tag{38}$$

因此，归一化因子 Z 可以表示为：

$$Z=\boldsymbol{\alpha}_{T+1}^{\mathrm{T}}(\boldsymbol{O})\mathbf{1} \tag{39}$$

类似地，可以定义后向向量 $\boldsymbol{\beta}_\tau(\boldsymbol{O})$ 为 N 维列向量，它的分量 $\boldsymbol{\beta}_{\tau,i}(\boldsymbol{O})$ 表示时刻 τ 处于状态 q_i 或状态取值为 i(其中 $i=1,2,\ldots,N$，N 为状态个数或取值个数) 且观测序列为 $O_{\tau+1},\ldots,O_T$ 的非规范概率，其中 $\tau=T,T-1,\ldots,0$。

同样，从公式 (33) 出发，初始化后向向量 ($\tau=T$)：

$$\boldsymbol{\beta}_T(\boldsymbol{O})=\mathbf{1} \tag{40}$$

为其它时序构造递推公式：

$$\boldsymbol{\beta}_\tau(\boldsymbol{O})=M_{\boldsymbol{O},\tau+1}\boldsymbol{\beta}_{\tau+1}(\boldsymbol{O}) \tag{41}$$

因此，归一化因子 Z 可以表示为：

$$Z=\mathbf{1}^{\mathrm{T}}\boldsymbol{\beta}_0(\boldsymbol{O}) \tag{42}$$

3.2 常见概率的计算

在定义了前向与后向 (非规范) 概率向量之后，下面给出一些概率计算的实例：

- 单状态的概率

[13]即 $P(\boldsymbol{I}|\boldsymbol{O})$ 公式的分母项。

在给定观测序列 $\boldsymbol{O}$、转移特征函数 t_k 与状态特征函数 s_l 及其相应的权值 λ_k 和 μ_l 的情况下，计算时刻 τ 处于状态 q_i 的概率 $P(I_\tau = q_i|\boldsymbol{O})$，并令 $\gamma_\tau(q_i) = P(I_\tau = q_i|\boldsymbol{O})$：

$$\gamma_\tau(q_i) = P(I_\tau = q_i|\boldsymbol{O}) = \frac{\boldsymbol{\alpha}_{\tau,i}^{\mathrm{T}}(\boldsymbol{O})\boldsymbol{\beta}_{\tau,i}(\boldsymbol{O})}{Z} \tag{43}$$

其中，$Z = \boldsymbol{\alpha}_{T+1}^{\mathrm{T}}(\boldsymbol{O})\mathbf{1}$，或者 $Z = \mathbf{1}^{\mathrm{T}}\boldsymbol{\beta}_0(\boldsymbol{O})$。

- 双状态的概率

  在给定观测序列 $\boldsymbol{O}$、转移特征函数 t_k 与状态特征函数 s_l 及其相应的权值 λ_k 和 μ_l 的情况下，计算时刻 τ 处于状态 q_i 和 $\tau+1$ 处于状态 q_j 的概率 $P(I_\tau = q_i, I_{\tau+1} = q_j|\boldsymbol{O})$，并令 $\xi_\tau(q_i,q_j) = P(I_\tau = q_i, I_{\tau+1} = q_j|\boldsymbol{O})$：

$$\xi_\tau(q_i,q_j) = P(I_\tau = q_i, I_{\tau+1} = q_j|\boldsymbol{O}) = \frac{\boldsymbol{\alpha}_{\tau,i}^{\mathrm{T}}(\boldsymbol{O})M_{\boldsymbol{O},\tau}\left[i,j\right]\boldsymbol{\beta}_{\tau+1,j}(\boldsymbol{O})}{Z} \tag{44}$$

  其中，$Z = \boldsymbol{\alpha}_{T+1}^{\mathrm{T}}(\boldsymbol{O})\mathbf{1}$，或者 $Z = \mathbf{1}^{\mathrm{T}}\boldsymbol{\beta}_0(\boldsymbol{O})$。

与 HMM 模型类似，还可以利用 γ_τ 和 ξ_τ 计算出一些有用的信息。具体方法，详见 HMM 模型部分，此处不再赘述。

3.3 特征函数期望值的计算

利用单状态与双状态概率公式[14]，可以计算全局特征函数 $f_{k'}(\boldsymbol{I},\boldsymbol{O})$(公式 (20)) 关于条件概率分布 $P(\boldsymbol{I}|\boldsymbol{O})$ 和联合概率分布 $P(\boldsymbol{I},\boldsymbol{O})$ 的数学期望：

- 全局特征函数 $f_{k'}(\boldsymbol{I},\boldsymbol{O})$ 关于条件概率分布 $P(\boldsymbol{I}|\boldsymbol{O})$ 的数学期望

  在给定观测序列 $\boldsymbol{O}$、转移特征函数 t_k 与状态特征函数 s_l 及其相应的权值 λ_k 和 μ_l 的情况下，其数学期望为：

$$\begin{aligned}\mathbb{E}_{P(\boldsymbol{I}|\boldsymbol{O})}\left[f_{k'}\right] &= \sum_{\boldsymbol{I}} P(\boldsymbol{I}|\boldsymbol{O})f_{k'}(\boldsymbol{I},\boldsymbol{O}) \\ &= \sum_{\boldsymbol{I}}\sum_{\tau} P(I_\tau, I_{\tau+1}|\boldsymbol{O})f_{k'}(I_\tau, I_{\tau+1}, \boldsymbol{O}, \tau) \\ &= \sum_{\boldsymbol{I}}\sum_{\tau} \frac{\boldsymbol{\alpha}_{\tau,I_\tau}^{\mathrm{T}}(\boldsymbol{O})M_{\boldsymbol{O},\tau}\left[I_\tau, I_{\tau+1}\right]\boldsymbol{\beta}_{\tau+1,I_{\tau+1}}(\boldsymbol{O})}{Z} \cdot f_{k'}(I_\tau, I_{\tau+1}, \boldsymbol{O}, \tau)\end{aligned} \tag{45}$$

[14]虽然完全可以只使用双状态概率公式来表示，但需要注意的是，在时序步右端边界，即 $\tau = T$ 时刻，根据前面的描述，只考虑状态特征函数 s_l，而忽略掉转移特征函数 t_k。此时，实际上在使用单状态概率公式。

其中，$k'=1,2,\ldots,K+L$，K 为转移特征函数 t_k 的个数，L 为状态特征函数 s_l 的个数；$\tau=1,2,\ldots,T$。上式还可以进一步细化，当 $k'=k=1,2,\ldots,K$ 时：

$$
\begin{aligned}
\mathbb{E}_{P(\boldsymbol{I}|\boldsymbol{O})}\left[f_{k'}\right] &= \sum_{\boldsymbol{I}}\sum_{\tau}\frac{\boldsymbol{\alpha}_{\tau,I_\tau}^{\mathrm{T}}(\boldsymbol{O})M_{\boldsymbol{O},\tau}\left[I_\tau,I_{\tau+1}\right]\boldsymbol{\beta}_{\tau+1,I_{\tau+1}}(\boldsymbol{O})}{Z}\cdot f_{k'}(I_\tau,I_{\tau+1},\boldsymbol{O},\tau) \\
&= \sum_{\boldsymbol{I}}\sum_{\tau=1}^{T-1}\frac{\boldsymbol{\alpha}_{\tau,I_\tau}^{\mathrm{T}}(\boldsymbol{O})M_{\boldsymbol{O},\tau}\left[I_\tau,I_{\tau+1}\right]\boldsymbol{\beta}_{\tau+1,I_{\tau+1}}(\boldsymbol{O})}{Z}\cdot t_k\left(I_\tau,I_{\tau+1},\boldsymbol{O},\tau\right)
\end{aligned}
\tag{46}
$$

当 $k'=K+l,\ l=1,2,\ldots,L$ 时：

$$
\begin{aligned}
\mathbb{E}_{P(\boldsymbol{I}|\boldsymbol{O})}\left[f_{k'}\right] &= \sum_{\boldsymbol{I}}\sum_{\tau}\frac{\boldsymbol{\alpha}_{\tau,I_\tau}^{\mathrm{T}}(\boldsymbol{O})M_{\boldsymbol{O},\tau}\left[I_\tau,I_{\tau+1}\right]\boldsymbol{\beta}_{\tau+1,I_{\tau+1}}(\boldsymbol{O})}{Z}\cdot f_{k'}(I_\tau,I_{\tau+1},\boldsymbol{O},\tau) \\
&= \sum_{\boldsymbol{I}}\sum_{\tau=1}^{T}\frac{\boldsymbol{\alpha}_{\tau,I_\tau}^{\mathrm{T}}(\boldsymbol{O})\boldsymbol{\beta}_{\tau,I_\tau}(\boldsymbol{O})}{Z}\cdot s_l\left(I_\tau,\boldsymbol{O},\tau\right)
\end{aligned}
\tag{47}
$$

即依据 k' 的值，分别求取转移特征函数和状态特征函数的条件期望。

- 全局特征函数 $f_{k'}(\boldsymbol{I},\boldsymbol{O})$ 关于联合概率分布 $P(\boldsymbol{I},\boldsymbol{O})$ 的数学期望

  在给定观测序列 $\boldsymbol{O}$、转移特征函数 t_k 与状态特征函数 s_l 及其相应的权值 λ_k 和 μ_l 的情况下，其数学期望为：

$$
\begin{aligned}
\mathbb{E}_{P(\boldsymbol{I},\boldsymbol{O})}\left[f_{k'}\right] &= \sum_{\boldsymbol{O}}\sum_{\boldsymbol{I}}P(\boldsymbol{I},\boldsymbol{O})f_{k'}(\boldsymbol{I},\boldsymbol{O}) \\
&= \sum_{\boldsymbol{O}}\sum_{\boldsymbol{I}}\tilde{P}(\boldsymbol{O})P(\boldsymbol{I}|\boldsymbol{O})f_{k'}(\boldsymbol{I},\boldsymbol{O}) \\
&= \sum_{\boldsymbol{O}}\tilde{P}(\boldsymbol{O})\sum_{\boldsymbol{I}}P(\boldsymbol{I}|\boldsymbol{O})f_{k'}(\boldsymbol{I},\boldsymbol{O})
\end{aligned}
\tag{48}
$$

  其中，$\tilde{P}(\boldsymbol{O})$ 为经验分布，直接从数据集中计算得到；$\sum\limits_{\boldsymbol{I}}P(\boldsymbol{I}|\boldsymbol{O})f_{k'}(\boldsymbol{I},\boldsymbol{O})=\mathbb{E}_{P(\boldsymbol{I}|\boldsymbol{O})}\left[f_{k'}\right]$ 为全局特征函数 $f_{k'}(\boldsymbol{I},\boldsymbol{O})$ 关于条件分布 $P(\boldsymbol{I}|\boldsymbol{O})$ 的数学期望，可参考公式 (45)、公式 (46) 和公式 (47)。

4 预测算法

预测算法的目标是，在给定观测序列 $\boldsymbol{O}$、转移特征函数 t_k 与状态特征函数 s_l 及其相应的权值 λ_k 和 μ_l 的情况下[15]，求解出与观测序列 $\boldsymbol{O}$ 匹配的最佳状态序列 $\boldsymbol{I}^*$，即：

$$\boldsymbol{I}^* = \arg\max_{\boldsymbol{I}} P(\boldsymbol{I}|\boldsymbol{O}) \tag{49}$$

4.1 贪心算法

与 HMM 模型类似，贪心算法简单地选取 τ 时刻条件概率 $P(I_\tau = q_i|\boldsymbol{O})$ 最大的状态作为“最优”状态，即：

$$I_\tau^* = \arg\max_{q_i} \gamma_\tau(q_i) \quad \tau = 1, 2, \ldots, T \tag{50}$$

其中，$\gamma_\tau(q_i)$ 的定义参见公式 (43)。

贪心算法的优缺点非常明显：算法实现简洁高效；但是，在大多数情况下，只能获得局部最优解，并不能获得全局最优解，且只有在满足贪心选择性质时，整体最优解与局部最优解才一致。此外，求解出的“最优”状态序列，可能存在实际情况中并不存在的相邻状态。

4.2 动态规划：Viterbi 算法

针对预测算法的目标 (公式 (49))，为方便，将 $P(\boldsymbol{I}|\boldsymbol{O})$ 的简化形式 (公式 (21)) 代入，可得：

$$\begin{aligned}
\boldsymbol{I}^* &= \arg\max_{\boldsymbol{I}} P(\boldsymbol{I}|\boldsymbol{O}) \\
&= \arg\max_{\boldsymbol{I}} \frac{1}{Z} \exp\left(\boldsymbol{w}_{k'}^T \boldsymbol{f}_{k'}(\boldsymbol{I}, \boldsymbol{O})\right) \\
&= \arg\max_{\boldsymbol{I}} \boldsymbol{w}_{k'}^T \boldsymbol{f}_{k'}(\boldsymbol{I}, \boldsymbol{O})
\end{aligned} \tag{51}$$

其中，$\boldsymbol{w}_{k'}$ 为全局权值参数向量，其每个分量为：

$$w_{k'} = \begin{cases} \lambda_k & k' = k = 1, 2, \ldots, K \\ \mu_l & k' = K + l,\ l = 1, 2, \ldots, L \end{cases} \tag{52}$$

[15] 在给定这些条件的情况下，可以依据前向-后向算法，计算出条件概率分布 $P(\boldsymbol{I}|\boldsymbol{O})$ 及其它概率与期望。此外，特征函数 t_k 与 s_l 及其权值 λ_k 和 μ_l，被称为模型参数。

其中，转移特征函数 t_k 的种类数为 K，状态特征函数 s_l 的种类数为 L。$\boldsymbol{f}_{k'}(\boldsymbol{I},\boldsymbol{O})$ 为全局特征函数的向量形式，其每个分量为：

$$f_{k'}(\boldsymbol{I},\boldsymbol{O})=\begin{cases} t'_k(\boldsymbol{I},\boldsymbol{O})=\sum_{\tau=1}^{T-1} t_k(I_\tau,I_{\tau+1},\boldsymbol{O},\tau) & k'=k=1,2,\dots,K \\ s'_l(\boldsymbol{I},\boldsymbol{O})=\sum_{\tau=1}^{T} s_l(I_\tau,\boldsymbol{O},\tau) & k'=K+l,\ l=1,2,\dots,L \end{cases} \tag{53}$$

从公式 (51) 可以看出，在给定观测序列 $\boldsymbol{O}$、转移特征函数 t_k 与状态特征函数 s_l 及其相应的权值 λ_k 和 μ_l 的情况下，预测算法的目标由确定最优条件概率 $P(\boldsymbol{I}^*|\boldsymbol{O})$ 所对应的最优状态序列 $\boldsymbol{I}^*$，转变为确定权值-特征函数的最优内积 $\boldsymbol{w}_{k'}^T\boldsymbol{f}_{k'}(\boldsymbol{I}^*,\boldsymbol{O})$ 所对应的最优状态序列 $\boldsymbol{I}^*$。

如果将序列 $\boldsymbol{I}=(I_1,I_2,\dots,I_T)$ 看作一条路径，那么预测问题将演变为求解 $\boldsymbol{w}_{k'}^T\boldsymbol{f}_{k'}(\boldsymbol{I},\boldsymbol{O})$ 值最大的那条路径。而路径求解问题，满足最优性原理或最优子结构性质——整体解与子问题的部分解满足一致性原理。于是，可以使用动态规划进行求解。

