第 23 章 时长预估建模

在工业级推荐系统中，时长预估是一个非常重要、但又经常被低估的建模问题。尤其是在短视频、直播、信息流等以用户消费时长为核心指标的推荐场景中，用户对一个内容的观看时长往往比简单的点击行为包含更加丰富的兴趣信息。例如，对于两个用户都点击了某个短视频，仅仅通过点击行为很难判断两个用户对该内容的兴趣程度。但如果一个用户只观看了 1 秒就快速划走，而另一个用户观看了 40 秒甚至完整看完，那么二者显然具有完全不同的用户兴趣强度。因此，相比 CTR 等传统点击类目标，观看时长能够更加细粒度地刻画用户与内容之间的交互强度。

本章节主要围绕**短视频场景下的时长预估问题**展开讨论。短视频是当前推荐系统中时长建模最为典型的应用场景之一，其核心问题是根据用户、内容以及上下文等信息，预估用户在一次内容曝光下可能产生的观看行为和观看时长。需要说明的是，虽然不同业务场景下的“时长”定义和具体业务含义有所差异，例如直播场景中的直播间停留时长、信息流场景中的页面停留时长等，但其背后的时长建模思想具有较强的共通性。因此，本章以短视频时长预估作为主要讨论对象，同时介绍的方法和建模思路也可以自然延伸到其他具有类似时长预估需求的推荐场景。

然而，时长预估并不是简单地将 CTR 中的 Label 从 0/1 替换成一个连续的观看秒数。真正进入工业系统后，我们首先需要回答三个基础但非常关键的问题。第一个问题是：

笔记 时长应该如何定义？

是直接使用用户实际观看的连续时长作为回归目标，还是将观看时长离散化为二分类或者多分类标签？例如，对于一次曝光，如果用户观看了 15 秒，我们究竟应该让模型直接预测 15 秒，还是判断用户是否观看超过 5 秒、10 秒、15 秒等不同的时长阈值？看似只是 Label 定义上的差异，实际上对应着不同的建模范式：前者更接近连续值回归问题，而后者则可以转化为多个分类概率的预估问题。不同的 Label 定义方式不仅会影响模型的输出形式和损失函数设计，还会进一步影响样本分布、模型训练稳定性以及最终线上排序效果。因此，在讨论“如何预估时长”之前，首先需要明确“我们究竟希望模型预估什么样的时长目标”。

更重要的是，观看时长本身具有非常明显的长尾特征。大量用户可能只观看几秒，而少量用户可能观看几十秒甚至数分钟。如果直接将原始观看时长作为连续值进行回归，模型很容易受到长时长样本的影响，同时也会面临预测分布难以拟合的问题。因此，工业界通常不会简单地认为“真实时长是多少，就直接预测多少”。此外，我们还需要关注下面的第二个问题：

笔记 要预估的究竟是什么时长？

这里的“时长”并不是一个唯一确定的概念。对于短视频场景，可以预估用户在单次曝光下的观看时长；对于直播场景，可以预估用户进入直播间之后的停留时长；对于信息流场景，还可以关注用户在侧边栏、评论区、作者主页等不同区域的停留时间。

进一步地，如果从整个推荐会话来看，还可以定义用户在一次 Session 内的累计观看时长，甚至进一步优化用户在整个生命周期内的长期消费时长。这些目标虽然都可以被称为“时长”，但实际上对应的是完全不同的优化问题。单次曝光下的观看时长更接近 Item-level Prediction，而 Session 级累计时长则已经涉及用户连续决策、内容之间的相互影响以及长期收益，因此后者通常需要结合序列建模或者强化学习进行优化。在本章中，我们暂且只关注**非 Session 级别的时长预估问题**，即用户在一次内容曝光下的观看行为以及对应的观看时长。对于 Session 级别的时长建模以及长期时长优化，我们将在后续第32章中进一步讨论。

最后，我们需要关注时长预估建模的第三个问题，即：

笔记 时长究竟应该如何建模？

即使已经确定了单次曝光观看时长作为目标，仍然存在不同的建模方式。例如，可以直接预测用户观看时长的期望值，也可以预测用户观看超过某个时间阈值的概率，还可以进一步将完整的观看时长离散成多个区间，对整个时长分布进行建模。

因此，时长预估真正困难的地方并不是简单地“预测一个秒数”，而是需要根据业务目标选择合适的 Label 定

义和建模方式。从工业推荐系统的实践来看，一个非常重要的思路是：

$$\text{Continuous Watch Time} \rightarrow \text{Threshold-based Labels} \rightarrow \text{Classification} \tag{23.1}$$

也就是说，不一定直接预测用户最终观看了多少秒，而是围绕业务真正关心的“有效观看”、“长观看”等行为定义多个时长阈值，并将其转化为概率预估问题。这种方法最大的优势在于，它能够直接将模型输出与推荐系统中的业务指标建立联系。例如，如果线上最终关心的是“用户观看超过 10 秒的概率”，那么模型就直接学习 $P(W > 10)$，而不是先预测一个观看时长，再通过人为规则将预测时长转换成“是否超过 10 秒”。因此，在工业推荐系统中，一个非常经典且实用的时长建模方法，就是**基于阈值化定义的时长预估（Threshold-based Duration Prediction）**。

23.1 基于阈值化定义的时长建模

在工业级推荐系统中的时长预估中，最简单的一类时长定义是基于阈值离散化（Threshold Binarization）的定义方式。假设一次曝光下用户的实际观看时长为 W，我们定义一个时长阈值 T。当用户观看时长超过该阈值时，将其定义为有效观看，否则定义为无效观看，即

$$y_T = \mathbb{I}(W > T) \tag{23.2}$$

其中，$\mathbb{I}(\cdot)$ 为指示函数。例如，当我们设置 $T = 10$ 时，对于不同的用户行为：

$$W = 3 \Rightarrow y_{10} = 0, \quad W = 8 \Rightarrow y_{10} = 0, \quad W = 15 \Rightarrow y_{10} = 1, \quad W = 35 \Rightarrow y_{10} = 1 \tag{23.3}$$

