序列建模（树）

Tip

手写核心结构： 绝对不要直接调用类库！ 哪怕用 Python，也要自己定义 `class Node` 和 `class LinkedList`。在报告里贴出你自己写的代码片段，并加上详细的注释。

价值：数据结构结课报告/电商漏斗/跳表工程优化/序列建模

报告核心题目建议

《基于概率前缀树（Trie）与多层跳跃索引的消费序列 OOD 检测系统设计与实现》

一、 数据结构设计：从基础到进阶

数据集：用户行为数据集，工程可行?

1. 基础结构：Bayesian Trie (贝叶斯前缀树)

- **节点：** 结构体，存储 `category_id`、`count`（访问次数）和 子节点指针（哈希表）。

class TrieNode:

slots = ['count', 'children', 'is_end'] # 内存优化必杀技

def **init**(self):

self.count = 0 # 核心：经过该节点的路径总数（用于计算条件概率）

self.children = {} # 核心：哈希表，Key 是(category_id, behavior_type)，Value 是指向子节点的指针（对象）

self.is_end = False # 辅助：标记是否有完整的消费序列在此结束

在本系统中，我们舍弃了**数组索引**（因为 `category_id` 空间极其稀疏，会造成 $O(MAX\_ID)$ 的空间浪费）和**链表索引**（查询复杂度 $O(K)$），选择了**哈希表索引**。这使得在处理拥有数万个类目的电商数据时，单次行为跳转的平均时间复杂度依然保持在 $O(1)$，确保了 OOD 检测的实时性。”

路径（Path）：一个（或一组）顾客的完整行为序列。

树保留了完整的**历史序列信息**，这对于计算条件概率 $P(C_n | C_{n-1}, ..., C_0)$ 至关重要。

特征工程，数据清洗

1. **数据清洗**：去重（重复行为记录）、处理缺失值、过滤异常值（如单次点击上万次的作弊用户）；
2. **时间维度拆解**：将时间戳转化为 “日期、小时、星期几”，分析用户活跃时段（如晚间 20-22 点是购物高峰）；
3. **行为类型编码**：将 “点击 / 收藏 / 加购 / 下单” 映射为权重（如下单 = 4，加购 = 3，收藏 = 2，点击 = 1），计算用户行为权重总和；
4. **特征衍生**：重点构造转化率、频次、间隔类特征，这类特征对电商建模最有价值

2. 进阶结构：Skip-Trie (跳跃前缀树 —— 知识迁移点)

- **设计灵感：** 借鉴跳表（Skip List）的多级索引逻辑。
- **实现逻辑：** 对于高频出现的固定长序列（如 `A -> B -> C -> D`），在 Root 节点或高层节点建立直达 D 的“跳跃指针”。
- **DS 价值：** 在处理海量数据时，减少树遍历的深度，体现对 **空间换时间** 和 **缓存友好性**（计组知识）的理解。

```
class TrieNode:
    __slots__ = ['count', 'children', 'skip_pointers', 'is_end']

    def __init__(self):
        self.count = 0      # 统计经过此节点的序列总数
        self.is_end = False  # 序列结束标志

        # --- 1. 横向哈希索引 (基础 Trie) ---
        # 解决“下一步点什么”的分叉问题
        # Key: (category_id, behavior_type), Value: TrieNode 对象
        self.children = {}

        # --- 2. 纵向跳表索引 (加速快表) ---
        # 解决“长序列纵向跳转”效率问题。类似于计组中的 TLB 或多级索引。
        # Key: 步长或特定模式的 Hash
        # Value: 指向该序列末尾节点的直接指针 (Direct Reference)
        self.skip_pointers = {}
```

A. 捕捉“高频确定性”（时间局部性）

- **要求**：只有那些转移概率 $P(B|A) \approx 1.0$ 的序列才值得存入。
- **价值**：如果 A 后面 99% 的概率接 B，那么 `A -> B` 就是一个“静态路由”，存入快表可以消除哈希计算的开销。

B. 捕捉“关键漏斗路径”（业务局部性）

- **要求**：捕捉从“加购”到“下单”的黄金路径。
- **价值**：这些路径虽然不一定是最高频的，但它们是**价值密度最高**的。在 OOD 检测中，这些路径的“命中”代表用户处于高度安全的交易状态。

二、 场景建模：ID vs OOD

你需要用数学和逻辑定义什么是“正常”，什么是“异常”。

1. 基础场景建模（In-Distribution, ID）

- **定义：** 符合历史统计规律的消费行为。
- **建模：** 序列 S 在 Trie 树中路径完整，且路径上的累积条件概率 $\prod P(c_i | prefix) > \tau$。
- **特性：** 体现了用户购物的逻辑性（如：搜索手机 -> 对比参数 -> 下单手机壳）。