因此，依据动态规划的算法流程，我们可以构造求解最优子序列的递推迭代公式，然后从初值 ($\tau=1$ 时刻) 开始，迭代应用递推式，直至 $\tau=T$ 为止。于是，便获得了最优路径值及其最优路径或序列 I^*。当然，在搜索的过程中，需要保存搜索路径上每个节点的父节点；并且在算法搜索停止后，从路径终点开始回溯，以便得到最优序列 I^*。

从上面的分析可以看出，为了应用动态规划，需要针对时序步 τ 构造递推迭代公式。因此，令：

$$f_{k'}(I_\tau,I_{\tau+1},\boldsymbol{O},\tau)=\begin{cases} t_k(I_\tau,I_{\tau+1},\boldsymbol{O},\tau) & k'=k=1,2,\dots,K \\ s_l(I_\tau,\boldsymbol{O},\tau) & k'=K+l,\ l=1,2,\dots,L \end{cases} \tag{54}$$

其中，$\tau=1,2,\dots,T$，且令 $t_k(I_T,I_{T+1},\boldsymbol{O},\tau)\equiv 0$[16]。其相应的向量形式为 $\boldsymbol{f}_{k'}(I_\tau,I_{\tau+1},\boldsymbol{O})=(f_1(I_\tau,I_{\tau+1},\boldsymbol{O},\tau),f_2(I_\tau,I_{\tau+1},\boldsymbol{O},\tau),\dots,f_{K+L}(I_\tau,I_{\tau+1},\boldsymbol{O},\tau))^{\mathrm{T}}$。

[16] 转移特征函数 t_k 的时序 τ 范围为 $1,2,\dots,T-1$。此处，将其时序范围补足至 T，仅形式上需要而已。实际上，前文分析表明，$t_k(I_T,I_{T+1},\boldsymbol{O},\tau)$ 可以取任何实值，不会影响结果的正确性。为了计算方便，将其设置为 0。

继续对公式 (51) 进行变形：

$$
\begin{aligned}
\boldsymbol{I}^* &= \arg\max_{\boldsymbol{I}} \boldsymbol{w}_{k'}^T \boldsymbol{f}_{k'}(\boldsymbol{I}, \boldsymbol{O}) \\
&= \arg\max_{\boldsymbol{I}} \boldsymbol{w}_{k'}^T \left(f_1(\boldsymbol{I}, \boldsymbol{O}), f_2(\boldsymbol{I}, \boldsymbol{O}), \ldots, f_{K+L}(\boldsymbol{I}, \boldsymbol{O})\right)^{\mathrm{T}} \\
&= \arg\max_{\boldsymbol{I}} \boldsymbol{w}_{k'}^T \left(\sum_{\tau=1}^{T} f_1(I_\tau, I_{\tau+1}, \boldsymbol{O}, \tau), \ldots, \sum_{\tau=1}^{T} f_{K+L}(I_\tau, I_{\tau+1}, \boldsymbol{O}, \tau)\right)^{\mathrm{T}} \\
&= \arg\max_{\boldsymbol{I}} \boldsymbol{w}_{k'}^T \sum_{\tau=1}^{T} \left(f_1(I_\tau, I_{\tau+1}, \boldsymbol{O}, \tau), \ldots, f_{K+L}(I_\tau, I_{\tau+1}, \boldsymbol{O}, \tau)\right)^{\mathrm{T}} \\
&= \arg\max_{\boldsymbol{I}} \sum_{\tau=1}^{T} \boldsymbol{w}_{k'}^T \boldsymbol{f}_{k'}(I_\tau, I_{\tau+1}, \boldsymbol{O})
\end{aligned}
\tag{55}
$$

于是，根据动态规划的最优性原理，依时序 τ 逐步求解最优子问题。设 $\delta_{\tau-1}(q_j)$ 表示时刻 τ 到达状态 q_j 的最优路径值，那么在 $\tau+1$ 时刻：

$$
\delta_\tau(q_i) = \max_{j=1,2,\ldots,N} \{\delta_{\tau-1}(q_j) + \boldsymbol{w}_{k'}^T \boldsymbol{f}_{k'}(I_\tau = q_j, I_{\tau+1} = q_i, \boldsymbol{O})\} \tag{56}
$$

据此，设置初值 $\delta_0(q_i)$ 为：

$$
\delta_0(q_i) = 0 \tag{57}
$$

其中，$i = 1, 2, \ldots, N$，N 表示状态个数。然后，设 $\psi_\tau(q_i)$ 表示时刻 $\tau+1$ 处于状态 q_i 时所对应的最优子序列 (或路径)$I_1^*, I_2^*, \ldots, I_\tau^*$ 中的终端节点 (即 I_τ^*) 为：

$$
\begin{aligned}
\psi_\tau(q_i) &= I_\tau^* \\
&= \arg\max_{q_j} \{\delta_{\tau-1}(q_j) + \boldsymbol{w}_{k'}^T \boldsymbol{f}_{k'}(I_\tau = q_j, I_{\tau+1} = q_i, \boldsymbol{O})\}
\end{aligned}
\tag{58}
$$

其中，$j = 1, 2, \ldots, N$。其初始值设置为：

$$
\psi_0(q_i) = 0 \quad i = 1, 2, \ldots, N \tag{59}
$$

原因是，路径起始点没有父节点。而在 $\tau = T$ 时刻[17]：

$$
I_T^* = \arg\max_{q_j} \{\delta_{T-1}(q_j) + \boldsymbol{w}_{k'}^T \boldsymbol{f}_{k'}(I_T = q_j, I_{T+1} = q_i, \boldsymbol{O})\} \tag{60}
$$

I_T^* 即为整个最优序列或路径的终端节点，从它出发，利用公式 (58) 进行回溯，即可得到最优路径。

下面给出 Viterbi 算法。

[17]正如前文所述，在 $\tau = T$ 时刻，转移特征函数 t_k 被设置为 0。

算法 **4.1**（*Viterbi* 算法）

Input:

Model Parameters: t_k and λ_k, s_l and μ_l, $k=1,2,\ldots,K$, $l=1,2,\ldots,L$

Observation Sequence: $\boldsymbol{O}=(O_1,O_2,\ldots,O_T)$

Output:

Best Path: $\boldsymbol{I}^*=(I_1^*,I_2^*,\ldots,I_T^*)$

Algorithm:

$\delta_0(q_i)=0 \quad i=1,2,\ldots,N$

$\psi_0(q_i)=0 \quad i=1,2,\ldots,N$

for $\tau=1,2\cdots T$:

$\delta_\tau(q_i)=\max\limits_{j=1,2,\ldots,N}\{\delta_{\tau-1}(q_j)+\boldsymbol{w}_{k'}^T\boldsymbol{f}_{k'}(I_\tau=q_j,I_{\tau+1}=q_i,\boldsymbol{O})\}$

$\psi_\tau(q_i)=\arg\max\limits_{q_j}\{\delta_{\tau-1}(q_j)+\boldsymbol{w}_{k'}^T\boldsymbol{f}_{k'}(I_\tau=q_j,I_{\tau+1}=q_i,\boldsymbol{O})\}$

$I_T^*=\arg\max\limits_{q_j}\{\delta_{T-1}(q_j)+\boldsymbol{w}_{k'}^T\boldsymbol{f}_{k'}(I_T=q_j,I_{T+1}=q_i,\boldsymbol{O})\}$

for $t=T-1\cdots 1$:

$I_t^*=\psi_{t+1}(I_{t+1}^*)$

return $\boldsymbol{I}^*=(I_1^*,I_2^*,\ldots,I_T^*)$

5 学习算法

在给定训练数据集时，即每个样本既包括观测序列 $\boldsymbol{O}$，又包括状态序列 $\boldsymbol{I}$ 时，如何估计线性链条件随机场的权值参数 λ_k 和 μ_l 呢?

线性链条件随机场的条件概率具有多种等价形式，下面对简化形式 (公式 (21)) 展开讨论。为讨论方便，将相关公式、权值参数及特征函数复述如下：

$$\begin{aligned} P(\boldsymbol{I}|\boldsymbol{O}) &= \frac{1}{Z}\exp\left(\sum_{k'=1}^{K+L} w_{k'} f_{k'}(\boldsymbol{I},\boldsymbol{O})\right) \\ &= \frac{1}{Z}\exp\left(\boldsymbol{w}_{k'}^T\boldsymbol{f}_{k'}(\boldsymbol{I},\boldsymbol{O})\right) \end{aligned} \tag{61}$$

其中，$\boldsymbol{w}_{k'}$ 为全局权值参数向量，其每个分量为：

$$w_{k'}=\begin{cases}\lambda_k & k'=k=1,2,\ldots,K \\ \mu_l & k'=K+l,\ l=1,2,\ldots,L\end{cases} \tag{62}$$

其中，转移特征函数 t_k 的种类数为 K，状态特征函数 s_l 的种类数为 L。$\boldsymbol{f}_{k'}(\boldsymbol{I},\boldsymbol{O})$

为全局特征函数的向量形式，其每个分量为：

$$
f_{k'}(\boldsymbol{I},\boldsymbol{O})=\begin{cases} t'_k(\boldsymbol{I},\boldsymbol{O})=\sum_{\tau=1}^{T-1} t_k(I_\tau,I_{\tau+1},\boldsymbol{O},\tau) & k'=k=1,2,\dots,K \\ s'_l(\boldsymbol{I},\boldsymbol{O})=\sum_{\tau=1}^{T} s_l(I_\tau,\boldsymbol{O},\tau) & k'=K+l,\ l=1,2,\dots,L \end{cases} \tag{63}
$$

在上述定义下，线性链条件随机场的学习目标是，确定全局权值参数 $\boldsymbol{w}_{k'}$。

观察线性链条件随机场的条件概率公式 (61)，可以发现，形式上，它与最大熵模型的条件概率公式非常相似。确实如此，实际上，最大熵模型是一种对数线性模型，而线性链条件随机场是定义在时序数据上的对数线性模型——通过在时序上求和，将局部特征函数转换为全局特征函数，并通过 Markov 随机场的势函数，将两者进行了紧密的关联。

既然如此，用于求解最大熵模型的优化方法，也能够用于求解线性链条件随机场。常见的方法有梯度下降法[18]、牛顿法、拟牛顿法和改进的迭代尺度法等。

5.1 改进的迭代尺度法

下面，直接将“最大熵模型”一章中关于改进迭代尺度法 (Improved Iterative Scaling, IIS) 的相关结论，移植到线性链条件随机场中，详细的推导过程可以参见“最大熵模型”的相关章节[19]，此处不再赘述。

为了显式表示优化对象 $\boldsymbol{w}_{k'}$，对相关公式添加相应的标识：

$$
\begin{aligned} P_{\boldsymbol{w}_{k'}}(\boldsymbol{I}|\boldsymbol{O}) &= \frac{1}{Z_{\boldsymbol{w}_{k'}}(\boldsymbol{O})}\exp\left(\sum_{k'=1}^{K+L} w_{k'} f_{k'}(\boldsymbol{I},\boldsymbol{O})\right) \\ &= \frac{1}{Z_{\boldsymbol{w}_{k'}}(\boldsymbol{O})}\exp\left(\boldsymbol{w}_{k'}^T \boldsymbol{f}_{k'}(\boldsymbol{I},\boldsymbol{O})\right) \end{aligned} \tag{64}
$$

其中：

$$
Z_{\boldsymbol{w}_{k'}}(\boldsymbol{O})=\sum_{\boldsymbol{I}}\exp\left(\sum_{k'=1}^{K+L} w_{k'} f_{k'}(\boldsymbol{I},\boldsymbol{O})\right)=\sum_{\boldsymbol{I}}\exp\left(\boldsymbol{w}_{k'}^T \boldsymbol{f}_{k'}(\boldsymbol{I},\boldsymbol{O})\right) \tag{65}
$$

[18]在“最大熵模型”一章中，已经介绍了梯度下降法。对于线性链条件随机场而言，也有非常相似的结论。

[19]推导过程非常相似，详见《机器学习》课程系列之《最大熵模型》，Chapter7-CN.pdf。

在给定训练数据的情况下，其对数似然函数可以写为[20]：

$$
\begin{aligned}
L(\boldsymbol{w}_{k'}) = L_{\tilde{P}}(P_{\boldsymbol{w}_{k'}}) &= \log \prod_{O,I} P_{\boldsymbol{w}_{k'}}(\boldsymbol{I}|\boldsymbol{O})^{\tilde{P}(\boldsymbol{O},\boldsymbol{I})} \\
&= \sum_{O,I} \tilde{P}(\boldsymbol{O},\boldsymbol{I}) \log P_{\boldsymbol{w}_{k'}}(\boldsymbol{I}|\boldsymbol{O})
\end{aligned}
\tag{66}
$$

将 $P_{\boldsymbol{w}_{k'}}(\boldsymbol{I}|\boldsymbol{O})$ 的公式 (64) 代入上式，得到：

$$
\begin{aligned}
L(\boldsymbol{w}_{k'}) &= \sum_{O,I} \tilde{P}(\boldsymbol{O},\boldsymbol{I}) \log P_{\boldsymbol{w}_{k'}}(\boldsymbol{I}|\boldsymbol{O}) \\
&= \sum_{O,I} \tilde{P}(\boldsymbol{O},\boldsymbol{I}) \log \left(\frac{1}{Z_{\boldsymbol{w}_{k'}}(\boldsymbol{O})} \exp\left(\boldsymbol{w}_{k'}^T \boldsymbol{f}_{k'}(\boldsymbol{I},\boldsymbol{O}) \right) \right) \\
&= \sum_{O,I} \tilde{P}(\boldsymbol{O},\boldsymbol{I}) \boldsymbol{w}_{k'}^T \boldsymbol{f}_{k'}(\boldsymbol{I},\boldsymbol{O}) - \sum_{O,I} \tilde{P}(\boldsymbol{O},\boldsymbol{I}) \log Z_{\boldsymbol{w}_{k'}}(\boldsymbol{O}) \\
&= \sum_{O,I} \tilde{P}(\boldsymbol{O},\boldsymbol{I}) \sum_{\tau=1}^{T} \boldsymbol{w}_{k'}^T \boldsymbol{f}_{k'}(I_\tau, I_{\tau+1}, \boldsymbol{O}) - \sum_{O} \tilde{P}(\boldsymbol{O}) \log Z_{\boldsymbol{w}_{k'}}(\boldsymbol{O})
\end{aligned}
\tag{67}
$$

其中，最后一步的推导，利用了公式 (55) 中的结论：

$$
\boldsymbol{w}_{k'}^T \boldsymbol{f}_{k'}(\boldsymbol{I},\boldsymbol{O}) = \sum_{\tau=1}^{T} \boldsymbol{w}_{k'}^T \boldsymbol{f}_{k'}(I_\tau, I_{\tau+1}, \boldsymbol{O}) \tag{68}
$$