这样，原本连续的观看时长就被转换成了一个标准的二分类 Label。此时，模型学习的目标不再是直接预测具体观看了多少秒，而是预测用户观看超过该阈值的概率：

$$p_T = P(W > T \mid X) \tag{23.4}$$

其中 X 表示用户、Item 以及上下文特征。例如，当 $T = 10$ 秒时，模型预测 $p_{10} = P(W > 10 \mid X)$，这个概率可以直接理解为用户对该内容产生“10 秒以上有效观看”的概率。由于 Label 是 0/1 二值变量，因此可以直接采用二分类交叉熵进行训练：

$$\mathcal{L}_T = -y_T \log p_T - (1 - y_T) \log(1 - p_T) \tag{23.5}$$

在实际模型中，通常令模型输出 Logit z_T，再通过 Sigmoid 得到最终概率：

$$p_T = \sigma(z_T) = \frac{1}{1 + \exp(-z_T)} \tag{23.6}$$

对应的训练目标也可以直接写成 BCEWithLogitsLoss：

$$\mathcal{L}_T = \text{BCEWithLogits}(z_T, y_T) \tag{23.7}$$

这种建模方式非常简单，但它有一个非常重要的优势：**模型的预测目标与业务指标之间具有非常直接的对应关系**。例如，如果线上业务关注 EVTR（Effective View Time Rate），那么可以直接定义：

$$EVTR_T = P(W > T) \tag{23.8}$$

模型实际上就在学习这个指标对应的概率。如果将阈值设置得更高，例如从 $T = 10$ 秒提高到 $T = 20$ 秒，则模型对应的目标就可以理解为更加严格的长观看概率：

$$LVTR = P(W > 20) \tag{23.9}$$

因此，从这个角度来看，EVTR、LVTR 等指标本质上都可以理解为不同观看时长阈值下的 Survival Probability（生存概率）：

$$S(T) = P(W > T) \tag{23.10}$$

其中 W 表示随机变量形式的观看时长，而 $S(T)$ 表示用户观看超过 T 秒的概率。这也意味着，当我们选择不同的 T 时，实际上是在观察用户观看时长分布的不同位置。例如，可以同时定义：

$$p_3 = P(W > 3), \quad p_{10} = P(W > 10), \quad p_{20} = P(W > 20), \quad p_{30} = P(W > 30) \tag{23.11}$$

这些概率分别对应不同程度的用户观看深度。从工程角度来看，这也是阈值化时长建模非常重要的一个优势：**通过改变阈值，就可以控制模型关注用户观看行为的不同深度，而不需要改变整个推荐模型的主体结构。**

23.1.1 Duration-aware 的时长阈值定义

然而，如果直接为所有视频设置一个统一的时长阈值，又会产生一个非常明显的问题。假设我们统一设置 $T = 10s$，那么对于一个只有 12 秒的短视频，用户观看 10 秒已经意味着其观看了绝大部分内容；而对于一个长度为 120 秒的视频，用户观看 10 秒可能只代表用户刚刚开始消费内容。那么对于一个只有 12 秒的短视频，用户观看 10 秒已经意味着其观看了绝大部分内容；而对于一个长度为 120 秒的视频，用户观看 10 秒可能只代表用户刚刚开始消费内容。因此，这说明了以下的结论：

命题 23.1

相同的观看秒数对于不同物理时长的内容，其行为意义并不相同。

这也是工业短视频推荐中非常常见的一种优化思路：根据 Item 自身的物理 Duration，为不同 Duration 的内容定义不同的时长阈值。假设视频物理时长为 D，可以首先按照视频 Duration 将 Item 划分为若干个 Duration 分桶：$D \in B_k$，其中 B_k 表示第 k 个 Duration 分桶。例如，可以定义：

$$B_1 = [0, 15), \quad B_2 = [15, 30), \quad B_3 = [30, 60), \quad B_4 = [60, 120), \quad B_5 = [120, +\infty) \tag{23.12}$$

然后，为每个 Duration 分桶设置独立的 EVTR 阈值：

$$T_{EVTR}(D) = T_k^{EVTR}, \qquad D \in B_k \tag{23.13}$$

同理，可以设置 LVTR 阈值：

$$T_{LVTR}(D) = T_k^{LVTR}, \qquad D \in B_k \tag{23.14}$$

最终得到两个不同的二分类 Label：

$$y_{EVTR} = \mathbb{I}(W > T_{EVTR}(D)), \quad y_{LVTR} = \mathbb{I}(W > T_{LVTR}(D)) \tag{23.15}$$

这里最关键的一点在于，**阈值不再是一个固定常数，而是一个与 Item Duration 相关的函数。**所以，整个研究问题就变成了如下形式，即：

笔记 如何定义一个合理的 $T_{EVTR}(D)$ 和 $T_{LVTR}(D)$ 函数，使得不同 Duration 的视频在达到正 Label 时具有相对一致的观看比例？

23.1.2 基于比例的 Duration-aware 时长阈值定义

一种更加通用的定义方式，是直接使用视频物理时长 D 的比例来确定观看阈值。例如，可以定义：

$$T_{EVTR}(D) = \alpha D, \quad T_{LVTR}(D) = \beta D \tag{23.16}$$

其中 $0 < \alpha < \beta < 1$。例如设置：

$$\alpha = \frac{1}{3}, \qquad \beta = \frac{2}{3} \tag{23.17}$$

则对于一个物理时长为 30 秒的视频：

$$T_{EVTR} = 10s, \quad T_{LVTR} = 20s \tag{23.18}$$

而对于一个物理时长为 60 秒的视频：

$$T_{EVTR} = 20s, \quad T_{LVTR} = 40s \tag{23.19}$$

这样就可以得到：

$$y_{EVTR} = \mathbb{I}\left(\frac{W}{D} > \alpha\right), \quad y_{LVTR} = \mathbb{I}\left(\frac{W}{D} > \beta\right) \tag{23.20}$$

从这个角度来看，模型实际上是在判断用户是否消费了视频足够高的比例。例如，表23.1这种定义方式能够在一定程度上消除不同视频物理时长带来的 Label 偏差。不过，在工业级推荐系统中也并不一定严格按照固定比例定义阈值。因为用户观看行为本身并不是简单的线性函数。例如，对于一个 10 秒的视频，用户观看 5 秒可能已经是非常强的兴趣；而对于一个 5 分钟的视频，用户观看 5 分钟并不一定意味着用户对内容的兴趣强度是观看 5 秒视频时的 30 倍。因此，实际系统中更常见的做法往往是 **Duration 分桶 + 经验阈值**，而不是简单地使用一个固定比例。

表 23.1: 基于视频物理时长比例定义 EVTR 阈值的示例。

视频 Duration	观看时长	观看比例	EVTR 阈值	EVTR Label
30s	5s	16.7%	10s	0
30s	15s	50.0%	10s	1
30s	25s	83.3%	10s	1
60s	10s	16.7%	20s	0
60s	30s	50.0%	20s	1
60s	50s	83.3%	20s	1

23.1.3 基于 Duration 分桶的经验阈值

在实际工业推荐系统中，用户观看行为与视频物理 Duration 之间通常并不是简单的比例关系。因此，直接采用固定比例并不一定能够得到最优的时长 Label 定义。固定比例虽然可以消除不同物理 Duration 带来的绝对时长差异，但同时也可能忽略观看行为中的绝对时长信息。例如，假设 EVTR 被定义为观看比例超过 50%，那么对于一个物理时长为 10 秒的视频，用户只需要观看 5 秒即可获得正 Label；而对于一个物理时长为 5 分钟的视频，则需要持续观看 2.5 分钟才能获得正 Label。虽然二者具有完全相同的观看比例，但从用户行为的角度来看，这两个观看行为所包含的信息量并不一定相同。