2. OOD 场景建模（Out-of-Distribution）

- **类型 A：拓扑断裂 (Hard OOD)**
  - **描述：** 输入序列在 Trie 树某一层找不到子节点。
  - **建模：** $c_{next} \notin node.children$。
  - **现实意义：** 全新的消费模式或系统未见的品类组合。
- **类型 B：概率坍塌 (Soft OOD / Anomaly)**
  - **描述：** 路径虽然存在，但逻辑极其罕见。
  - **建模：** 某一步的转移概率 $P < \epsilon$。
  - **现实意义：** 可能是恶意刷单、爬虫抓取（行为模式与人类统计规律不符）。

类型c：因果违背型异常 (Causal / Logic Outlier)

- **现象：** 序列虽然在类目跳转上正常，但在**权重转换（因果链）**上违背了电商的“漏斗模型”。
- **建模：** * **能量突变：** 序列很短但权重总和（Weight Sum）极高（如：冷启动后直接 4 级下单）。
  - **逆序因果：** 在 Trie 树的分支逻辑中，`（A，下单）-> （A，点击）` 的频率极低。

- **现实意义：** * **机器作弊：** 爬虫或自动脚本通常跳过预览和比价，直接触发加购或下单。
  - **异常流量：** 某种非人类点击流驱动的瞬间转化。
- **报告亮点：** 强调将行为类型嵌入 `(cat, type)` 复合索引后，Trie 树从“搜索树”升级为“语义因果模型”。

三、 算法流程与复杂度分析（高分必备）

1. 算法伪代码逻辑

第一阶段：离线训练与树构建 (Offline Building)

这个阶段的目标是把原始的消费数据转化为一颗“充满统计信息”的 Trie 树。

1. **数据预处理**：读取 CSV，执行去重、过滤刷单异常、时间排序。
2. **序列生成**：将每个用户的行为转化为 `[(cat1, type1), (cat2, type2), ...]` 格式。
3. **递归建树**：
   - 从 `Root` 开始，按序列插入节点。
   - 每个节点更新 `count` （访问次数）。
   - **哈希表索引**：`current.children[(cat, type)] = next_node`。

第二阶段：快表特征捕捉 (TLB-like Feature Capture)

这是你强调的**工程价值**所在。我们需要选出哪些路径值得放入 `skip_pointers`。

1. **模式挖掘**：统计所有长度为 N（建议 $N=3$）的子序列出现的频次。
2. **置信度筛选**：
   - 计算转移概率 $P(C|A,B)$。如果该路径极其稳定且高频，则判定为“热点特征”。
3. **建立快表**：
   - 在起始节点 A 的 `skip_pointers` 字典中，直接存入指向终点节点 C 的内存地址。
   - **存储内容**：`root.skip_pointers[pattern_tuple] = end_node_reference`。

第三阶段：在线 OOD 检测流程 (Online Detection)

这是你强调的**现实价值**所在。当一个新的序列进来时，系统如何判定它是 ID（正常）还是 OOD（异常）。

算法伪代码逻辑：

1. **初始化**：指针指向 `Root`，总异常分 $S = 0$。
2. **循环扫描输入序列**：
   - **Step A：快表尝试 (TLB Hit?)**
     - 从当前位置取长度为 N 的片段，查询 `current.skip_pointers`。
     - **如果命中**：直接“瞬移”到目标节点，跳过中间哈希计算，记录“快表命中”。
   - **Step B：基础哈希检索 (Standard Search)**
     - 如果快表未命中，查询 `current.children`。
     - **如果失败** $\to$ **触发 Hard OOD**（路径断裂），中断并报警。
   - **Step C：概率评估 (Probabilistic Check)**
     - 计算 $P = \frac{next\_node.count}{current.count}$。
     - $S = S + (-\ln P)$。
     - 如果 S 超过动态阈值 $\to$ **触发 Soft OOD**（统计异常）。
3. **结果输出**：返回最终异常得分及判定类型。

2. 复杂度对比表（体现专业性）

维度	传统线性存储	标准 Trie 树	你的 Skip-Trie
查询复杂度	$O(N \times L)$	$O(L)$	$O(L / \text{jump\_step})$
空间复杂度	$O(N \times L)$	$O(\text{Unique Prefixes})$	$O(\text{Unique Prefixes} + \text{Skip Pointers})$
OOD 检测能力	需重新计算分布	静态节点概率对比	动态概率 + 快速模式匹配

四、 研究创新点（报告结论提炼）

1. **跨学科迁移：** 成功将计组中的跳表思想应用于树形结构，优化了长序列 OOD 检测的响应速度。
2. **可解释性 OOD：** 不同于神经网络的黑盒判定，本模型能精确指出序列中哪一个节点（行为）导致了分布偏离。
3. **空间压缩率：** 通过实验证明 Trie 树对电商序列数据的压缩比，以及 Skip 索引带来的额外空间开销在可接受范围内。

五、 实验日记

基础baseline版

跳表版