下面，直接对照“最大熵模型”章节中的 IIS 算法，引入全局特征计数函数 $f^{\#}(\boldsymbol{I},\boldsymbol{O})$：

$$
f^{\#}(\boldsymbol{I},\boldsymbol{O}) = \sum_{k'=1}^{K+L} f_{k'}(\boldsymbol{I},\boldsymbol{O}) = \sum_{k'=1}^{K+L} \sum_{\tau=1}^{T} f_{k'}(I_\tau, I_{\tau+1}, \boldsymbol{O}, \tau) \tag{69}
$$

上式表示样本点 $(\boldsymbol{O},\boldsymbol{I})$ 中出现的特征值总和。如果 $f_{k'}(I_\tau, I_{\tau+1}, \boldsymbol{O}, \tau)$ 为二值函数 (即 0 或 1)，那么 $f^{\#}(\boldsymbol{I},\boldsymbol{O})$ 就表示样本点 $(\boldsymbol{O},\boldsymbol{I})$ 中出现的特征函数的总个数。

于是，直接求解下面的方程，从而得到全局权值参数 $w_{k'}$ 的增量 $\delta_{k'}$：

$$
\sum_{O,I} \tilde{P}(\boldsymbol{O}) P_{\boldsymbol{w}_{k'}}(\boldsymbol{I}|\boldsymbol{O}) f_{k'}(\boldsymbol{I},\boldsymbol{O}) \exp\left(\delta_{k'} f^{\#}(\boldsymbol{I},\boldsymbol{O}) \right) = \mathbb{E}_{\tilde{P}}(f_{k'}(\boldsymbol{I},\boldsymbol{O})) \tag{70}
$$

其中，$\mathbb{E}_{\tilde{P}}(f_{k'}(\boldsymbol{I},\boldsymbol{O}))$ 表示全局特征函数 $f_{k'}(\boldsymbol{I},\boldsymbol{O})$ 关于联合分布 $\tilde{P}(\boldsymbol{O},\boldsymbol{I})$ 的期望：

$$
\mathbb{E}_{\tilde{P}}(f_{k'}(\boldsymbol{I},\boldsymbol{O})) = \sum_{O,I} \tilde{P}(\boldsymbol{O},\boldsymbol{I}) f_{k'}(\boldsymbol{I},\boldsymbol{O}) \tag{71}
$$

[20]需要注意的是，与“最大熵模型”章节中的情形类似，除非特别说明，否则，在本章求和或求积公式中出现的符号“O, I”，应被理解为枚举数据集中所有的特征对，而不是枚举所有的样本点对 $(\boldsymbol{O},\boldsymbol{I})$。因此，在它们出现的公式中，与之相关的联合概率、条件概率等项的含义也随之被确定了。

公式 (70) 还可以进一步细化。考虑到：

$$f_{k'}(\boldsymbol{I},\boldsymbol{O})=\begin{cases} t'_k(\boldsymbol{I},\boldsymbol{O})=\sum\limits_{\tau=1}^{T-1}t_k(I_\tau,I_{\tau+1},\boldsymbol{O},\tau) & k'=k=1,2,\ldots,K \\ s'_l(\boldsymbol{I},\boldsymbol{O})=\sum\limits_{\tau=1}^{T}s_l(I_\tau,\boldsymbol{O},\tau) & k'=K+l,\ l=1,2,\ldots,L \end{cases} \tag{72}$$

那么，将上述转移特征函数与状态特征函数分别代入公式 (70) 中，就可以分别求解出相应的 $\delta_{k'}$：

$$\begin{aligned}
&\sum_{O,I}\tilde{P}(\boldsymbol{O})P_{\boldsymbol{w}_{k'}}(\boldsymbol{I}|\boldsymbol{O})\sum_{\tau=1}^{T-1}t_k(I_\tau,I_{\tau+1},\boldsymbol{O},\tau)\exp\left(\delta_k f^{\#}(\boldsymbol{I},\boldsymbol{O})\right)=\sum_{O,I}\tilde{P}(\boldsymbol{O},\boldsymbol{I})\sum_{\tau=1}^{T-1}t_k(I_\tau,I_{\tau+1},\boldsymbol{O},\tau)\\
&\sum_{O,I}\tilde{P}(\boldsymbol{O})P_{\boldsymbol{w}_{k'}}(\boldsymbol{I}|\boldsymbol{O})\sum_{\tau=1}^{T}s_l(I_\tau,\boldsymbol{O},\tau)\exp\left(\delta_l f^{\#}(\boldsymbol{I},\boldsymbol{O})\right)=\sum_{O,I}\tilde{P}(\boldsymbol{O},\boldsymbol{I})\sum_{\tau=1}^{T}s_l(I_\tau,\boldsymbol{O},\tau)
\end{aligned} \tag{73}$$

如果所有样本点中出现的全局特征函数的个数一致，那么 $f^{\#}(\boldsymbol{I},\boldsymbol{O})$ 是一个常量，令 $f^{\#}(\boldsymbol{I},\boldsymbol{O})=\mathrm{M}$，则可直接从公式 (73) 中求解出相应的 $\delta_{k'}$，得到：

$$\delta_{k'}=\begin{cases} \delta_k=\dfrac{1}{\mathrm{M}}\log\dfrac{\mathbb{E}_{\tilde{P}}(t'_k(\boldsymbol{I},\boldsymbol{O}))}{\mathbb{E}_{P}(t'_k(\boldsymbol{I},\boldsymbol{O}))}=\dfrac{1}{\mathrm{M}}\log\dfrac{\mathbb{E}_{\tilde{P}}(\sum\limits_{\tau=1}^{T-1}t_k(I_\tau,I_{\tau+1},\boldsymbol{O},\tau))}{\mathbb{E}_{P}(\sum\limits_{\tau=1}^{T-1}t_k(I_\tau,I_{\tau+1},\boldsymbol{O},\tau))} \\ \delta_l=\dfrac{1}{\mathrm{M}}\log\dfrac{\mathbb{E}_{\tilde{P}}(s'_l(\boldsymbol{I},\boldsymbol{O}))}{\mathbb{E}_{P}(s'_l(\boldsymbol{I},\boldsymbol{O}))}=\dfrac{1}{\mathrm{M}}\log\dfrac{\mathbb{E}_{\tilde{P}}(\sum\limits_{\tau=1}^{T}s_l(I_\tau,\boldsymbol{O},\tau))}{\mathbb{E}_{P}(\sum\limits_{\tau=1}^{T}s_l(I_\tau,\boldsymbol{O},\tau))} \end{cases} \tag{74}$$

其中，$\mathbb{E}_P(f_{k'}(\boldsymbol{I},\boldsymbol{O}))$ 表示全局特征函数 $f_{k'}(\boldsymbol{I},\boldsymbol{O})$ 关于联合分布 $\tilde{P}(\boldsymbol{O})P_{\boldsymbol{w}_{k'}}(\boldsymbol{I}|\boldsymbol{O})$ 的期望：

$$\mathbb{E}_P(f_{k'}(\boldsymbol{I},\boldsymbol{O}))=\sum_{O,I}\tilde{P}(\boldsymbol{O})P_{\boldsymbol{w}_{k'}}(\boldsymbol{I}|\boldsymbol{O})f_{k'}(\boldsymbol{I},\boldsymbol{O}) \tag{75}$$

将上式中的 $f_{k'}(\boldsymbol{I},\boldsymbol{O})$，分别使用公式 (72) 中的 t_k 和 s_l 替换后，就得到了公式 (74)。

如果训练集中所有样本点的全局特征函数的个数不一致，即 $f^{\#}(\boldsymbol{I},\boldsymbol{O})$ 不是一个常量，那么，此时可以考虑定义所谓的松弛特征：

$$s(\boldsymbol{I},\boldsymbol{O})=\mathrm{S}-f^{\#}(\boldsymbol{I},\boldsymbol{O})\geqslant 0\quad\forall(\boldsymbol{I},\boldsymbol{O}) \tag{76}$$

其中，S 是一个足够大的常数，使得对训练集中的任意样本点，$s(\boldsymbol{I},\boldsymbol{O})\geqslant 0$ 总成立。使用松弛常数 S 取代前面的常数 M，就得到了 S 算法。

在 S 算法中，要求常数 S 足够大。这样，每步迭代的增量 $\delta_{k'}$ 将变得比较小，从而导致算法收敛比较慢。

在此情况下，可以考虑使用数值迭代的方法计算出 $\delta_{k'}$。例如，使用牛顿法 (即 Newton-Raphson 方法)，令公式 (70) 所表示的方程为：

$$g(\delta_{k'}) = \sum_{O,I} \tilde{P}(\boldsymbol{O}) P_{\boldsymbol{w}_{k'}}(\boldsymbol{I}|\boldsymbol{O}) f_{k'}(\boldsymbol{I},\boldsymbol{O}) \exp\left(\delta_{k'} f^{\#}(\boldsymbol{I},\boldsymbol{O})\right) - \mathbb{E}_{\tilde{P}}(f_{k'}(\boldsymbol{I},\boldsymbol{O})) = 0 \quad (77)$$

迭代公式为：

$$\delta_{k'}^{(t+1)} = \delta_{k'}^{(t)} - \frac{g\left(\delta_{k'}^{(t)}\right)}{g'\left(\delta_{k'}^{(t)}\right)} \quad (78)$$

方程 $g(\delta_{k'})$ 有单根。因此，只要适当地选取初始值 $\delta_{k'}^{(0)}$，牛顿法将会很快收敛而得到解 $\delta_{k'}^{*}$。

下面给出 IIS 算法。

算法 5.1（*IIS* 算法）

Input:

Feature Function: t_k and s_l, $k = 1, 2, \ldots, K$, $l = 1, 2, \ldots, L$

Train Dataset: $\{(\boldsymbol{O}_1, \boldsymbol{I}_1), (\boldsymbol{O}_2, \boldsymbol{I}_2), \ldots, \}$

Output:

Model Parameters $\boldsymbol{w}_{k'}^{*}$: λ_k and μ_l, $k = 1, 2, \ldots, K$, $l = 1, 2, \ldots, L$

Algorithm:

$w_{k'}^{(1)} = 0$, $k' = 1, 2, \ldots, K + L$

for $t = 1, 2 \cdots$ T:

for $k' = 1, 2 \cdots K + L$:

if $k' \in 1 \cdots K$:

$$\sum_{O,I} \tilde{P}(\boldsymbol{O}) P_{\boldsymbol{w}_{k'}}(\boldsymbol{I}|\boldsymbol{O}) \sum_{\tau=1}^{T-1} t_k(I_\tau, I_{\tau+1}, \boldsymbol{O}, \tau) \exp\left(\delta_k f^{\#}(\boldsymbol{I},\boldsymbol{O})\right)$$
$$= \sum_{O,I} \tilde{P}(\boldsymbol{O},\boldsymbol{I}) \sum_{\tau=1}^{T-1} t_k(I_\tau, I_{\tau+1}, \boldsymbol{O}, \tau) \quad \Rightarrow \quad \delta_k$$

else:

$$\sum_{O,I} \tilde{P}(\boldsymbol{O}) P_{\boldsymbol{w}_{k'}}(\boldsymbol{I}|\boldsymbol{O}) \sum_{\tau=1}^{T} s_l(I_\tau, \boldsymbol{O}, \tau) \exp\left(\delta_l f^{\#}(\boldsymbol{I},\boldsymbol{O})\right)$$
$$= \sum_{O,I} \tilde{P}(\boldsymbol{O},\boldsymbol{I}) \sum_{\tau=1}^{T} s_l(I_\tau, \boldsymbol{O}, \tau) \quad \Rightarrow \quad \delta_l$$

$w_{k'}^{(t+1)} = w_{k'}^{(t)} + \delta_{k'}$

5.2 拟牛顿法：BFGS 算法

在牛顿法中，需要计算 Hessian 矩阵的逆矩阵。当自变量的维度很大时，所需的时间与空间都很多。

拟牛顿法 (Quasi-Newton Method) 采取近似计算的方式，避免直接计算 Hessian 矩阵。它的基本思想是，从某个初始的正定矩阵开始，迭代地近似计算出 Hessian 矩阵或逆矩阵，然后作用于自变量，对其进行优化[21]。

BFGS(Broyden-Fletcher-Goldfarb-Shanno) 算法是最流行的拟牛顿算法之一，它对 Hessian 矩阵进行近似计算。下面，讨论针对线性链条件随机场的 BFGS 算法。

对于线性链条件随机场 (公式 (64))：

$$P_{\boldsymbol{w}_{k'}}(\boldsymbol{I}|\boldsymbol{O}) = \frac{1}{Z_{\boldsymbol{w}_{k'}}(\boldsymbol{O})} \exp\left(\sum_{k'=1}^{K+L} w_{k'} f_{k'}(\boldsymbol{I}, \boldsymbol{O})\right) \tag{79}$$

待优化的目标函数 (公式67) 为[22]：

$$\begin{aligned} \mathrm{L}(\boldsymbol{w}_{k'}) = -L(\boldsymbol{w}_{k'}) &= -\left(\sum_{O,I} \tilde{P}(\boldsymbol{O}, \boldsymbol{I}) \boldsymbol{w}_{k'}^T \boldsymbol{f}_{k'}(\boldsymbol{I}, \boldsymbol{O}) - \sum_{O} \tilde{P}(\boldsymbol{O}) \log Z_{\boldsymbol{w}_{k'}}(\boldsymbol{O})\right) \\ &= \sum_{O} \tilde{P}(\boldsymbol{O}) \log Z_{\boldsymbol{w}_{k'}}(\boldsymbol{O}) - \sum_{O,I} \tilde{P}(\boldsymbol{O}, \boldsymbol{I}) \sum_{k'=1}^{K+L} w_{k'} f_{k'}(\boldsymbol{I}, \boldsymbol{O}) \end{aligned} \tag{80}$$

其中：

$$Z_{\boldsymbol{w}_{k'}}(\boldsymbol{O}) = \sum_{\boldsymbol{I}} \exp\left(\sum_{k'=1}^{K+L} w_{k'} f_{k'}(\boldsymbol{I}, \boldsymbol{O})\right) \tag{81}$$

为应用 BFGS 算法，需要求解偏导数 $\frac{\partial \mathrm{L}(\boldsymbol{w}_{k'})}{\partial w_{k'}}$：

$$\begin{aligned} \frac{\partial \mathrm{L}(\boldsymbol{w}_{k'})}{\partial w_{k'}} &= \sum_{O} \tilde{P}(\boldsymbol{O}) \frac{\sum_{\boldsymbol{I}} \exp\left(\sum_{k'=1}^{K+L} w_{k'} f_{k'}(\boldsymbol{I}, \boldsymbol{O})\right)}{Z_{\boldsymbol{w}_{k'}}(\boldsymbol{O})} f_{k'}(\boldsymbol{I}, \boldsymbol{O}) - \sum_{O,I} \tilde{P}(\boldsymbol{O}, \boldsymbol{I}) f_{k'}(\boldsymbol{I}, \boldsymbol{O}) \\ &= \sum_{O} \tilde{P}(\boldsymbol{O}) \sum_{\boldsymbol{I}} P_{\boldsymbol{w}_{k'}}(\boldsymbol{I}|\boldsymbol{O}) f_{k'}(\boldsymbol{I}, \boldsymbol{O}) - \sum_{O,I} \tilde{P}(\boldsymbol{O}, \boldsymbol{I}) f_{k'}(\boldsymbol{I}, \boldsymbol{O}) \\ &= \sum_{O,I} \tilde{P}(\boldsymbol{O}) P_{\boldsymbol{w}_{k'}}(\boldsymbol{I}|\boldsymbol{O}) f_{k'}(\boldsymbol{I}, \boldsymbol{O}) - \sum_{O,I} \tilde{P}(\boldsymbol{O}, \boldsymbol{I}) f_{k'}(\boldsymbol{I}, \boldsymbol{O}) \\ &= \mathbb{E}_P(f_{k'}(\boldsymbol{I}, \boldsymbol{O})) - \mathbb{E}_{\tilde{P}}(f_{k'}(\boldsymbol{I}, \boldsymbol{O})) \end{aligned} \tag{82}$$