特别是在短视频场景中，较短的绝对观看时长本身可能具有较弱的区分度。用户在浏览短视频时，即使没有产生明显的兴趣，也可能因为正常的内容加载、视觉刺激或者滑动决策而自然停留 3 ∼ 5 秒。因此，一个 10 秒视频被观看 5 秒，虽然已经达到 50% 的观看比例，但并不一定意味着用户产生了非常强的兴趣。相反，一个 5 分钟的视频能够持续观看 2.5 分钟，通常需要用户在较长时间内持续保持注意力，因此往往能够提供更强的用户兴趣信号。

这说明了以下的结论，即：

命题 23.2

观看比例和绝对观看时长实际上分别刻画了用户观看行为的不同维度。观看比例能够反映用户对视频内容的相对消费深度，而绝对观看时长则能够反映用户持续消费内容的时间长度。只使用固定比例作为时长 Label 的定义，相当于假设不同物理 Duration 下相同的观看比例具有完全一致的行为意义。但在实际的短视频业务场景中，这一假设对实际用户行为来说往往并不成立。

♠

因此，工业级推荐系统中的时长 Label 定义通常需要在**相对消费深度**和**绝对观看时长**之间进行权衡。一方面不能简单使用固定的绝对时长阈值，否则容易产生前面讨论的 Duration Bias，使短视频天然更容易达到阈值；另一方面也不能完全依赖固定观看比例，否则又可能低估较短视频中绝对观看时长所提供的兴趣信息。在实际工业级推荐系统中，更常见的做法往往是采用 **Duration 分桶 + 经验阈值**的方式，根据不同物理 Duration 区间中的真实用户观看分布和业务目标，对 EVTR、LVTR 等时长阈值进行独立校准，而不是简单地使用一个固定比例函数。与上一小节中的连续比例函数相比，这种方法不再强制要求时长阈值与视频物理 Duration 保持严格的线性关系，而是允许不同 Duration 区间拥有不同的阈值增长速度，从而更加灵活地拟合实际用户观看行为。

具体来说，假设将视频按照物理 Duration 划分成如下几个 Duration 分桶：

$$B_1 = [0, 30), \quad B_2 = [30, 60), \quad B_3 = [60, 120), \quad B_4 = [120, 300), \quad B_5 = [300, 600) \tag{23.21}$$

可以根据历史数据分布、用户观看行为以及业务经验，为每个 Duration 分桶设置不同的 EVTR 和 LVTR 阈值，如

表23.2所示。此时，对于任意一个视频 i，首先根据其物理 Duration D_i 确定其所属的 Duration Bucket B_k，然后使用该 Bucket 对应的时长阈值：

$$T_i^{EVTR} = T_k^{EVTR}, \qquad T_i^{LVTR} = T_k^{LVTR}, \qquad D_i \in B_k \tag{23.22}$$

最终得到对应的 EVTR 和 LVTR 二分类 Label：

$$y_i^{EVTR} = \mathbb{I}\left(W_i > T_k^{EVTR}\right), \quad y_i^{LVTR} = \mathbb{I}\left(W_i > T_k^{LVTR}\right) \tag{23.23}$$

其中，W_i 表示用户对视频 i 的实际观看时长。

表 23.2: 基于 Duration 分桶的经验阈值定义示例。

Duration 分桶	EVTR 阈值	LVTR 阈值
$[0, 30)$	5s	15s
$[30, 60)$	10s	30s
$[60, 120)$	20s	60s
$[120, 300)$	30s	90s
$[300, 600)$	40s	120s

相比上一小节的固定比例定义，这种经验阈值的优势在于具有更强的灵活性。例如，一个 20 秒的视频观看 10 秒，可以认为用户已经消费了相当一部分内容；但对于一个 3 分钟的视频而言，观看 10 秒可能仅仅代表用户进行了非常浅层的内容消费。因此，不同物理 Duration 的视频如果共享完全相同的绝对 EVTR Label 定义，实际上会面临明显不同的 Label 难度。

这一问题在工业短视频推荐系统中还可能进一步表现为明显的 **Duration Bias**。短视频天然具有更高的有效播放率、长播率以及完播率。当 EVTR、LVTR 等指标采用固定绝对时长作为阈值时，较短的视频更加容易达到对应的正 Label。例如，如果 EVTR 被定义为“观看超过 10 秒”，那么一个 12 秒的视频只需要用户观看超过 10 秒即可获得正 Label；而一个 120 秒的视频即使用户观看了 10 秒，其实际消费深度可能仍然非常有限。因此，在固定绝对时长阈值的定义下，不同 Duration 的视频实际上并没有处于完全公平的 Label 评价尺度上。模型在优化 EVTR、LVTR 等目标时，很容易将“物理 Duration 较短”这一特征与“更容易产生有效观看”联系起来，从而逐渐形成对短视频的系统性偏好。这种偏好并不一定完全来自用户真实兴趣，而可能部分来自 Label 定义本身所带来的结构性偏差。

更进一步地，当推荐系统长期围绕这类时长阈值指标进行优化时，还可能出现一定程度的 **Metric Hacking**。例如，如果 EVTR 定义为

$$y^{EVTR} = \mathbb{I}(W > 10s) \tag{23.24}$$

那么一个 12 秒的视频只需要用户观看绝大部分内容即可获得正 Label，而一个更长的视频则需要用户投入更长的绝对观看时间才能获得相同的正 Label。在这种情况下，如果模型和内容分发系统仅仅围绕 EVTR 进行优化，就可能逐渐倾向于推荐更加容易达到固定阈值的短内容，甚至可能进一步诱导内容生产侧通过缩短内容物理 Duration 来获得指标上的优势。

因此，Duration Bias 并不仅仅是一个简单的统计分布偏差问题，而可能进一步传导到模型训练、在线排序以及内容生态等多个环节。尤其是在工业推荐系统中，模型会持续通过在线实验不断放大能够带来指标收益的模式。如果 Label 本身存在结构性偏差，那么模型越强，反而越容易学习并放大这种偏差。因此，时长预估中的一个重要问题并不是简单地让模型更加准确地预测既定 Label，而是需要首先保证 **Label 本身能够较为准确地表达我们真正希望优化的用户观看行为**。

基于 Duration 分桶的经验阈值正是为了缓解这一问题。通过为不同物理 Duration 的视频设置不同的绝对时长阈值，可以在一定程度上校准不同 Duration 内容之间的 Label 难度，使得“达到正 Label”所代表的用户消费深度更加合理。这里需要注意的是，Duration-aware 阈值的目标并不是简单地消除短视频的天然优势，也不是要求不同 Duration 的视频在 EVTR、LVTR 等指标上完全一致。短视频由于内容本身较短，具有更高的完播概率和更容易完成消费的特征，这是正常的用户行为现象。真正需要避免的是：**模型因为 Label 定义过于简单，而将“内容物理时长较短”错误地学习成“用户兴趣更强”**。