[21]关于拟牛顿法的详细讨论，请阅读《机器学习》课程系列之基础知识，Chapter1-CN.pdf。

[22]按照惯例，将最大化问题转换为等价的最小化问题。

下面给出 BFGS 算法。

算法 5.2（*BFGS* 算法）

Input:

Global Feature Function: $f_{k'}$, $k' = 1, 2, \ldots, K + L$

Train Dataset: $\{(\boldsymbol{O}_1, \boldsymbol{I}_1), (\boldsymbol{O}_2, \boldsymbol{I}_2), \ldots,\}$

Output:

Model Parameters: $w^*_{k'}$, $k' = 1, 2, \ldots, K + L$

Algorithm:

Initialize randomly: $w^{(1)}_{k'}$, $k' = 1, 2, \ldots, K + L$

$\boldsymbol{B}_1 = \boldsymbol{I}$

for $t = 1, 2 \cdots T$:

for $k' = 1, 2 \cdots K + L$:

$$\nabla \mathrm{L}(\boldsymbol{w}^{(t)}_{k'}) = \frac{\partial \mathrm{L}(\boldsymbol{w}^{(t)}_{k'})}{\partial w^{(t)}_{k'}} = \mathbb{E}_P(f_{k'}(\boldsymbol{I}, \boldsymbol{O})) - \mathbb{E}_{\tilde{P}}(f_{k'}(\boldsymbol{I}, \boldsymbol{O}))$$

$$\boldsymbol{B}_t \Delta w^{(t)}_{k'} = -\nabla \mathrm{L}(\boldsymbol{w}^{(t)}_{k'}) \quad \Rightarrow \quad \Delta w^{(t)}_{k'}$$

Line Search for best $\lambda^*_{k'}$:

$$\mathrm{L}(w^{(t)}_{k'} + \lambda^*_{k'} \Delta w^{(t)}_{k'}) = \min_{\lambda \geqslant 0} \mathrm{L}(w^{(t)}_{k'} + \lambda \Delta w^{(t)}_{k'})$$

$$w^{(t+1)}_{k'} = w^{(t)}_{k'} + \lambda^*_{k'} \Delta w^{(t)}_{k'}$$

$$\nabla \mathrm{L}(\boldsymbol{w}^{(t+1)}_{k'}) = \frac{\partial \mathrm{L}(\boldsymbol{w}^{(t+1)}_{k'})}{\partial w^{(t+1)}_{k'}} = \mathbb{E}_P(f_{k'}(\boldsymbol{I}, \boldsymbol{O})) - \mathbb{E}_{\tilde{P}}(f_{k'}(\boldsymbol{I}, \boldsymbol{O}))$$

$$\mathbf{B}_{t+1} = \mathbf{B}_t + \frac{\Delta\mathbf{G}\Delta\mathbf{G}^T}{\Delta\mathbf{G}^T\Delta\mathbf{W}} - \frac{\mathbf{B}_t\Delta\mathbf{W}(\mathbf{B}_t\Delta\mathbf{W})^T}{(\mathbf{B}_t\Delta\mathbf{W})^T\Delta\mathbf{W}}$$

where: $\Delta\mathbf{G} = \nabla \mathrm{L}(\boldsymbol{w}^{(t+1)}_{k'}) - \nabla \mathrm{L}(\boldsymbol{w}^{(t)}_{k'})$

$\Delta\mathbf{W} = w^{(t+1)}_{k'} - w^{(t)}_{k'}$

6 线性链条件随机场实验

线性链条件随机场的实现

第 1 个版本：一次性生成与载入全部数据及其特征，运行较慢，对空间要求较高，不推荐运行；直接运行第 2 个版本（优化版本）。建议对 2 个实现版本进行比较研究，找出关键优化点。注：使用完整数据集，第 1 个版本的运行时间约为 30 小时，第 2 个版本的运行时间约为 23 小时。

```
In [1]: import numpy as np
        from collections import Counter
```

数据集的特征提取类

```
In [2]: class FeatureSet:
            def __init__(self):
                self.label_ids = {'*':0} # 存放标签及其 ID。缺省：开始标签及其 ID
                # 存放子特征字符串、前 1 个标签 ID-当前标签 ID、子特征 ID，
                # 形成字典的字典结构。后面 2 个形成内部字典的 key-value
                self.subfeature_yy_ids = {}
                # 对子特征的出现次数计数，key：子特征 ID；value：出现次数
                self.subfeature_counters = Counter()
                self.scale_threshold = 1e250 # 用于计算结果溢出处理

            # 读取 CONLL 格式的语料文件
            def ReadCorpusCONLL(self, filename):
                with open(filename, 'r') as f:
                    lines = f.readlines()
                self.data = [([], [])] # 分别用来存放 TOKEN_POS 序列及其 LABEL
                for line in lines:
                    words = line.strip().split()
                    if words == []: # 遇到空行
                        if self.data[-1] != ([], []):
                            self.data.append(([], []))
                    else:
                        self.data[-1][0].append(words[:-1]) # 添加 TOKEN 和 POS
                        self.data[-1][1].append(words[-1]) # 添加 LABEL
                if self.data[-1] == ([], []):
                    del(self.data[-1])

            # 为句子生成子特征集，这是缺省的特征函数
            #sentence: [[TOKEN1, POS1], [TOKEN2, POS2], ...]
            def Featurize(self, sentence):
                sentence_subfeatures, T = [], len(sentence)
                for t in range(T):
                    token_features = []
                    # 第 t 步 TOKEN
                    token_features.append('U[0]:%s' % sentence[t][0])
                    # 第 t 步 POS(Part-Of-Speech)
                    token_features.append('POS_U[0]:%s' % sentence[t][1])
                    if t < T-1:
                        # 第 t+1 步 TOKEN
```

```
                token_features.append('U[+1]:%s' % (sentence[t+1][0]))
                token_features.append('B[0]:%s %s' % (sentence[t][0],
                 sentence[t+1][0])) # 第 t 和 t+1 步 TOKEN
                # 第 t+1 步 POS
                token_features.append('POS_U[1]:%s' % sentence[t+1][1])
                token_features.append('POS_B[0]:%s %s' % (sentence[t][1],
                 sentence[t+1][1])) # 第 t 和 t+1 步 POS
                if t < T-2:
                    token_features.append('U[+2]:%s' %
                     (sentence[t+2][0])) # 第 t+2 步 TOKEN
                    token_features.append('POS_U[+2]:%s' %
                     (sentence[t+2][1])) # 第 t+2 步 POS
                    # 第 t+1 和 t+2 步 POS
                    token_features.append('POS_B[+1]:%s %s' %
                     (sentence[t+1][1], sentence[t+2][1]))
                    # 第 t、t+1 和 t+2 步 POS
                    token_features.append('POS_T[0]:%s %s %s' %
                     (sentence[t][1], sentence[t+1][1], sentence[t+2][1]))
            if t > 0:
                # 第 t-1 步 TOKEN
                token_features.append('U[-1]:%s' % (sentence[t-1][0]))
                token_features.append('B[-1]:%s %s' % (sentence[t-1][0],
                 sentence[t][0])) # 第 t-1 和 t 步 TOKEN
                # 第 t-1 步 POS
                token_features.append('POS_U[-1]:%s' % (sentence[t-1][1]))
                token_features.append('POS_B[-1]:%s %s' % (sentence[t-1][1],
                 sentence[t][1])) # 第 t-1 和 t 步 POS
                if t < T-1:
                    # 第 t-1、t 和 t+1 步 POS
                    token_features.append('POS_T[-1]:%s %s %s' %
                     (sentence[t-1][1], sentence[t][1], sentence[t+1][1]))
                if t > 1:
                    # 第 t-2 步 TOKEN
                    token_features.append('U[-2]:%s' % (sentence[t-2][0]))
                    token_features.append('POS_U[-2]:%s' %
                     (sentence[t-2][1])) # 第 t-2 步 POS
                    # 第 t-2 和 t-1 步 POS
                    token_features.append('POS_B[-2]:%s %s' %
                     (sentence[t-2][1], sentence[t-1][1]))
                    # 第 t-2、t-1 和 t 步 POS
                    token_features.append('POS_T[-2]:%s %s %s' %
                     (sentence[t-2][1], sentence[t-1][1], sentence[t][1]))
            sentence_subfeatures.append(token_features)
        return sentence_subfeatures

    def Add(self, y_prev, y, subfeatures):
        for subfeature in subfeatures: # 遍历每个子特征
            if subfeature in self.subfeature_yy_ids.keys(): # 子特征，已存在
```

```
                # 标签对，已存在
                if (y_prev, y) in self.subfeature_yy_ids[subfeature].keys():
                    self.subfeature_counters[self.subfeature_yy_ids
                     [subfeature][(y_prev, y)]] += 1
                else: # 标签对，不存在，新增
                    new_id = len(self.subfeature_counters)
                    self.subfeature_yy_ids[subfeature][(y_prev, y)] = new_id
                    self.subfeature_counters[new_id] = 1
                #(-1, y) 表示当前标签 (y 的当前状态)
                if (-1, y) in self.subfeature_yy_ids[subfeature].keys():
                    self.subfeature_counters[self.subfeature_yy_ids
                     [subfeature][(-1, y)]] += 1
                else: # 不存在，新增
                    new_id = len(self.subfeature_counters)
                    self.subfeature_yy_ids[subfeature][(-1, y)] = new_id
                    self.subfeature_counters[new_id] = 1
            else: # 子特征，不存在
                self.subfeature_yy_ids[subfeature] = {}
                #Bigram feature
                new_id = len(self.subfeature_counters)
                self.subfeature_yy_ids[subfeature][(y_prev, y)] = new_id
                self.subfeature_counters[new_id] = 1
                # Unigram feature
                new_id = len(self.subfeature_counters)
                self.subfeature_yy_ids[subfeature][(-1, y)] = new_id
                self.subfeature_counters[new_id] = 1

    # 为语料数据生成所有的子特征
    def GenerateAllSubfeatures(self):
        self.sentences_subfeatures = []
        for token_poses, labels in self.data: # 遍历每条句子（语料数据）
            sentence_subfeatures = self.Featurize(token_poses)
            self.sentences_subfeatures.append(sentence_subfeatures)
            y_prev = 0 # 开始标签'*' 的 ID 号
            for t in range(len(token_poses)): # 遍历每个单词（语料数据）
                try:
                    y = self.label_ids[labels[t]]
                except KeyError: # 当前标签及其 ID 号不存在，新增 1 个
                    y = len(self.label_ids) # 新 ID 号
                    self.label_ids[labels[t]] = y # 新增标签及其 ID 号
                self.Add(y_prev, y, sentence_subfeatures[t]) # 统计子特征数据
                y_prev = y
        self.weights = np.zeros(len(self.subfeature_counters))
        # 数组：保存每个子特征的出现次数
        self.empirical_counts = np.zeros(len(self.subfeature_counters))
        for id, counts in self.subfeature_counters.items():
            self.empirical_counts[id] = counts
        # 按句子存放 (y_prev, y) 到 ids 的映射关系
```

```
        self.yy_ids = []
        # 遍历每个句子
        for sentence_subfeatures in self.sentences_subfeatures:
            sentence_yy_ids = []
            for t in range(len(sentence_subfeatures)): # 遍历每个 TOKEN
                token_yy_ids = {}
                # 遍历 TOKEN 的每个子特征
                for subfeature in sentence_subfeatures[t]:
                    for (y_prev, y), id in self.subfeature_yy_ids
                     [subfeature].items():
                        if (y_prev, y) in token_yy_ids.keys():
                            token_yy_ids[(y_prev, y)].add(id)
                        else:
                            token_yy_ids[(y_prev, y)] = {id}
                sentence_yy_ids.append([((y_prev, y), ids)
                 for (y_prev, y), ids in token_yy_ids.items()])
            self.yy_ids.append(sentence_yy_ids)
        # 生成 id-labels 数组, 用于路径回溯
        self.id_labels = ['?']*len(self.label_ids) # 初始化
        for label, id in self.label_ids.items():
            self.id_labels[id] = label

    # 计算给定句子的权值-特征的点积 (在 K 个特征函数上求和)
    def CalcWeightDotFeature(self, subfeature_yy_ids, label_ids,
     sentence_subfeatures, weights):
        T, M = len(sentence_subfeatures), len(label_ids)
        table = np.zeros((T, M, M))
        for t in range(T): # 遍历每个 TOKEN-POS
            # 遍历每个 TOKEN-POS 的子特征
            for subfeature in sentence_subfeatures[t]:
                try:
                    for (y_prev, y), id in subfeature_yy_ids
                     [subfeature].items():
                        if y_prev != -1: #Bigram feature
                            table[t, y_prev, y] = table[t, y_prev, y]
                             + weights[id]
                        else: #Unigram feature
                            table[t, :, y] = table[t, :, y] + weights[id]
                # 在使用测试数据集进行测试时, 可能遇到不存在的子特征
                except KeyError:
                    pass
            table[t] = np.exp(table[t]) # 权值-特征的势函数
            # 清除不存在的特征
            if t == 0:
                table[t, 1:] = 0
            else:
                table[t, :, 0] = 0
                table[t, 0, :] = 0
```

```python
        return table

    # 为数据集生成所有的 M 矩阵
    def GeneratePotentialMatrix(self, weights):
        self.Ms = []
        # 遍历每个句子
        for sentence_subfeatures in self.sentences_subfeatures:
            self.Ms.append(self.CalcWeightDotFeature(
             self.subfeature_yy_ids, self.label_ids,
             sentence_subfeatures, weights))

    # 计算给定句子的 alpha、beta 与 Z
    def CalcForwardBackward(self, i_sentence, sentence_subfeatures):
        T, M = len(sentence_subfeatures), len(self.label_ids)
        #scales 用于计算结果溢出处理
        alphas, betas, scales = np.zeros((T, M)), np.zeros((T, M)), {}
        alphas[0] = self.Ms[i_sentence][0][0, :] # 初始化 alpha 向量
        t = 1
        while t < T: # 递推计算前向概率
            overflow = False
            for id in range(1, M):
                alphas[t, id] = np.dot(alphas[t-1, :], self.Ms[i_sentence]
                 [t][:, id])
                if alphas[t, id] > self.scale_threshold: # 结果溢出
                    overflow = True
                    scales[t-1] = self.scale_threshold
                    break
            if overflow: # 溢出，计算结果的处理
                alphas[t-1], alphas[t] = alphas[t-1] / self.scale_threshold, 0
            else:
                t += 1
        betas[T-1] = 1.0 # 初始化 beta 向量
        for t in range(T-2, -1, -1): # 递推计算后向概率
            betas[t] = np.dot(betas[t+1].reshape(1, -1), self.Ms[i_sentence]
             [t+1].T)
            if t in scales.keys():
                betas[t] = betas[t] / scales[t]
        return alphas, betas, sum(alphas[T-1]), scales

    # 为数据集生成所有的 alpha、beta 向量与 Z
    def GenerateForwardBackward(self):
        self.alphas, self.betas, self.Zs, self.scales = [], [], [], []
        for i_sentence, sentence_subfeatures in enumerate(
         self.sentences_subfeatures): # 遍历每个句子
            alphas, betas, Z, scales = self.CalcForwardBackward(i_sentence,
             sentence_subfeatures)
            self.alphas.append(alphas)
            self.betas.append(betas)
```

```
                self.Zs.append(Z)
                self.scales.append(scales)
```