从数学形式上来看，上一小节中基于比例的时长阈值可以表示为一个连续函数：

$$T(D) = \alpha D \tag{23.25}$$

而 Duration 分桶策略则可以将这一连续关系进一步扩展为一个分段函数：

$$T(D) = T_k, \qquad D \in B_k \tag{23.26}$$

其中，T_k 表示第 k 个 Duration 分桶对应的经验阈值。与固定比例不同，分桶方法不要求 $T(D)$ 严格满足线性关系，因此可以根据真实用户行为数据对不同 Duration 区间进行更加灵活的校准。例如，可以进一步根据不同 Duration 分桶中用户观看时长的历史分布确定阈值：

$$T_k = Q_\alpha(W \mid D \in B_k) \tag{23.27}$$

其中，$Q_\alpha(\cdot)$ 表示观看时长分布的 α 分位数。此时，Duration 分桶的阈值不再完全依赖人工经验，而可以根据真实用户观看行为进行数据驱动的确定。实际工业系统中，也可以结合业务指标、Label 正负样本比例以及线上 A/B 实验结果对这些时长 Label 的阈值进行进一步调整。

这种基于 Duration 分桶的阈值定义在实际工业推荐系统中非常常见。例如，在一个实际的短视频推荐系统中，可以根据视频物理 Duration 设置不同的 Long view 阈值，并将用户实际观看时长是否超过对应阈值作为 `long_view` Label。其本质上可以表示为：

$$y_{long\_view} = \mathbb{I}(W > T_{LV}(D)) \tag{23.28}$$

其中，W 表示用户实际观看时长，D 表示视频物理 Duration，而 $T_{LV}(D)$ 则是与视频 Duration 相关的 Long view 时长阈值。

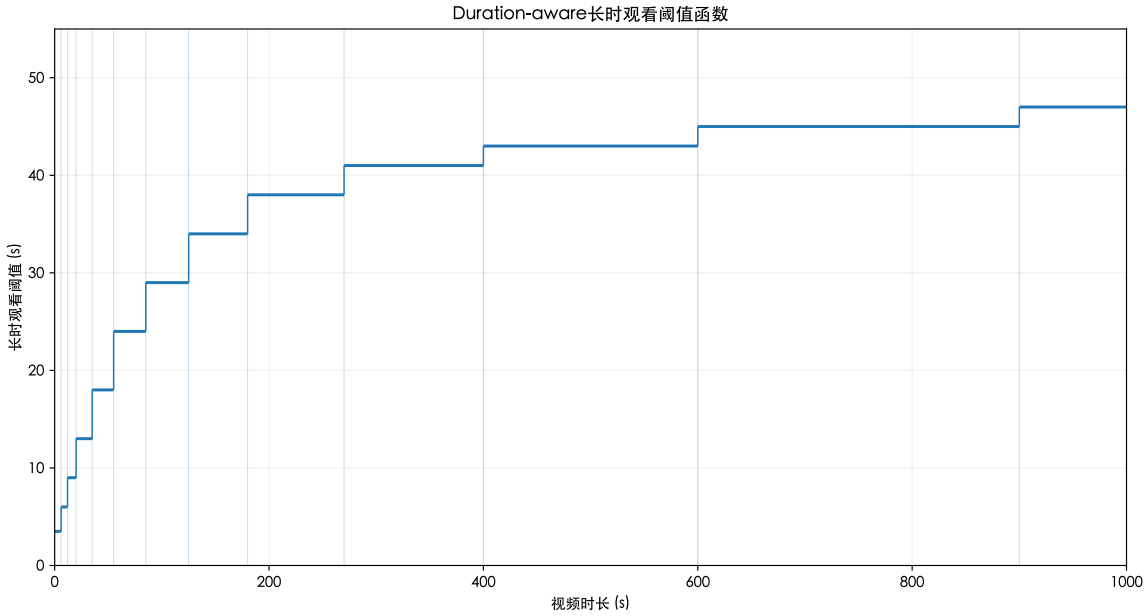

图 23.1: 基于 Duration 分桶的 Long-view 阈值定义示例。

图23.1给出了一个经过脱敏处理的工业配置示例。横轴表示视频物理 Duration，纵轴表示触发 `long_view=1` 所需要达到的最小观看时长。可以看到，整个阈值函数并不是简单的固定常数，也不是严格按照某一个固定比例线性增长，而是由多个 Duration 区间对应的经验阈值共同构成的分段函数。例如，对于 Duration 较短的视频，Long view 阈值从约 3.5 秒逐步增加到 20 秒左右；当视频 Duration 进一步增加到 50 ∼ 200 秒时，阈值继续提升至约 20 ∼ 40 秒；而对于更长的视频，阈值的增长速度逐渐放缓，并最终稳定在约 45 ∼ 50 秒的范围内。这说明在实际工业配置中，随着视频物理 Duration 增加，Long view 所要求的绝对观看时长通常也会提高，但这种增长并不需要严格遵循线性或固定比例关系。

23.1.4 多个时长阈值的联合建模

如果希望模型能够同时刻画不同深度的观看行为，还可以进一步定义多个阈值。例如：

$$T_1 < T_2 < T_3 < T_4 \tag{23.29}$$

则可以分别定义：

$$y_1 = \mathbb{I}(W > T_1), \quad y_2 = \mathbb{I}(W > T_2), \quad y_3 = \mathbb{I}(W > T_3), \quad y_4 = \mathbb{I}(W > T_4) \tag{23.30}$$

然后模型同时预测：

$$p_k = P(W > T_k \mid X) \tag{23.31}$$

并使用多任务损失：

$$\mathcal{L} = \sum_{k=1}^{K} \lambda_k \mathcal{L}_{BCE}(p_k, y_k) \tag{23.32}$$

其中 λ_k 用于控制不同阈值的重要程度。例如，可以设置：

$$(T_1, T_2, T_3) = (5s, 10s, 20s) \tag{23.33}$$

这样模型就可以同时学习用户：

$$P(W > 5), \quad P(W > 10), \quad P(W > 20) \tag{23.34}$$

三个不同层次的观看概率。从概率分布的角度来看，这实际上已经开始逼近对完整观看时长分布进行建模。因为如果有：

$$S(t) = P(W > t) \tag{23.35}$$

那么 $S(t)$ 本身就是观看时长的生存函数（Survival Function）。

因此，基于阈值化的时长 Label 定义并不只是一个简单的二分类技巧，它实际上可以被理解为：

> 通过多个时长阈值上的二分类概率，逐步刻画完整的观看时长分布。

这也是为什么在实际工业系统中，可以同时存在多个时长相关预估目标，而不是只训练一个“观看时长模型”。

23.1.5 基于阈值化时长预估的实现案例

下面给出一个基于 PyTorch 的简单实现。假设我们已经获得了用户的观看时长 `watch time` 和视频物理时长 `duration`，并根据 Duration 分桶定义不同的 EVTR 阈值。

代码 23.1: 基于 Duration 分桶构造 EVTR Label

```
import torch
import torch.nn as nn
# 视频物理 duration
duration = torch.tensor([
    20., 25., 40., 55.,
    80., 100., 180., 240.
])

# 用户实际观看时长
watch_time = torch.tensor([
    8., 12., 15., 35.,
    10., 70., 50., 120.
])
```

```
# duration bucket:
# [0, 30)    -> 5s
# [30, 60)   -> 10s
# [60, 120)  -> 20s
# [120, +inf) -> 30s
threshold = torch.where(duration < 30, torch.tensor(5.),
    torch.where(duration < 60, torch.tensor(10.),
        torch.where(duration < 120, torch.tensor(20.), torch.tensor(30.))
    )
)

# EVTR binary label
label = (watch_time > threshold).float()
print("threshold:", threshold)
print("label:", label)
```

例如，第一个视频的物理时长为 20 秒，因此使用 5 秒作为 EVTR 阈值。如果用户观看了 8 秒，则：

$$8 > 5 \Rightarrow y_{EVTR} = 1 \tag{23.36}$$

而第六个视频的物理时长为 100 秒，对应的 EVTR 阈值为 20 秒。即使用户观看了 10 秒：

$$10 < 20 \Rightarrow y_{EVTR} = 0 \tag{23.37}$$

这样，不同物理 Duration 的视频就能够拥有不同的有效观看定义。

进一步地，如果已经有一个推荐模型输出对应的 Logit，则可以直接使用 BCE Loss：

代码 23.2: 基于 BCEWithLogitsLoss 训练 EVTR 模型

```
# 假设模型输出的 EVTR logit
logits = torch.randn(len(label))
criterion = nn.BCEWithLogitsLoss()
loss = criterion(logits, label)
print("loss =", loss.item())
```

如果需要同时预测 EVTR 和 LVTR，则可以分别构造两个 Label：

代码 23.3: 联合构造 EVTR 和 LVTR Label

```
evtr_threshold = torch.tensor([
    5., 5., 10., 10.,
    20., 20., 30., 30.
])
lvtr_threshold = torch.tensor([
    15., 15., 30., 30.,
    60., 60., 90., 90.
])

evtr_label = (watch_time > evtr_threshold).float()
lvtr_label = (watch_time > lvtr_threshold).float()
evtr_logits = torch.randn(len(label))
lvtr_logits = torch.randn(len(label))
```

```
loss_evtr = nn.functional.binary_cross_entropy_with_logits(evtr_logits, evtr_label)
loss_lvtr = nn.functional.binary_cross_entropy_with_logits(lvtr_logits, lvtr_label)
loss = loss_evtr + loss_lvtr
```

在实际工业模型中，这两个输出通常可以共享底层 User/Item Embedding 和特征提取网络，再通过不同的预测头分别预测 EVTR、LVTR 等目标。例如，设共享底层网络输出为 $H = f(X)$，则可以分别构造 EVTR 和 LVTR 的预测头：

$$p_{EVTR} = \sigma(W_{EVTR}H + b_{EVTR}), \quad p_{LVTR} = \sigma(W_{LVTR}H + b_{LVTR}) \tag{23.38}$$

最终使用加权多任务损失进行联合训练：

$$\mathcal{L} = \lambda_{EVTR}\mathcal{L}_{EVTR} + \lambda_{LVTR}\mathcal{L}_{LVTR} \tag{23.39}$$

其中，λ_{EVTR} 和 λ_{LVTR} 用于控制不同任务对整体模型训练的贡献。

需要注意的是，多个阈值并不是越多越好。如果阈值过多，一方面会增加模型训练和线上推理的成本，另一方面相邻阈值对应的 Label 往往具有较强的相关性，可能导致任务之间存在较大的信息冗余。因此，在实际工业系统中，通常需要结合业务目标、Label 分布、模型复杂度以及线上 A/B 实验结果选择合适的阈值数量和具体阈值。

总体而言，基于阈值化定义的时长预估形成了一条非常典型的工业建模路径：

$$\text{播放时长} \rightarrow \text{Duration-aware 阈值} \rightarrow \text{二值化 Label} \rightarrow \text{分类} \rightarrow P(W > T) \tag{23.40}$$

其核心并不是简单地将一个连续值“转换成二分类”，而是**通过业务目标重新定义观看时长的有效性，并让模型直接学习与线上业务指标对应的概率**。相比于直接追求“把用户究竟观看了多少秒预测得最准确”，工业推荐系统很多时候更加关注一个与线上排序目标直接对应的问题：**这个用户观看这个内容达到某个业务认可的时长阈值的概率究竟有多大？**例如，EVTR 和 LVTR 本质上分别对应不同观看深度下的行为概率，模型可以直接利用这些概率参与后续的多目标融合与排序。

在此基础上，如果希望进一步提高时长建模的粒度，就不应该只判断用户是否超过某一个阈值，而是可以设置多个不同的时长阈值，并同时预测用户超过这些阈值的概率。进一步地，还可以将连续观看时长离散成多个 Duration 分桶，或者直接对完整的观看时长概率分布进行建模，从而使时长预估从单一阈值概率逐步扩展到更加完整的观看时长分布建模。

23.1.6 基于阈值化时长建模的总结

阈值化时长建模的核心思想是：**通过定义与业务目标对应的观看时长阈值，将连续的观看时长问题转化为一个或多个二分类问题，从而让模型直接学习与线上指标对应的概率**。这种方法具有以下几个显著优势：

- **与业务指标直接对应**：阈值化时长建模可以直接将模型输出概率与线上业务指标（如 EVTR、LVTR）对应起来，使得模型优化目标与业务目标高度一致。相比直接回归一个连续的观看时长，模型输出的概率更加容易参与后续的多目标融合、排序以及策略优化。
- **灵活的阈值定义**：通过 Duration-aware 的阈值定义，可以根据视频物理时长的不同，为不同 Duration 的内容设置不同的时长阈值，从而缓解 Duration Bias 问题，使不同 Duration 的视频在达到正 Label 时具有更加接近的业务含义，而不是简单地共享一个固定的绝对时长阈值。
- **多阈值联合建模**：可以同时定义多个时长阈值，并让模型预测用户超过这些阈值的概率，从而逐步逼近完整的观看时长分布建模。不同阈值可以分别对应浅层观看、中等深度观看以及深度观看等不同用户行为。
- **工程实现简单**：阈值化时长建模可以直接使用现有的二分类模型架构和损失函数（如 BCEWithLogitsLoss），无需额外复杂的回归或分布建模方法，便于在工业系统中快速落地。同时，EVTR、LVTR 等多个目标也可以比较自然地通过 Multi-Task Learning 的方式进行联合建模。