定义 scipy.optimize 的 L-BFGS 函数所需要的回调函数

```
In [2]: from math import log
```

```
In [4]: # 计算数据集的损失函数（负对数似然函数）
        def LossFunction(weights, *args):
            feature_set, squared_sigma, step = args # 对象、正则化参数
            step[0] += 1
            feature_set.GeneratePotentialMatrix(weights) # 使用新权值，更新 M 矩阵
            # 使用新权值，更新 alpha、beta 和 Z
            feature_set.GenerateForwardBackward()
            logZ_sum = sum([log(Z) for Z in feature_set.Zs]) + sum([log(scale)
             for scales in feature_set.scales for t, scale in scales.items()])

            # 数据集的对数似然函数 logP(y1|x1)+..., 每个单样本的对数似然函数为:
            # sum(w*f)-logZ, 所有样本的对数似然函数为: sum(sum(w*f))-logZ_sum
            # 最后一项是正则化项
            loglikelihood = np.dot(feature_set.empirical_counts, weights) -
             logZ_sum - np.dot(weights, weights)/(squared_sigma*2)
            loss = -loglikelihood
            # 计算梯度（其中，logZ_sum 对 weights 的梯度，都归结为此项。
            # 先考虑单样本情形，logZ 求偏导 =>F(有效)*P(y|x)，即下面的 prob 项；
            # 再考虑多样本情形即可）
            expected_counts = np.zeros(len(feature_set.subfeature_counters))
            # 每个句子
            for i_sentence, sentence_yy_ids in enumerate(feature_set.yy_ids):
                # 每个 TOKEN
                for i_token, token_yy_ids in enumerate(sentence_yy_ids):
                    # 在相应的特征（有效）处，+p(prev,y|X,t)
                    for (y_prev, y), ids in token_yy_ids:
                        if y_prev == -1: # 单状态
                            # 结果溢出过
                            if i_token in feature_set.scales[i_sentence].keys():
                                prob = (feature_set.alphas[i_sentence][i_token, y] *
                                 feature_set.betas[i_sentence][i_token, y]) *
                                 feature_set.scales[i_sentence][i_token] /
                                 feature_set.Zs[i_sentence]
                            else:
                                prob = (feature_set.alphas[i_sentence][i_token, y] *
                                 feature_set.betas[i_sentence][i_token, y]) /
                                 feature_set.Zs[i_sentence]
                        elif i_token == 0: # 初始状态
                            if y_prev != 0: # 需要判断，否则出现 overflow
                                continue
                            prob = feature_set.Ms[i_sentence][i_token][0, y] *
```

```
                        feature_set.betas[i_sentence][i_token, y] /
                        feature_set.Zs[i_sentence]
                else: # 双状态
                    if y_prev == 0 or y == 0: # 需要判断，否则出现 overflow
                        continue
                    prob = feature_set.alphas[i_sentence][i_token-1, y_prev] *
                        feature_set.Ms[i_sentence][i_token][y_prev, y] *
                        feature_set.betas[i_sentence][i_token, y] /
                        feature_set.Zs[i_sentence]
                for id in ids: # 特征有效处，+prob
                    expected_counts[id] = expected_counts[id] + prob
        gradients = feature_set.empirical_counts - expected_counts -
         weights/squared_sigma
        gradients = -gradients
        print('Step', step[0], 'Loss:', loss)
        return loss, gradients
```

In [3]:
```
import os
import pickle
from scipy.optimize import fmin_l_bfgs_b
import time
import datetime
```

线性链条件随机场类

In [6]:
```
class LinearChainCRF:
    def __init__(self, filename):
        self.filename = filename
        self.squared_sigma = 10.0 # 正则化参数
        self.feature_set = FeatureSet()

    def fit(self):
        print('Reading corpus data ...')
        self.feature_set.ReadCorpusCONLL(self.filename)
        self.feature_set.GenerateAllSubfeatures()
        print('Data reading completed!')

        start_time = time.time()
        print('[%s] Starting to train ...' % datetime.datetime.now())
        self.feature_set.weights, self.loss, self.info = fmin_l_bfgs_b(
         func = LossFunction, x0 = self.feature_set.weights, args = (
         self.feature_set, self.squared_sigma, [0]))
        elapsed_time = time.time() - start_time
        print('* Elapsed time: %f' % elapsed_time)
        print('* [%s] CRF Training done' % datetime.datetime.now())

        self.SaveModel()
```

```
def predict(self, filename = None):
    if filename == None:
        filename = self.filename
    test_feature_set = FeatureSet()
    test_feature_set.ReadCorpusCONLL(filename) # 读取语料测试数据集
    totals = corrects = 0
    print('Starting to predict ...')
    # 遍历测试数据集中所有的句子
    for token_poses, labels in test_feature_set.data:
        labels_predict = self.inference(test_feature_set, token_poses)
        for t, label in enumerate(labels):
            totals += 1
            if label == labels_predict[t]:
                corrects += 1
    print('Corrects: %d' % corrects)
    print('Totals: %d' % totals)
    print('Performance: %f' % (corrects/totals))

def inference(self, test_feature_set, token_poses):
    sentence_subfeatures = test_feature_set.Featurize(token_poses)
    Mtable = test_feature_set.CalcWeightDotFeature(self.feature_set.
     subfeature_yy_ids, self.feature_set.label_ids, sentence_subfeatures,
     self.feature_set.weights)
    return self.Viterbi(sentence_subfeatures, Mtable)

def Viterbi(self, sentence_subfeatures, Mtable):
    T, M = Mtable.shape[0], Mtable.shape[1]
    max_table, argmax_table = np.zeros((T, M)), np.zeros((T, M),
     dtype='int64')
    max_table[0] = Mtable[0][0] # 初始化
    for t in range(1, T):
        for id in range(1, M):
            max_value, max_id = -float('inf'), None
            for prev_id in range(1, M):
                value = max_table[t-1, prev_id] * Mtable[t][prev_id, id]
                if value > max_value:
                    max_value, max_id = value, prev_id
            max_table[t, id], argmax_table[t, id] = max_value, max_id
    path = []
    next_id = max_table[T-1].argmax()
    path.append(next_id)
    for t in range(T-1, -1, -1):
        next_id = argmax_table[t, next_id]
        path.append(next_id)
    return [self.feature_set.id_labels[id] for id in path[::-1][1:]]

def SaveModel(self, filename = None):
    if filename == None:
```

```
                filename = os.path.splitext(self.filename)[0] + '.pkl'
            print('* Writing data into file "%s/%s"...' % (os.getcwd(), filename))
            with open(filename, 'wb') as f:
                str = pickle.dumps(self.feature_set)
                f.write(str)
            print('* Trained CRF Model has been saved at "%s/%s"' % (os.getcwd(),
             filename))

        def LoadModel(self, filename = None):
            if filename == None:
                filename = os.path.splitext(self.filename)[0] + '.pkl'
            print('* Loading file "%s/%s" ...' % (os.getcwd(), filename))
            self.feature_set = FeatureSet()
            with open(filename, 'rb') as f:
                self.feature_set = pickle.loads(f.read())
            print('* Trained CRF Model has been loaded at "%s/%s"' % (os.getcwd(),
             filename))
```

使用较小的训练数据集进行训练

```
In [8]: crf = LinearChainCRF('small_train.data')
        crf.fit()
```

```
Reading corpus data ...
Data reading completed!
[2020-04-07 20:50:51.180575] Starting to train ...
Step 1 Loss: 5003.65269695
Step 2 Loss: 4060.00809341
Step 3 Loss: 1943.63442154
Step 4 Loss: 1102.12231231
Step 5 Loss: 497.61166689
Step 6 Loss: 181.51651819
Step 7 Loss: 87.9987527507
Step 8 Loss: 44.3637129976
Step 9 Loss: 41.0589210027
Step 10 Loss: 35.4069726632
Step 11 Loss: 33.3593721226
Step 12 Loss: 32.1185389828
Step 13 Loss: 31.4017352761
Step 14 Loss: 30.7156707716
Step 15 Loss: 30.0662425865
Step 16 Loss: 29.5865062084
Step 17 Loss: 29.3233355665
Step 18 Loss: 29.2010378168
Step 19 Loss: 29.1035012415
Step 20 Loss: 29.0717114227
Step 21 Loss: 29.0298952745
Step 22 Loss: 29.0183871421
```

```
Step 23 Loss: 29.0004827719
Step 24 Loss: 28.9839983685
Step 25 Loss: 28.9888756275
Step 26 Loss: 28.9778307479
Step 27 Loss: 28.971453893
Step 28 Loss: 28.9694121189
Step 29 Loss: 28.9679331712
Step 30 Loss: 28.9670256181
Step 31 Loss: 28.966562413
Step 32 Loss: 28.9661968718
Step 33 Loss: 28.965935806
Step 34 Loss: 28.965880818
Step 35 Loss: 28.965656917
Step 36 Loss: 28.965626421
Step 37 Loss: 28.9655597551
Step 38 Loss: 28.9655435663
Step 39 Loss: 28.9655150935
Step 40 Loss: 28.9655107143
Step 41 Loss: 28.9655044219
Step 42 Loss: 28.9654997819
Step 43 Loss: 28.9655047043
Step 44 Loss: 28.9654989913
Step 45 Loss: 28.9654980378
Step 46 Loss: 28.9654976245
Step 47 Loss: 28.9654970586
Step 48 Loss: 28.9654969924
Step 49 Loss: 28.9654967564
Step 50 Loss: 28.9654966976
* Elapsed time: 40.976157
* [2020-04-07 20:51:32.156733] CRF Training done
* Writing data into file "E:\dxq\2019-2020-2\机器学习\第 11 讲
  \实现 1/small_train.pkl"...
* Trained CRF Model has been saved at "E:\dxq\2019-2020-2\机器学习\第 11 讲
  \实现 1/small_train.pkl"
```

载入训练好的 CRF 模型，并在较小的测试数据集上进行测试

```
In [9]: test_crf = LinearChainCRF('small_test.data')
        test_crf.LoadModel('small_train.pkl')
        test_crf.predict()
```

```
* Loading file "E:\dxq\2019-2020-2\机器学习\第 11 讲\实现 1/small_train.pkl" ...
* Trained CRF Model has been loaded at "E:\dxq\2019-2020-2\机器学习\第 11 讲
  \实现 1/small_train.pkl"
Starting to predict ...
Corrects: 17237
Totals: 19172
```

```
Performance: 0.899072
```