从数学的角度理解，无论是 EVTR、LVTR 还是其他基于时长阈值定义的指标，其本质上都是不同 Threshold 下的 Survival Probability。通过阈值化的方式，模型可以直接学习用户观看时长超过某个业务阈值的概率，从而

在推荐系统中更加直接地刻画用户的观看行为和兴趣偏好。具体而言，可以将 Label 看作视频 Duration 与实际观看时长共同决定的函数：

$$y = \mathbb{I}(W > T(D)) = f(D, W) \tag{23.41}$$

其中，$\mathbb{I}(\cdot)$ 表示指示函数，D 表示视频物理 Duration，W 表示用户实际观看时长，而 $T(D)$ 则表示与视频 Duration 相关的时长阈值。进一步地，模型真正需要学习的是给定用户、视频及上下文特征 X 后，用户超过该阈值的概率：

$$p = P(W > T(D) \mid X) \tag{23.42}$$

通过这种方式，模型可以在不同 Duration 下学习到不同的观看行为模式，从而更好地服务于工业推荐系统中的多目标优化。

因此，在具体工业级推荐优化实践中，如何定义合理的 $f(D, W)$ 形式是非常关键的，它直接影响到模型对用户观看行为的理解以及最终线上指标的优化效果。更进一步地，$f(D, W)$ 并不是一个一经确定就长期不变的函数。随着用户观看习惯、内容生态以及视频供给结构不断变化，原有的时长阈值可能逐渐出现 Label 分布漂移或者业务含义弱化的问题。因此，在实际工业系统中，Duration-aware 的时长阈值通常需要进行持续迭代，可以结合周期性数据分析以及线上 A/B 实验对阈值进行重新校准，而不是简单地长期沿用一套固定配置。例如，可以按照季度、半年或者年度等周期重新分析不同 Duration 分桶下的观看时长分布，并结合业务指标变化对 EVTR、LVTR 等阈值进行调整。

当然，Duration 分桶本身也存在一定局限性。首先，分桶的边界会引入人为的不连续性。例如，如果规定：

$$D < 30 \Rightarrow T(D) = 5s \tag{23.43}$$

而：

$$D \geq 30 \Rightarrow T(D) = 10s \tag{23.44}$$

那么一个物理时长为 29.9 秒的视频和一个物理时长为 30.1 秒的视频，其内容形态和用户观看行为可能几乎没有差异，但二者对应的 Label 定义却会发生突然变化。其次，如果 Duration 分桶数量过少，可能无法充分刻画不同物理时长下的用户观看行为；而如果分桶数量过多，又可能导致部分分桶样本量过少，使阈值估计不够稳定，并增加线上规则维护的复杂度。

因此，在工业实践中，Duration 分桶的数量、分桶边界以及每个分桶对应的时长阈值都需要结合实际数据进行调优。更一般地，可以将时长阈值表示为一个 Duration-aware 的阈值函数：

$$T_{EVTR} = f_{EVTR}(D), \qquad T_{LVTR} = f_{LVTR}(D) \tag{23.45}$$

其中，Duration 分桶可以看作对这一连续函数进行工程上的分段近似。与直接学习一个连续函数相比，分桶方式虽然牺牲了一定的连续性，但具有实现简单、解释性强、便于人工调整以及容易与业务规则结合等优势，因此在工业系统中仍然具有较高的实用价值。

除此之外，随着阈值化时长 Label 定义不断优化，我们还可以进一步考虑不同用户群体之间的观看行为差异，对时长阈值进行更加细粒度的个性化调整。例如，不同活跃度、兴趣偏好以及观看习惯的用户，其观看时长分布可能存在明显差异，因此完全相同的 Duration-aware 阈值对不同用户的行为含义也可能并不完全一致。此时，可以进一步将用户特征纳入阈值函数中，即：

$$T_{EVTR} = f_{EVTR}(D, U), \qquad T_{LVTR} = f_{LVTR}(D, U) \tag{23.46}$$

其中，U 表示用户特征向量或者用户历史行为信息。通过这种方式，时长阈值不仅依赖于视频自身的物理 Duration，还可以结合用户自身的观看习惯进行调整，从而进一步提升 Label 定义的个性化程度。

这里介绍一种相对简单的实现方式，即根据用户群体的历史观看行为数据，统计不同用户群体在不同 Duration 分桶下的观看时长分布，并据此为不同用户群体设置不同的 EVTR、LVTR 阈值。例如，对于观看行为更加丰富、历史观看时长整体较高的用户群体，可以根据其实际行为分布设置相对更高的阈值；而对于观看行为较少的用户群体，则可以采用相对保守的阈值设置。需要注意的是，这里的个性化阈值主要是为了让 Label 在不同用户群体中具有更加一致的行为含义，而不是简单地通过提高或降低阈值来“鼓励”用户观看。因此，阈值调整最

终仍然需要以用户行为数据和实际业务指标为依据。

进一步地，可以使用不同用户群体的观看时长分位数来动态确定阈值。例如，可以定义：

$$T_k^{(g)} = Q_\alpha \left(W \mid D \in B_k, U \in G_g \right) \tag{23.47}$$

其中，G_g 表示第 g 个用户群体，B_k 表示第 k 个 Duration 分桶。这样就可以同时考虑用户群体差异和视频 Duration 差异，从而构造更加细粒度的 Duration-aware 阈值。

但这种个性化定义方式在实际落地时也会存在明显问题。对于高活跃用户而言，其历史行为通常比较丰富，因此可以较为稳定地估计其观看时长分布，个性化阈值往往具有更高的可靠性；而对于低活用户、新用户或者行为历史非常稀少的用户，如果完全依赖用户自身的历史行为来定义阈值，则很容易出现统计不稳定甚至无法估计的问题。因此，在实际工业系统中，个性化时长阈值通常不能完全依赖单个用户的历史数据，而需要结合用户活跃度、历史行为数据量以及业务目标进行综合考虑。例如，可以采用用户级别、用户群体级别以及全局级别的分层策略，即：

命题 23.3

对于行为充分的用户使用更加个性化的阈值，对于行为较少的用户则逐渐回退到用户群体或者全局阈值，从而在个性化程度和统计稳定性之间取得平衡。

在工业界的时长预估优化中，这种时长阈值化的定义方式虽然看起来简单，却往往是非常有效的一类优化手段，也经常成为算法工程师在实际业务中的重要优化抓手。学术界在研究推荐系统时长预估问题时，则通常更加关注连续观看时长回归、概率分布建模以及更加复杂的分布预测方法，而较少系统性讨论如何针对具体业务重新设计 EVTR、LVTR 等时长 Label 的定义方式。这也是学术研究与工业落地之间比较典型的 Gap 之一。