使用完整的训练数据集进行训练

```
In [10]: crf = LinearChainCRF('full_train.data')
         crf.fit()
```

```
Reading corpus data ...
Data reading completed!
[2020-04-07 20:55:07.367372] Starting to train ...
Step 1 Loss: 654457.59004
Step 2 Loss: 550493.756926
Step 3 Loss: 268138.843217
Step 4 Loss: 148258.862569
Step 5 Loss: 98261.5130191
Step 6 Loss: 70158.0689031
Step 7 Loss: 56750.0350409
Step 8 Loss: 50207.4763284
Step 9 Loss: 46446.2009321
Step 10 Loss: 40053.1127408
Step 11 Loss: 34981.6296758
Step 12 Loss: 31672.0202771
Step 13 Loss: 28950.4652865
Step 14 Loss: 25666.271324
Step 15 Loss: 23255.9300538
Step 16 Loss: 20988.6652898
Step 17 Loss: 20052.9957806
Step 18 Loss: 18371.9685515
Step 19 Loss: 17192.5098554
Step 20 Loss: 15979.9171822
Step 21 Loss: 14891.2157996
Step 22 Loss: 13490.7295361
Step 23 Loss: 11778.6554907
Step 24 Loss: 11151.3112403
Step 25 Loss: 9364.72350401
Step 26 Loss: 8765.71846471
Step 27 Loss: 7648.21958947
Step 28 Loss: 7509.37421188
Step 29 Loss: 6997.85325813
Step 30 Loss: 6198.22809484
Step 31 Loss: 5395.00537908
Step 32 Loss: 4848.63787787
Step 33 Loss: 4251.82890234
Step 34 Loss: 4003.79487536
Step 35 Loss: 3462.38566839
Step 36 Loss: 3190.23593853
Step 37 Loss: 2843.75722277
```

```
Step 38 Loss: 2623.71359313
Step 39 Loss: 2393.12949594
Step 40 Loss: 2151.93235835
Step 41 Loss: 2091.62614406
Step 42 Loss: 1854.56116605
Step 43 Loss: 1805.81852041
Step 44 Loss: 1719.60868645
Step 45 Loss: 1648.62455427
Step 46 Loss: 1769.43841314
Step 47 Loss: 1606.75544176
Step 48 Loss: 1556.10512309
Step 49 Loss: 1521.57297842
Step 50 Loss: 1476.11633754
Step 51 Loss: 1451.02822773
Step 52 Loss: 1416.60031377
Step 53 Loss: 1398.38504995
Step 54 Loss: 1377.47346506
Step 55 Loss: 1346.31441898
Step 56 Loss: 1352.33694265
Step 57 Loss: 1326.61582815
Step 58 Loss: 1297.11750305
Step 59 Loss: 1279.81316449
Step 60 Loss: 1264.41499895
Step 61 Loss: 1246.37536654
Step 62 Loss: 1232.72960938
Step 63 Loss: 1219.10585415
Step 64 Loss: 1202.77579802
Step 65 Loss: 1200.78603967
Step 66 Loss: 1188.95329895
Step 67 Loss: 1167.76276574
Step 68 Loss: 1152.77181903
Step 69 Loss: 1134.62204706
Step 70 Loss: 1122.47703258
Step 71 Loss: 1109.86132469
Step 72 Loss: 1100.03030962
Step 73 Loss: 1088.98102193
Step 74 Loss: 1093.17014005
Step 75 Loss: 1083.09526142
Step 76 Loss: 1073.02496793
Step 77 Loss: 1064.79312611
Step 78 Loss: 1056.91849581
Step 79 Loss: 1047.95812109
Step 80 Loss: 1039.66143578
Step 81 Loss: 1031.80698306
Step 82 Loss: 1025.91038542
Step 83 Loss: 1020.40086951
Step 84 Loss: 1016.022125
Step 85 Loss: 1009.82116692
```

```
Step 86 Loss: 1006.54307199
Step 87 Loss: 1003.38185824
Step 88 Loss: 1008.17345299
Step 89 Loss: 1001.5100916
Step 90 Loss: 998.858363115
Step 91 Loss: 993.968088145
Step 92 Loss: 991.412879233
Step 93 Loss: 987.07443502
Step 94 Loss: 988.035056506
Step 95 Loss: 984.495712591
Step 96 Loss: 981.786850377
Step 97 Loss: 979.458756292
Step 98 Loss: 977.02518632
Step 99 Loss: 975.280231952
Step 100 Loss: 972.501148371
Step 101 Loss: 971.236557417
Step 102 Loss: 969.313350033
Step 103 Loss: 969.310190123
Step 104 Loss: 968.214969275
Step 105 Loss: 966.489181718
Step 106 Loss: 965.268289304
Step 107 Loss: 963.390571843
Step 108 Loss: 962.711464211
Step 109 Loss: 960.740793857
Step 110 Loss: 959.886239368
Step 111 Loss: 958.830834947
Step 112 Loss: 957.963274829
Step 113 Loss: 956.757058247
Step 114 Loss: 956.004950972
Step 115 Loss: 955.179912557
Step 116 Loss: 953.888924214
Step 117 Loss: 954.306302457
Step 118 Loss: 953.241583703
Step 119 Loss: 952.240902964
Step 120 Loss: 951.791728325
Step 121 Loss: 951.409241719
Step 122 Loss: 950.823258107
Step 123 Loss: 950.483734085
Step 124 Loss: 949.847721846
Step 125 Loss: 949.469548019
Step 126 Loss: 949.162546515
Step 127 Loss: 948.590196048
Step 128 Loss: 949.053525252
Step 129 Loss: 948.293412731
Step 130 Loss: 947.955220491
Step 131 Loss: 947.720803276
Step 132 Loss: 947.381212814
Step 133 Loss: 946.982010914
```

```
Step 134 Loss: 946.689706
Step 135 Loss: 946.506954167
Step 136 Loss: 946.361344674
Step 137 Loss: 946.248646977
Step 138 Loss: 946.036070974
Step 139 Loss: 945.823828519
Step 140 Loss: 945.618047423
Step 141 Loss: 945.485509994
Step 142 Loss: 945.345078472
Step 143 Loss: 945.215903161
Step 144 Loss: 945.092491858
Step 145 Loss: 944.949487946
Step 146 Loss: 944.846449555
Step 147 Loss: 944.754174573
Step 148 Loss: 944.666090005
Step 149 Loss: 944.574472638
Step 150 Loss: 944.47195361
Step 151 Loss: 944.391280462
Step 152 Loss: 944.309911501
Step 153 Loss: 944.23648528
Step 154 Loss: 944.172333024
Step 155 Loss: 944.091450249
Step 156 Loss: 944.048424438
Step 157 Loss: 943.99354929
Step 158 Loss: 943.932320952
Step 159 Loss: 943.899677352
Step 160 Loss: 943.837230105
Step 161 Loss: 943.832162256
Step 162 Loss: 943.801227963
Step 163 Loss: 943.771752195
Step 164 Loss: 943.739049791
Step 165 Loss: 943.715235818
Step 166 Loss: 943.656948052
Step 167 Loss: 943.665021154
Step 168 Loss: 943.631353841
Step 169 Loss: 943.604257022
Step 170 Loss: 943.573927137
Step 171 Loss: 943.550876787
Step 172 Loss: 943.523091231
Step 173 Loss: 943.49305442
Step 174 Loss: 943.478404921
Step 175 Loss: 943.46066862
Step 176 Loss: 943.444275754
Step 177 Loss: 943.423529249
Step 178 Loss: 943.409784915
Step 179 Loss: 943.388342964
Step 180 Loss: 943.362710742
Step 181 Loss: 943.391717154
```

```
Step 182 Loss: 943.35182495
Step 183 Loss: 943.33412757
Step 184 Loss: 943.321839548
Step 185 Loss: 943.310131764
Step 186 Loss: 943.299517569
Step 187 Loss: 943.290524555
Step 188 Loss: 943.282351408
Step 189 Loss: 943.276733902
Step 190 Loss: 943.265749227
Step 191 Loss: 943.259468669
Step 192 Loss: 943.254336274
Step 193 Loss: 943.244795011
Step 194 Loss: 943.23707733
Step 195 Loss: 943.224876457
Step 196 Loss: 943.22111137
Step 197 Loss: 943.214711767
Step 198 Loss: 943.214026255
Step 199 Loss: 943.205840096
Step 200 Loss: 943.202509011
Step 201 Loss: 943.199039453
Step 202 Loss: 943.196487178
Step 203 Loss: 943.193020571
Step 204 Loss: 943.189358163
Step 205 Loss: 943.183408966
Step 206 Loss: 943.187853638
Step 207 Loss: 943.181368214
Step 208 Loss: 943.178636849
Step 209 Loss: 943.175156558
Step 210 Loss: 943.17318762
Step 211 Loss: 943.170544892
Step 212 Loss: 943.16840718
Step 213 Loss: 943.16621863
Step 214 Loss: 943.165135947
Step 215 Loss: 943.162595805
Step 216 Loss: 943.161689682
Step 217 Loss: 943.16015122
Step 218 Loss: 943.158009758
Step 219 Loss: 943.156164588
Step 220 Loss: 943.154809353
Step 221 Loss: 943.153267848
Step 222 Loss: 943.15196846
Step 223 Loss: 943.15063696
Step 224 Loss: 943.149134656
Step 225 Loss: 943.147918652
Step 226 Loss: 943.147013046
Step 227 Loss: 943.146006746
Step 228 Loss: 943.145287558
Step 229 Loss: 943.144828497
```

```
Step 230 Loss: 943.144252571
Step 231 Loss: 943.143711502
Step 232 Loss: 943.142942111
Step 233 Loss: 943.144320645
Step 234 Loss: 943.142439843
Step 235 Loss: 943.141466969
Step 236 Loss: 943.140840077
Step 237 Loss: 943.140253925
Step 238 Loss: 943.139885141
Step 239 Loss: 943.139485999
Step 240 Loss: 943.138811875
Step 241 Loss: 943.138425752
Step 242 Loss: 943.138013828
Step 243 Loss: 943.138375307
Step 244 Loss: 943.13780397
Step 245 Loss: 943.137513945
Step 246 Loss: 943.137156374
Step 247 Loss: 943.136838323
Step 248 Loss: 943.136306832
Step 249 Loss: 943.136353326
Step 250 Loss: 943.136006538
Step 251 Loss: 943.135794152
Step 252 Loss: 943.135618133
Step 253 Loss: 943.135413025
Step 254 Loss: 943.135149715
Step 255 Loss: 943.134883884
Step 256 Loss: 943.134733664
Step 257 Loss: 943.134599801
Step 258 Loss: 943.134500716
Step 259 Loss: 943.134361391
Step 260 Loss: 943.134237911
Step 261 Loss: 943.13417914
Step 262 Loss: 943.133960555
Step 263 Loss: 943.134326772
Step 264 Loss: 943.13384815
Step 265 Loss: 943.133719908
Step 266 Loss: 943.133635734
Step 267 Loss: 943.13354788
Step 268 Loss: 943.133397805
Step 269 Loss: 943.133189583
Step 270 Loss: 943.13360936
Step 271 Loss: 943.133110387
Step 272 Loss: 943.133018793
Step 273 Loss: 943.132958421
Step 274 Loss: 943.132904645
Step 275 Loss: 943.132849954
Step 276 Loss: 943.132792179
Step 277 Loss: 943.132738469
```

```
Step 278 Loss: 943.132701766
Step 279 Loss: 943.132678635
Step 280 Loss: 943.132594899
Step 281 Loss: 943.132571355
Step 282 Loss: 943.132546962
Step 283 Loss: 943.132511117
Step 284 Loss: 943.132482203
Step 285 Loss: 943.132424549
Step 286 Loss: 943.13240681
Step 287 Loss: 943.13238511
Step 288 Loss: 943.132356112
Step 289 Loss: 943.132320204
Step 290 Loss: 943.132294364
Step 291 Loss: 943.13227813
Step 292 Loss: 943.132255737
Step 293 Loss: 943.132240292
Step 294 Loss: 943.132220777
Step 295 Loss: 943.132204055
Step 296 Loss: 943.132195171
Step 297 Loss: 943.132177453
Step 298 Loss: 943.132164674
Step 299 Loss: 943.13214688
Step 300 Loss: 943.132138052
Step 301 Loss: 943.132129188
Step 302 Loss: 943.132116056
Step 303 Loss: 943.132101491
Step 304 Loss: 943.132082594
Step 305 Loss: 943.132076067
Step 306 Loss: 943.132070772
Step 307 Loss: 943.132064586
Step 308 Loss: 943.132058975
Step 309 Loss: 943.132044932
Step 310 Loss: 943.132066106
Step 311 Loss: 943.132041134
Step 312 Loss: 943.132035284
Step 313 Loss: 943.132027162
Step 314 Loss: 943.132024071
Step 315 Loss: 943.132018309
Step 316 Loss: 943.132015287
Step 317 Loss: 943.132010779
Step 318 Loss: 943.132005391
Step 319 Loss: 943.132010099
Step 320 Loss: 943.132002416
Step 321 Loss: 943.131998826
Step 322 Loss: 943.131997026
* Elapsed time: 111151.960003
* [2020-04-09 03:47:39.321367] CRF Training done
* Writing data into file "E:\dxq\2019-2020-2\机器学习\第 11 讲\实现 1
```

```
  /full_train.pkl"...
* Trained CRF Model has been saved at "E:\dxq\2019-2020-2\机器学习\第 11 讲
  \实现 1/full_train.pkl"
```

载入训练好的 CRF 模型，并在完整的测试数据集上进行测试

```
In [7]: test_crf = LinearChainCRF('full_test.data')
        test_crf.LoadModel('full_train.pkl')
        test_crf.predict()
```

```
* Loading file "E:\dxq\2019-2020-2\机器学习\第 11 讲\实现 1/full_train.pkl" ...
* Trained CRF Model has been loaded at "E:\dxq\2019-2020-2\机器学习\第 11 讲
  \实现 1/full_train.pkl"
Starting to predict ...
Corrects: 45488
Totals: 47377
Performance: 0.960128
```

第 2 个版本（优化）
数据集的特征提取类

```
In [4]: class FeatureSet:
            def __init__(self):
                # 字典的字典：存放子特征字符串、前 1 个标签 ID-当前标签 ID、子特征 ID
                self.feature_dict = {}
                # 对子特征的出现次数计数，key: 子特征 ID; value: 出现次数
                self.empirical_dict = Counter()
                self.num_features = 0 # 子特征个数
                self.squared_sigma = 10.0 # 正则化参数
                self.scale_threshold = 1e250 # 用于计算结果溢出处理

                self.label_dict = {'*': 0} # 开始符号及 ID
                self.label_array = ['*']

            # 读取 CONLL 格式的语料文件
            def ReadCorpusCONLL(self, filename):
                with open(filename, 'r') as f:
                    lines = f.readlines()
                self.data = [([], [])] # 分别用来存放 TOKEN_POS 序列及其 LABEL
                for line in lines:
                    words = line.strip().split()
                    if words == []: # 遇到空行
                        if self.data[-1] != ([], []):
                            self.data.append(([], []))
                    else:
                        self.data[-1][0].append(words[:-1]) # 添加 TOKEN 和 POS
                        self.data[-1][1].append(words[-1]) # 添加 LABEL
```

```
        if self.data[-1] == ([], []):
            del(self.data[-1])

    # 为句子生成子特征集，这是缺省的特征函数
    def Featurize(self, X, t):
        length = len(X)

        features = []
        features.append('U[0]:%s' % X[t][0]) # 第 t 步 TOKEN
        # 第 t 步 POS(Part-Of-Speech)
        features.append('POS_U[0]:%s' % X[t][1])
        if t < length-1:
            features.append('U[+1]:%s' % (X[t+1][0])) # 第 t+1 步 TOKEN
            # 第 t 和 t+1 步 TOKEN
            features.append('B[0]:%s %s' % (X[t][0], X[t+1][0]))
            features.append('POS_U[1]:%s' % X[t+1][1]) # 第 t+1 步 POS
            # 第 t 和 t+1 步 POS
            features.append('POS_B[0]:%s %s' % (X[t][1], X[t+1][1]))
            if t < length-2:
                features.append('U[+2]:%s' % (X[t+2][0])) # 第 t+2 步 TOKEN
                features.append('POS_U[+2]:%s' % (X[t+2][1])) # 第 t+2 步 POS
                # 第 t+1 和 t+2 步 POS
                features.append('POS_B[+1]:%s %s' % (X[t+1][1], X[t+2][1]))
                features.append('POS_T[0]:%s %s %s' % (X[t][1], X[t+1][1],
                 X[t+2][1])) # 第 t、t+1 和 t+2 步 POS
        if t > 0:
            features.append('U[-1]:%s' % (X[t-1][0])) # 第 t-1 步 TOKEN
            # 第 t-1 和 t 步 TOKEN
            features.append('B[-1]:%s %s' % (X[t-1][0], X[t][0]))
            features.append('POS_U[-1]:%s' % (X[t-1][1])) # 第 t-1 步 POS
            # 第 t-1 和 t 步 POS
            features.append('POS_B[-1]:%s %s' % (X[t-1][1], X[t][1]))
            if t < length-1:
                features.append('POS_T[-1]:%s %s %s' % (X[t-1][1],
                 X[t][1], X[t+1][1])) # 第 t-1、t 和 t+1 步 POS
            if t > 1:
                features.append('U[-2]:%s' % (X[t-2][0])) # 第 t-2 步 TOKEN
                features.append('POS_U[-2]:%s' % (X[t-2][1])) # 第 t-2 步 POS
                # 第 t-2 和 t-1 步 POS
                features.append('POS_B[-2]:%s %s' % (X[t-2][1], X[t-1][1]))
                features.append('POS_T[-2]:%s %s %s' % (X[t-2][1],
                 X[t-1][1], X[t][1])) # 第 t-2、t-1 和 t 步 POS

        return features

    # 为语料数据生成所有的子特征及相关数据
    def GenerateAllFeatures(self):
        for X, Y in self.data: # 遍历数据集中的每一条句子
```

```
            prev_y = 0 #START 索引 ID 号
            for t in range(len(X)): # 遍历每个 TOKEN, t 表示序列时间步
                # Gets a label id
                try: # 标签 ID 的处理
                    y = self.label_dict[Y[t]]
                # 当前标签 ID 不在 self.label_dict 中, 新增 1 个
                except KeyError:
                    y = len(self.label_dict)
                    self.label_dict[Y[t]] = y
                    self.label_array.append(Y[t])
                # Adds features
                # 对当前 TOKEN 的特征进行处理, 并统计 Bigram Feature 和
                # Unigram Feature 出现的次数
                self.AddFeature(prev_y, y, X, t)
                prev_y = y #LABEL 索引 ID 号
        self.params = np.zeros(self.num_features)
        self.GenerateEmpiricalCounts()
        self.GenerateAll_YY_IDS()

    def AddFeature(self, prev_y, y, X, t):
        for feature_string in self.Featurize(X, t):
            if feature_string in self.feature_dict.keys():
                # 字典的字典: 前 1 个标签 ID 和当前标签 ID 作为 key, 已存在
                if (prev_y, y) in self.feature_dict[feature_string].keys():
                    self.empirical_dict[self.feature_dict[feature_string]
                     [(prev_y, y)]] += 1 #(prev_y,y) 的 ID 号是唯一的
                else: #(prev_y,y) 的 ID 号不存在, 创建 1 个
                    feature_id = self.num_features
                    self.feature_dict[feature_string][(prev_y, y)] =
                     feature_id # 生成子特征 ID 号
                    self.empirical_dict[feature_id] = 1
                    self.num_features += 1
                if (-1, y) in self.feature_dict[feature_string].keys():
                    self.empirical_dict[self.feature_dict[feature_string]
                     [(-1, y)]] += 1
                else:
                    feature_id = self.num_features
                    self.feature_dict[feature_string][(-1, y)] = feature_id
                    self.empirical_dict[feature_id] = 1
                    self.num_features += 1
            else:
                self.feature_dict[feature_string] = {}
                # Bigram feature
                feature_id = self.num_features
                self.feature_dict[feature_string][(prev_y, y)] = feature_id
                self.empirical_dict[feature_id] = 1
                self.num_features += 1
                # Unigram feature
```

```
                feature_id = self.num_features
                self.feature_dict[feature_string][(-1, y)] = feature_id
                self.empirical_dict[feature_id] = 1
                self.num_features += 1

    # 计算给定句子的权值-特征的点积（在 K 个特征函数上求和），生成 M 矩阵
    def GenerateMtable(self, params, num_labels, X):
        tables = []
        for t in range(len(X)):
            # 每个时间步 t 对应的 M 方阵
            table = np.zeros((num_labels, num_labels))
            for (prev_y, y), feature_ids in X[t]:
                score = sum(params[fid] for fid in feature_ids)
                if prev_y == -1:
                    table[:, y] += score
                else:
                    table[prev_y, y] += score
            table = np.exp(table)
            if t == 0:
                table[1:] = 0
            else:
                table[:, 0] = 0
                table[0, :] = 0

            tables.append(table)

        return tables

    # 计算给定句子的 alpha、beta 与 Z（还包括溢出处理）
    def ForwardBackward(self, num_labels, time_length, potential_table):
        alpha = np.zeros((time_length, num_labels)) #alpha 矩阵
        scaling_dict = {}

        t = 0
        for label_id in range(num_labels): #alpha 前向概率，从 t=0 开始
            alpha[t, label_id] = potential_table[t][0, label_id]
        t = 1
        while t < time_length: # 递推计算前向概率
            scaling_time = None
            scaling_coefficient = None
            overflow_occured = False
            for label_id in range(1, num_labels):
                alpha[t, label_id] = np.dot(alpha[t-1,:],
                 potential_table[t][:,label_id]) # 计算前向概率

                if alpha[t, label_id] > self.scale_threshold:
                    overflow_occured = True
                    scaling_time = t - 1
```

```
                    scaling_coefficient = self.scale_threshold
                    scaling_dict[scaling_time] = scaling_coefficient
                    break

            if overflow_occured:
                alpha[t-1] /= scaling_coefficient
                alpha[t] = 0
            else:
                t += 1

        beta = np.zeros((time_length, num_labels))
        t = time_length - 1
        for label_id in range(num_labels):
            beta[t, label_id] = 1.0
        for t in range(time_length-2, -1, -1):
            for label_id in range(1, num_labels):
                beta[t, label_id] = np.dot(beta[t+1,:],
                 potential_table[t+1][label_id,:])

            if t in scaling_dict.keys():
                beta[t] /= scaling_dict[t]

        Z = sum(alpha[time_length-1])

        return alpha, beta, Z, scaling_dict

    # 数组：子特征的出现次数
    def GenerateEmpiricalCounts(self):
        self.empirical_counts = np.ndarray((self.num_features,))
        for feature_id, counts in self.empirical_dict.items():
            self.empirical_counts[feature_id] = counts

    # 生成给定 TOKEN 的 YY-IDS 对应表
    def GenerateYY_IDS(self, X, t):
        feature_list_dict = {}
        # 遍历特征中的每个子特征（字符串）
        for feature_string in self.Featurize(X, t):
            try:
                for (prev_y, y), feature_id in self.feature_dict
                 [feature_string].items():
                    if (prev_y, y) in feature_list_dict.keys():
                        feature_list_dict[(prev_y, y)].add(feature_id)
                    else:
                        feature_list_dict[(prev_y, y)] = {feature_id}
            # 应用于测试数据集：可能存在训练数据集中没有的特征
            except KeyError:
                pass
        return [((prev_y, y), feature_ids) for (prev_y, y), feature_ids
```

```
         in feature_list_dict.items()]

    # 所有语料数据集的 YY-IDS 对应表
    def GenerateAll_YY_IDS(self):
        self.training_feature_data =  [[self.GenerateYY_IDS(X, t)
         for t in range(len(X))] for X, Y in self.data]
```