学术研究更加关注模型本身是否具有更强的表达能力以及是否能够在公开数据集上取得更好的预测效果，而工业推荐系统最终关注的则是这些预测结果能否真正转化为线上业务收益。因此，在具体的工业级推荐系统时长预估建模中，除了需要关注模型结构和损失函数之外，还应该更加重视时长阈值如何与具体业务场景、内容生态以及用户行为习惯相匹配。一个定义不合理的 Label，即使模型能够对其进行非常准确的预测，也未必意味着推荐系统真正优化了我们希望优化的用户行为。

因此，对于算法工程师而言，一方面需要理解时长预估相关学术论文中的新技术和新方法，另一方面也需要对这些方法进行一定的“祛魅”，理解其在真实工业环境中的收益空间、数据要求以及工程实现成本。相比单纯追求更加复杂的模型结构，有时候重新审视 Label 定义、重新设计 Duration-aware 阈值函数，反而能够以更低的工程成本带来更加直接的线上收益。这也是工业推荐系统中一个非常典型的现象：**模型本身只是优化的一部分，如何定义模型要学习的目标，同样决定了最终能够优化什么。**

总体而言，从上一小节的比例阈值到本小节的 Duration 分桶经验阈值，实际上体现了时长 Label 定义从**简单归一化**向**数据驱动的业务校准**演进的过程。比例阈值具有简单、统一和易解释的优势，但其隐含了观看时长与视频物理 Duration 之间近似线性关系的假设；而 Duration 分桶则放松了这一假设，使不同 Duration 区间能够根据真实用户行为和业务目标采用不同的阈值。在工业推荐系统中，这种对 Label 定义本身进行校准的过程往往非常重要，因为一个存在 Duration Bias 的 Label 即使能够被模型准确预测，也未必能够真正代表我们希望优化的用户行为。

从更一般的角度来看，时长预估的优化并不应该仅仅理解为“选择一个更加先进的模型去预测观看时长”，而应该形成一个完整的优化链路：

$$\text{用户行为} \rightarrow \text{Label 定义} \rightarrow \text{Duration-aware 阈值} \rightarrow \text{预估模型} \rightarrow \text{排序目标} \tag{23.48}$$

其中，Label 定义是整个链路的起点。如果时长 Label 本身无法准确反映业务希望优化的用户行为，那么后续无论采用多么复杂的模型结构，都可能只是在更加准确地预测一个并不理想的目标。因此，在工业级推荐系统中，理解用户的心智与行为、基于对业务场景和痛点的理解定义合理的业务指标，并据此设计合理的时长阈值函数，往往比单纯追求模型结构的复杂度更加重要。

23.2 基于加权二分类的时长建模

除了上一节介绍的基于阈值化定义的时长预估，在工业级推荐系统中，我们还经常需要对用户的**绝对观看时长**进行预估，而不仅仅是判断用户是否超过某一个时长阈值。与 EVTR、LVTR 等阈值化指标不同，这类方法希望模型能够保留更加完整的观看时长信息，并最终得到一个可以直接用于排序和多目标融合的连续时长预估值。

在工业界中，这类时长预估通常会使用 PVTR（Predicted View Time Rate）等指标表示模型的预测结果，对应的真实目标通常记为 VTR（View Time Rate）。不同业务和公司对于具体指标名称的定义可能存在一定差异，但其核心思想基本一致，即：**希望模型不仅判断用户是否观看，还能够进一步刻画用户预计观看多长时间。**

一种经典的时长建模方法来自 YouTube 推荐系统 [1]，通常被称为 Weighted Log Loss（WLL）或者 Weighted Logistic Regression（WLR）。该方法并不直接将观看时长作为连续值进行普通回归，而是将时长预估问题转化为一个特殊的二分类学习问题。相比直接使用 MSE 对观看时长进行回归，WLR 更加充分地利用了推荐系统中大量曝光但未产生观看行为的样本，并通过二分类模型较为稳定的优化形式间接学习连续的观看时长。

23.2.1 Weighted LR 的基本原理与时长反解

与普通 CTR 预估不同，Weighted LR 的核心思想可以概括为：

定义 23.1

对于一次曝光样本，其既贡献正样本损失，也贡献负样本损失，只不过两部分损失具有不同的权重。通过这种方式，可以利用二分类任务训练稳定、易于优化的特点，间接学习连续的观看时长信息。 ♣

假设用户对于某个视频或直播间的真实观看时长为 $t \geq 0$。对于任意一次曝光样本，都贡献一个负样本部分，即 $y = 0$，其对应权重设置为 $w_{neg} = 1$。如果用户产生了观看行为，即 $t > 0$，则额外贡献一个正样本部分，即 $y = 1$，其对应权重设置为 $w_{pos} = t$。从工程实现的角度来看，一个具有观看行为的曝光样本实际上可以被理解为展开成两条训练样本，如表23.3所示。

表 23.3: Weighted LR 时长预估标签及权重定义

Label	Weight	含义
0	1	负样本部分
1	t	正样本部分

因此，对于一个真实观看时长为 t 的曝光样本而言，它同时参与正样本和负样本损失的计算。换句话说，同一个曝光既贡献 $-\log(1-p)$ 对应的负样本损失，又贡献 $-t\log p$ 对应的正样本损失。将两部分损失合并，可以得到单个曝光样本的 Weighted LR 损失：

$$L_i = -t_i \log p_i - \log(1 - p_i) \tag{23.49}$$

其中 $p_i = \sigma(z_i)$ 表示模型输出的 Sigmoid 概率。从整个训练集的角度来看，其优化目标可以写为：

$$L = -\sum_i (t_i \log p_i + \log(1 - p_i)) \tag{23.50}$$

这里需要特别注意，模型输出的 p_i 已经不再表示传统意义上的点击率或者观看概率。虽然从形式上看模型仍然输出一个 $0 \sim 1$ 之间的概率，但这个概率实际上通过正样本权重 t_i 隐式编码了连续的观看时长信息。