定义 scipy.optimize 的 L-BFGS 函数所需要的回调函数

```
In [5]: # 计算数据集的损失函数（负对数似然函数）
        def Loss(params, *args):
            feature_set, step = args
            step[0] += 1
            expected_counts = np.zeros(feature_set.num_features)

            total_logZ = 0
            for X_features in feature_set.training_feature_data:
                potential_table = feature_set.GenerateMtable(params,
                 len(feature_set.label_dict), X_features)
                alpha, beta, Z, scaling_dict = feature_set.ForwardBackward(
                 len(feature_set.label_dict), len(X_features), potential_table)
                # 计算 log(Z1(x)*Z2(x)*...ZT(x)), T 表示序列长度
                total_logZ += log(Z) + sum(log(scaling_coefficient)
                 for _, scaling_coefficient in scaling_dict.items())
                for t in range(len(X_features)):
                    potential = potential_table[t]
                    # Adds p(prev_y, y | X, t)
                    for (prev_y, y), feature_ids in X_features[t]:
                        if prev_y == -1:
                            if t in scaling_dict.keys():
                                prob = (alpha[t, y] * beta[t, y] * scaling_dict[t])/Z
                            else:
                                prob = (alpha[t, y] * beta[t, y])/Z
                        elif t == 0:
                            if prev_y != 0:
                                continue
                            else:
                                prob = (potential[0, y] * beta[t, y])/Z
                        else:
                            if prev_y == 0 or y == 0:
                                continue
                            else:
                                prob = (alpha[t-1, prev_y] * potential[prev_y, y] *
                                 beta[t, y]) / Z
                        for fid in feature_ids:
                            expected_counts[fid] += prob

            # 数据集的对数似然函数 logP(y1|x1)+..., 每个单样本的对数似然函数为:
```

```
        # sum(w*f)-logZ, 所有样本的对数似然函数为: sum(sum(w*f))-total_logZ
        # 最后一项是正则化项
        # 带正则化项的对数似然函数
        likelihood = np.dot(feature_set.empirical_counts, params) - total_logZ - \
                     np.sum(np.dot(params,params))/(feature_set.squared_sigma*2)
        loss = -likelihood

        # 计算梯度（其中, logZ_sum 对 weights 的梯度, 都归结为此项。
        # 先考虑单样本情形, logZ 求偏导 =>F(有效)*P(y|x), 即下面的 prob 项;
        # 再考虑多样本情形即可）
        gradients = -(feature_set.empirical_counts - expected_counts -
         params/feature_set.squared_sigma) # 负梯度

        print('Step', step[0], 'Loss:', loss)

        return loss, gradients
```

线性链条件随机场类

```
In [6]: class LinearChainCRF:
            def __init__(self, filename):
                self.filename = filename
                self.feature_set = FeatureSet()

            def fit(self):
                print("* Reading training data ... ", end="")
                self.feature_set.ReadCorpusCONLL(self.filename)
                print("Done")

                self.feature_set.GenerateAllFeatures()
                print("* Number of labels: %d" % (len(self.feature_set.label_array)-1))
                print("* Number of features: %d" % self.feature_set.num_features)

                start_time = time.time()
                print('[%s] Start training' % datetime.datetime.now())
                print('* Squared sigma:', self.feature_set.squared_sigma)
                print('* Start L-BGFS')
                self.feature_set.params, loss, information = fmin_l_bfgs_b(
                 func = Loss, x0 = self.feature_set.params,
                 args = (self.feature_set, [0]))

                if information['warnflag'] != 0:
                    print('* Warning (code: %d)' % information['warnflag'])
                    if 'task' in information.keys():
                        print('* Reason: %s' % (information['task']))
                print('* Final loss: %s' % str(loss))

                elapsed_time = time.time() - start_time
```

```
        print('* Elapsed time: %f' % elapsed_time)
        print('* [%s] Training done' % datetime.datetime.now())
        self.SaveModel()

    def predict(self, filename = None):
        if filename == None:
            filename = self.filename

        test_feature_set = FeatureSet()
        test_feature_set.ReadCorpusCONLL(self.filename)

        total_count = 0
        correct_count = 0
        for X, Y in test_feature_set.data:
            potential_table = self.feature_set.GenerateMtable(
             self.feature_set.params, len(self.feature_set.label_array),
             [self.feature_set.GenerateYY_IDS(X, t) for t in range(len(X))])
            Y_HAT = self.Viterbi(X, potential_table)
            for t in range(len(Y)):
                total_count += 1
                if Y[t] == Y_HAT[t]:
                    correct_count += 1

        print('Correct: %d' % correct_count)
        print('Total: %d' % total_count)
        print('Performance: %f' % (correct_count/total_count))

    def Viterbi(self, X, potential_table):
        num_labels = len(self.feature_set.label_array)
        time_length = len(X)
        max_table = np.zeros((time_length, num_labels))
        argmax_table = np.zeros((time_length, num_labels), dtype='int64')

        t = 0
        for label_id in range(num_labels):
            max_table[t, label_id] = potential_table[t][0, label_id]
        for t in range(1, time_length):
            for label_id in range(1, num_labels):
                max_value = -float('inf')
                max_label_id = None
                for prev_label_id in range(1, num_labels):
                    value = max_table[t-1, prev_label_id] *
                     potential_table[t][prev_label_id, label_id]
                    if value > max_value:
                        max_value = value
                        max_label_id = prev_label_id
                max_table[t, label_id] = max_value
                argmax_table[t, label_id] = max_label_id
```

```
                sequence = []
                next_label = max_table[time_length-1].argmax()
                sequence.append(next_label)
                for t in range(time_length-1, -1, -1):
                    next_label = argmax_table[t, next_label]
                    sequence.append(next_label)
                return [self.feature_set.label_array[label_id] for label_id
                 in sequence[::-1][1:]]

            def SaveModel(self, filename = None):
                if filename == None:
                    filename = os.path.splitext(self.filename)[0] + '_new' + '.pkl'
                print('* Writing data into file "%s/%s"...' % (os.getcwd(), filename))
                with open(filename, 'wb') as f:
                    str = pickle.dumps(self.feature_set)
                    f.write(str)
                print('* Trained CRF Model has been saved at "%s/%s"' % (os.getcwd(),
                 filename))

            def LoadModel(self, filename = None):
                if filename == None:
                    filename = os.path.splitext(self.filename)[0] + '.pkl'
                print('* Loading file "%s/%s" ...' % (os.getcwd(), filename))
                self.feature_set = FeatureSet()
                with open(filename, 'rb') as f:
                    self.feature_set = pickle.loads(f.read())
                print('* Trained CRF Model has been loaded at "%s/%s"' % (os.getcwd(),
                 filename))
```

使用较小的训练数据集进行训练

```
In [7]: crf = LinearChainCRF('small_train.data')
        crf.fit()
```

```
* Reading training data ... Done
* Number of labels: 14
* Number of features: 31018
[2020-04-09 08:55:09.968802] Start training
* Squared sigma: 10.0
* Start L-BGFS
Step 1 Loss: 5003.65269695
Step 2 Loss: 4060.00809341
Step 3 Loss: 1943.63442154
Step 4 Loss: 1102.12231231
Step 5 Loss: 497.61166689
Step 6 Loss: 181.51651819
Step 7 Loss: 87.9987527507
```

```
Step 8 Loss: 44.3637129976
Step 9 Loss: 41.0589210027
Step 10 Loss: 35.4069726632
Step 11 Loss: 33.3593721226
Step 12 Loss: 32.1185389828
Step 13 Loss: 31.4017352761
Step 14 Loss: 30.7156707716
Step 15 Loss: 30.0662425865
Step 16 Loss: 29.5865062084
Step 17 Loss: 29.3233355665
Step 18 Loss: 29.2010378168
Step 19 Loss: 29.1035012415
Step 20 Loss: 29.0717114227
Step 21 Loss: 29.0298952744
Step 22 Loss: 29.0183871421
Step 23 Loss: 29.0004827719
Step 24 Loss: 28.9839983684
Step 25 Loss: 28.9888756275
Step 26 Loss: 28.9778307479
Step 27 Loss: 28.971453893
Step 28 Loss: 28.9694121189
Step 29 Loss: 28.9679331712
Step 30 Loss: 28.9670256181
Step 31 Loss: 28.966562413
Step 32 Loss: 28.9661968718
Step 33 Loss: 28.965935806
Step 34 Loss: 28.965880818
Step 35 Loss: 28.965656917
Step 36 Loss: 28.965626421
Step 37 Loss: 28.9655597551
Step 38 Loss: 28.9655435663
Step 39 Loss: 28.9655150935
Step 40 Loss: 28.9655107143
Step 41 Loss: 28.9655044219
Step 42 Loss: 28.9654997819
Step 43 Loss: 28.9655047043
Step 44 Loss: 28.9654989912
Step 45 Loss: 28.9654980378
Step 46 Loss: 28.9654976245
Step 47 Loss: 28.9654970586
Step 48 Loss: 28.9654969924
Step 49 Loss: 28.9654967564
Step 50 Loss: 28.9654966976
* Final loss: 28.9654966976
* Elapsed time: 29.928082
* [2020-04-09 08:55:39.896884] Training done
* Writing data into file "E:\dxq\2019-2020-2\机器学习\第 11 讲\实现 1
  /small_train_new.pkl"...
```

```
* Trained CRF Model has been saved at "E:\dxq\2019-2020-2\机器学习\第 11 讲\实现 1
  /small_train_new.pkl"
```

载入训练好的 CRF 模型，并在较小的测试数据集上进行测试

```
In [8]: test_crf = LinearChainCRF('small_test.data')
        test_crf.LoadModel('small_train_new.pkl')
        test_crf.predict()
```

```
* Loading file "E:\dxq\2019-2020-2\机器学习\第 11 讲\实现 1/small_train_new.pkl" ...
* Trained CRF Model has been loaded at "E:\dxq\2019-2020-2\机器学习\第 11 讲\实现 1
  /small_train_new.pkl"
Correct: 17237
Total: 19172
Performance: 0.899072
```

使用完整的训练数据集进行训练

```
In [9]: crf = LinearChainCRF('full_train.data')
        crf.fit()
```

```
* Reading training data ... Done
* Number of labels: 22
* Number of features: 1051501
[2020-04-09 09:02:13.340572] Start training
* Squared sigma: 10.0
* Start L-BGFS
Step 1 Loss: 654457.59004
Step 2 Loss: 550493.756926
Step 3 Loss: 268138.843217
Step 4 Loss: 148258.862569
Step 5 Loss: 98261.5130191
Step 6 Loss: 70158.0689031
Step 7 Loss: 56750.0350409
Step 8 Loss: 50207.4763284
Step 9 Loss: 46446.2009321
Step 10 Loss: 40053.1127408
Step 11 Loss: 34981.6296758
Step 12 Loss: 31672.0202771
Step 13 Loss: 28950.4652865
Step 14 Loss: 25666.271324
Step 15 Loss: 23255.9300538
Step 16 Loss: 20988.6652898
Step 17 Loss: 20052.9957806
Step 18 Loss: 18371.9685515
Step 19 Loss: 17192.5098554
Step 20 Loss: 15979.9171822
```

```
Step 21 Loss: 14891.2157996
Step 22 Loss: 13490.7295361
Step 23 Loss: 11778.6554907
Step 24 Loss: 11151.3112403
Step 25 Loss: 9364.72350401
Step 26 Loss: 8765.71846471
Step 27 Loss: 7648.21958948
Step 28 Loss: 7509.37421188
Step 29 Loss: 6997.85325813
Step 30 Loss: 6198.22809484
Step 31 Loss: 5395.00537909
Step 32 Loss: 4848.63787786
Step 33 Loss: 4251.82890234
Step 34 Loss: 4003.79487536
Step 35 Loss: 3462.38566837
Step 36 Loss: 3190.23593852
Step 37 Loss: 2843.75722274
Step 38 Loss: 2623.71359311
Step 39 Loss: 2393.12949592
Step 40 Loss: 2151.93235833
Step 41 Loss: 2091.62614409
Step 42 Loss: 1854.56116607
Step 43 Loss: 1805.81852042
Step 44 Loss: 1719.60868645
Step 45 Loss: 1648.62455428
Step 46 Loss: 1769.43841308
Step 47 Loss: 1606.75544175
Step 48 Loss: 1556.10512312
Step 49 Loss: 1521.57297841
Step 50 Loss: 1476.11633754
Step 51 Loss: 1451.02822782
Step 52 Loss: 1416.60031375
Step 53 Loss: 1398.38504998
Step 54 Loss: 1377.47346507
Step 55 Loss: 1346.31441907
Step 56 Loss: 1352.33694292
Step 57 Loss: 1326.61582828
Step 58 Loss: 1297.11750319
Step 59 Loss: 1279.81316464
Step 60 Loss: 1264.41499908
Step 61 Loss: 1246.3753666
Step 62 Loss: 1232.72960946
Step 63 Loss: 1219.10585421
Step 64 Loss: 1202.77579811
Step 65 Loss: 1200.78603975
Step 66 Loss: 1188.95329903
Step 67 Loss: 1167.76276579
Step 68 Loss: 1152.77181908
```

```
Step 69 Loss: 1134.62204705
Step 70 Loss: 1122.47703253
Step 71 Loss: 1109.86132449
Step 72 Loss: 1100.03030934
Step 73 Loss: 1088.98102168
Step 74 Loss: 1093.17013585
Step 75 Loss: 1083.09526061
Step 76 Loss: 1073.02496687
Step 77 Loss: 1064.7931255
Step 78 Loss: 1056.9184953
Step 79 Loss: 1047.9581206
Step 80 Loss: 1039.66143447
Step 81 Loss: 1031.80698237
Step 82 Loss: 1025.9103833
Step 83 Loss: 1020.40086535
Step 84 Loss: 1016.02212719
Step 85 Loss: 1009.82116254
Step 86 Loss: 1006.5430724
Step 87 Loss: 1003.38185737
Step 88 Loss: 1008.17350665
Step 89 Loss: 1001.51009381
Step 90 Loss: 998.858366015
Step 91 Loss: 993.96808805
Step 92 Loss: 991.412875804
Step 93 Loss: 987.074446602
Step 94 Loss: 988.034878312
Step 95 Loss: 984.495704561
Step 96 Loss: 981.786853622
Step 97 Loss: 979.458750549
Step 98 Loss: 977.025191141
Step 99 Loss: 975.280215261
Step 100 Loss: 972.50113546
Step 101 Loss: 971.236546748
Step 102 Loss: 969.313324689
Step 103 Loss: 969.310267093
Step 104 Loss: 968.214955881
Step 105 Loss: 966.489182033
Step 106 Loss: 965.268288484
Step 107 Loss: 963.390567143
Step 108 Loss: 962.711432571
Step 109 Loss: 960.740776826
Step 110 Loss: 959.886223764
Step 111 Loss: 958.83081882
Step 112 Loss: 957.963288175
Step 113 Loss: 956.757066477
Step 114 Loss: 956.004971234
Step 115 Loss: 955.179948546
Step 116 Loss: 953.888955654
```

```
Step 117 Loss: 954.305739982
Step 118 Loss: 953.241543151
Step 119 Loss: 952.240909905
Step 120 Loss: 951.791473658
Step 121 Loss: 951.408996941
Step 122 Loss: 950.821957605
Step 123 Loss: 950.491071485
Step 124 Loss: 949.850752754
Step 125 Loss: 949.475615472
Step 126 Loss: 949.167746368
Step 127 Loss: 948.596102082
Step 128 Loss: 949.221620666
Step 129 Loss: 948.313973341
Step 130 Loss: 947.961980286
Step 131 Loss: 947.726928682
Step 132 Loss: 947.374690969
Step 133 Loss: 947.033448046
Step 134 Loss: 946.850168358
Step 135 Loss: 946.592212389
Step 136 Loss: 946.480428013
Step 137 Loss: 946.368637867
Step 138 Loss: 946.0862147
Step 139 Loss: 946.078107256
Step 140 Loss: 945.915894364
Step 141 Loss: 945.653869729
Step 142 Loss: 945.503853027
Step 143 Loss: 945.364663261
Step 144 Loss: 945.252456291
Step 145 Loss: 945.138572124
Step 146 Loss: 944.972273644
Step 147 Loss: 944.890676486
Step 148 Loss: 944.801614984
Step 149 Loss: 944.732381562
Step 150 Loss: 944.673741248
Step 151 Loss: 944.548169962
Step 152 Loss: 944.482195865
Step 153 Loss: 944.380533699
Step 154 Loss: 944.260473553
Step 155 Loss: 944.212075729
Step 156 Loss: 944.131835371
Step 157 Loss: 944.078239981
Step 158 Loss: 944.030447139
Step 159 Loss: 943.970126755
Step 160 Loss: 943.936332407
Step 161 Loss: 943.887064222
Step 162 Loss: 943.828473662
Step 163 Loss: 943.806531209
Step 164 Loss: 943.757088473
```

```
Step 165 Loss: 943.699893004
Step 166 Loss: 943.685729514
Step 167 Loss: 943.633224439
Step 168 Loss: 943.611861254
Step 169 Loss: 943.582579286
Step 170 Loss: 943.55465191
Step 171 Loss: 943.525202585
Step 172 Loss: 943.497647804
Step 173 Loss: 943.468713259
Step 174 Loss: 943.457128436
Step 175 Loss: 943.431481737
Step 176 Loss: 943.41782629
Step 177 Loss: 943.404034673
Step 178 Loss: 943.379655485
Step 179 Loss: 943.465006432
Step 180 Loss: 943.373057879
Step 181 Loss: 943.354213302
Step 182 Loss: 943.339928254
Step 183 Loss: 943.328738419
Step 184 Loss: 943.313336001
Step 185 Loss: 943.308762966
Step 186 Loss: 943.299505009
Step 187 Loss: 943.288629592
Step 188 Loss: 943.268738658
Step 189 Loss: 943.330939442
Step 190 Loss: 943.263645969
Step 191 Loss: 943.253240645
Step 192 Loss: 943.245270411
Step 193 Loss: 943.239287844
Step 194 Loss: 943.230885829
Step 195 Loss: 943.22707532
Step 196 Loss: 943.22156417
Step 197 Loss: 943.214358403
Step 198 Loss: 943.209569476
Step 199 Loss: 943.204565437
Step 200 Loss: 943.202516776
Step 201 Loss: 943.198386281
Step 202 Loss: 943.191745147
Step 203 Loss: 943.187630411
Step 204 Loss: 943.183850643
Step 205 Loss: 943.180834468
Step 206 Loss: 943.17820486
Step 207 Loss: 943.175664433
Step 208 Loss: 943.172436411
Step 209 Loss: 943.170596262
Step 210 Loss: 943.168952537
Step 211 Loss: 943.164440009
Step 212 Loss: 943.173682373
```

```
Step 213 Loss: 943.162965631
Step 214 Loss: 943.161030314
Step 215 Loss: 943.159525388
Step 216 Loss: 943.157716248
Step 217 Loss: 943.155974194
Step 218 Loss: 943.15424064
Step 219 Loss: 943.1529873
Step 220 Loss: 943.151479224
Step 221 Loss: 943.150381541
Step 222 Loss: 943.149533528
Step 223 Loss: 943.148336209
Step 224 Loss: 943.146812845
Step 225 Loss: 943.145754456
Step 226 Loss: 943.145179045
Step 227 Loss: 943.143888751
Step 228 Loss: 943.143505527
Step 229 Loss: 943.142312208
Step 230 Loss: 943.141831754
Step 231 Loss: 943.141169104
Step 232 Loss: 943.141637639
Step 233 Loss: 943.140799751
Step 234 Loss: 943.140203822
Step 235 Loss: 943.139715688
Step 236 Loss: 943.139368219
Step 237 Loss: 943.139079088
Step 238 Loss: 943.138756988
Step 239 Loss: 943.138373832
Step 240 Loss: 943.137607315
Step 241 Loss: 943.137336055
Step 242 Loss: 943.136897169
Step 243 Loss: 943.136717721
Step 244 Loss: 943.136443805
Step 245 Loss: 943.13605558
Step 246 Loss: 943.135764857
Step 247 Loss: 943.13555705
Step 248 Loss: 943.13539671
Step 249 Loss: 943.135238026
Step 250 Loss: 943.135127368
Step 251 Loss: 943.134785764
Step 252 Loss: 943.134700995
Step 253 Loss: 943.134509309
Step 254 Loss: 943.134314601
Step 255 Loss: 943.134226346
Step 256 Loss: 943.134049708
Step 257 Loss: 943.133967446
Step 258 Loss: 943.133872296
Step 259 Loss: 943.133714323
Step 260 Loss: 943.133571749
```

```
Step 261 Loss: 943.133494195
Step 262 Loss: 943.133398795
Step 263 Loss: 943.133340515
Step 264 Loss: 943.133244435
Step 265 Loss: 943.133168786
Step 266 Loss: 943.13310771
Step 267 Loss: 943.132985964
Step 268 Loss: 943.133070741
Step 269 Loss: 943.132931489
Step 270 Loss: 943.132849971
Step 271 Loss: 943.132794482
Step 272 Loss: 943.132760783
Step 273 Loss: 943.13269898
Step 274 Loss: 943.132674446
Step 275 Loss: 943.132637725
Step 276 Loss: 943.132581872
Step 277 Loss: 943.132857815
Step 278 Loss: 943.132560635
Step 279 Loss: 943.132508744
Step 280 Loss: 943.13248116
Step 281 Loss: 943.132438841
Step 282 Loss: 943.132406007
Step 283 Loss: 943.132371918
Step 284 Loss: 943.132340522
Step 285 Loss: 943.132316889
Step 286 Loss: 943.132298709
Step 287 Loss: 943.132278598
Step 288 Loss: 943.132255382
Step 289 Loss: 943.1322335
Step 290 Loss: 943.132218421
Step 291 Loss: 943.132198335
Step 292 Loss: 943.132184709
Step 293 Loss: 943.132169989
Step 294 Loss: 943.132155059
Step 295 Loss: 943.132139454
Step 296 Loss: 943.132124124
Step 297 Loss: 943.13211103
Step 298 Loss: 943.132104197
Step 299 Loss: 943.132094201
Step 300 Loss: 943.132082597
Step 301 Loss: 943.132070452
Step 302 Loss: 943.132064179
Step 303 Loss: 943.132057539
Step 304 Loss: 943.13204781
Step 305 Loss: 943.132039962
Step 306 Loss: 943.132034501
Step 307 Loss: 943.132029867
Step 308 Loss: 943.132024492
```

```
Step 309 Loss: 943.132021839
Step 310 Loss: 943.132012728
Step 311 Loss: 943.132009881
Step 312 Loss: 943.132005786
Step 313 Loss: 943.132001853
Step 314 Loss: 943.131997797
Step 315 Loss: 943.131995244
Step 316 Loss: 943.131992779
Step 317 Loss: 943.131989964
Step 318 Loss: 943.131987143
Step 319 Loss: 943.131984494
Step 320 Loss: 943.131982235
Step 321 Loss: 943.131978865
Step 322 Loss: 943.131976463
Step 323 Loss: 943.131974806
* Final loss: 943.131974806
* Elapsed time: 84938.303673
* [2020-04-10 08:37:51.644245] Training done
* Writing data into file "E:\dxq\2019-2020-2\机器学习\第 11 讲\实现 1
  /full_train_new.pkl"...
* Trained CRF Model has been saved at "E:\dxq\2019-2020-2\机器学习\第 11 讲\实现 1
  /full_train_new.pkl"
```

载入训练好的 CRF 模型，并在完整的测试数据集上进行测试

```
In [7]: test_crf = LinearChainCRF('full_test.data')
        test_crf.LoadModel('full_train_new.pkl')
        test_crf.predict()
```

```
* Loading file "E:\dxq\2019-2020-2\机器学习\第 11 讲\实现 1
  /full_train_new.pkl" ...
* Trained CRF Model has been loaded at "E:\dxq\2019-2020-2\机器学习\第 11 讲\实现 1
  /full_train_new.pkl"
Correct: 45488
Total: 47377
Performance: 0.960128
```

7 参考文献

1. Charles Sutton and Andrew McCallum. An Introduction to Conditional Random Fields, DOI: 10.1561/2200000013.

2. Trevor Hastie, Robert Tibshirani, Jerome Friedman. The Elements of Statistical Learning, Data Mining, Inference, and Prediction. Second Edition. Springer Series in Statistics.

3. 周志华。《机器学习》，清华大学出版社，2016 年 1 月第 1 版。

4. Christopher M. Bishop. Pattern Recognition and Machine Learning. Springer, 2006.

5. Kevin P. Murphy. Machine Learning: A Probabilistic Perspective, The MIT Press, 2012.

6. 李航。《统计学习方法》，清华大学出版社，2019 年 5 月第 2 版。

7. https://zhuanlan.zhihu.com/p/54101808.

8. https://github.com/lancifollia/crf/.

9. https://github.com/fferegrino/vuelax-crf.

10. https://dev.to/fferegrino/sequence-labelling-in-python-part-1-4noa, Part1-5.

11. https://github.com/shuyo/iir/blob/master/sequence/crf.py.

12. https://baike.baidu.com/item/条件随机场/10804560.

13. https://www.zhihu.com/question/24094554.

14. https://www.zhihu.com/question/20380549/answer/45066785.

15. https://en.wikipedia.org/wiki/Inside-outside-beginning_(tagging).

16. https://towardsdatascience.com/a-practitioners-guide-to-natural-language-processing-part-i-processing-understanding-text-9f4abfd13e72.

17. https://www.geeksforgeeks.org/nlp-iob-tags/.

18. http://nathanlvzs.github.io/Several-Tagging-Schemes-for-Sequential-Tagging.html.