下面进一步分析该损失函数为什么能够用于时长预估。对于固定特征条件 x，设该条件下的真实条件期望观看时长为：

$$\mu(x) = E[t|x] \tag{23.51}$$

则对应的条件风险可以写为：

$$R(p|x) = \mu(x) \log p + \log(1 - p) \tag{23.52}$$

对 p 求导：

$$\frac{\partial R}{\partial p} = \frac{\mu(x)}{p} - \frac{1}{1-p} \tag{23.53}$$

令导数为零：

$$\frac{\mu(x)}{p} = \frac{1}{1-p} \tag{23.54}$$

整理可得：

$$p = \frac{\mu(x)}{1+\mu(x)} \tag{23.55}$$

因此：

$$\frac{p}{1-p} = \mu(x) = E[t|x] \tag{23.56}$$

这说明，Weighted LR 在理论最优情况下学习到的并不是传统意义上的观看概率，而是通过概率的 Odds 对条件期望观看时长进行编码。因此，在上线推理阶段，可以通过 Odds 反变换恢复观看时长预估值：

$$\widehat{watch\_time} = \frac{p}{1-p} \tag{23.57}$$

在工业界实践中，这种从模型输出概率反推出观看时长的过程通常可以称为“**时长反解**”。如果模型直接输出 Logit z，由于 Weighted LR 的定义：

$$z = \log \frac{p}{1-p} \tag{23.58}$$

因此可以进一步得到：

$$\widehat{watch\_time} = e^z \tag{23.59}$$

从这个角度来看，WLR 实际上是在使用一个二分类模型作为时长预测器。模型表面上输出的是一个概率 p，但真正用于排序和融合的时长信号则来自该概率对应的 Odds。

需要注意的是，工业系统中的观看时长通常具有非常明显的长尾分布。少量用户可能产生远高于平均水平的观看时长，如果直接使用原始时长作为正样本权重，这些极端长时长样本可能产生过大的梯度，从而影响训练稳定性。因此，实际工程中通常会对时长权重进行截断或者压缩，例如：

$$w = t' = \min(t, T_{max}) \tag{23.60}$$

或者：

$$w = t' = \log(1+t) \tag{23.61}$$

通过这种方式，可以降低极端长时长样本对模型训练的影响。需要指出的是，此时模型严格意义上学习的已经是经过变换后的时长统计量，而不再是原始观看时长的条件期望，因此在设计时需要结合线上业务目标选择合适的权重变换方式。

从代码实现来看，WLR 的核心其实非常简单。假设模型输出一个 Logit：

代码 23.4: Weighted LR 的核心实现

```
logits = model(features)

# logits -> probability
p = torch.sigmoid(logits)

# Weighted LR Loss
loss = (duration * F.softplus(-logits) + F.softplus(logits)).mean()

# Online duration decoding
```

```
pred_duration = p / (1.0 - p + 1e-8)
```

这里使用 `softplus` 而不是显式计算 $\log\sigma(z)$ 和 $\log(1-\sigma(z))$，主要是为了获得更加稳定的数值计算。实际工业模型通常还会进一步加入时长截断、样本权重以及多目标损失等工程处理。

23.2.2 Weighted LR 与直接时长回归的对比

除了 Weighted LR 之外，一个更加直观的时长建模方式是直接将观看时长 t 作为连续 Label，通过 MSE 等回归损失进行训练。例如：

$$L_{MSE} = (\hat{t} - t)^2 \tag{23.62}$$

其中 $\hat{t}$ 为模型直接输出的观看时长。理论上，MSE 的最优预测同样对应于条件期望：

$$\hat{t}^*(x) = E[t|x] \tag{23.63}$$

因此，从最终学习目标来看，WLR 与 MSE 并不是完全不同的问题。二者在理想条件下都希望学习条件期望观看时长：

$$\hat{t}(x) \approx E[t|x] \tag{23.64}$$

真正的差异主要体现在**优化形式、样本利用方式以及工程实现**上。对于 MSE 而言，每个曝光样本只产生一个连续的回归 Label。例如，用户观看 0 秒、3 秒和 100 秒的样本分别对应三个不同的回归目标。模型直接通过预测值与真实时长之间的平方误差进行优化。而对于 WLR 而言，每个曝光样本都会被转化为“负样本 + 正样本”的加权二分类形式。特别是大量观看时长为 0 的曝光，会自然地贡献负样本损失；而产生观看行为的样本则通过观看时长作为正样本权重，将连续时长信息注入分类任务。因此，WLR 充分利用了推荐系统中大量曝光但没有产生有效观看的样本。

表 23.4: Weighted LR 与直接时长回归的对比。

对比维度	Weighted LR	MSE Regression
模型输出	$p \in (0,1)$	$\hat{t} \geq 0$
训练目标	加权二分类	连续值回归
时长恢复	$\hat{t} = p/(1-p)$	直接输出 $\hat{t}$
核心统计量	$E[t\|x]$	$E[t\|x]$
零时长样本	作为负样本参与训练	作为 $t=0$ 的回归样本
长尾影响	时长权重可能放大长时长样本	MSE 对极端误差更加敏感
工程特点	分类框架成熟，便于稳定训练	实现简单、含义直接

可以将两种方法简单概括如表23.4所示，这里有一个非常重要的区别：虽然 MSE 和 WLR 在理论最优解上都可以得到 $E[t|x]$，但它们对训练样本的**梯度作用方式**并不相同。对于 MSE 来说：

$$\frac{\partial L_{MSE}}{\partial \hat{t}} = 2(\hat{t} - t) \tag{23.65}$$

因此，一个远离真实值的长时长样本可能产生非常大的梯度。例如真实观看时长为 300 秒，而模型只预测了 10 秒，那么这个样本对应的误差会被平方放大。而 WLR 则通过观看时长作为正样本权重将长时长样本的信息注入分类损失。虽然长时长仍然会产生更大的梯度，但其作用形式与直接 MSE 的平方误差并不相同，因此可以进一步通过时长截断或者对数压缩控制长尾样本的影响。

这里需要强调的是，通常在工业级推荐系统的粗精排模型或者对应的 LTR 模型建模中，通常都会按照 Multi-Task Learning 的方式同时预测多个目标，例如 CTR、EVTR、LVTR、VTR 等。对于这些目标，WLR 和 MSE 都可以作为其中的一个子任务进行训练，但它们在整体损失函数中的权重以及与其他任务的关系可能会影响最终的优化效果。如果采用 MSE 的损失函数来学习 VTR，而其他的目标仍然采用交叉熵的损失函数，可能会出现 MSE 损失在整体优化中占比过大或者过小的问题，从而影响模型对其他目标的学习效果。而 WLR 由于本身就

是一个加权二分类任务，更容易与其他分类任务进行统一的交叉熵损失加权和优化。

为了更加直观地理解两种方法的差异，可以构造一个简单的合成观看时长数据集，其中用户是否观看以及观看多长时间都与输入特征相关，同时让观看时长具有明显的 0 值和长尾分布。然后分别使用 MSE Regression 和 Weighted LR 进行训练，并通过 Odds 将 WLR 的输出转换为观看时长。具体实现如代码23.5所示。

代码 23.5: MSE 与 WLR 的简单对比实验

```
# MSE
pred = mse_model(x)
loss_mse = F.mse_loss(pred, duration)

# Weighted Logistic
logits = wll_model(x)
loss_wll = (duration * F.softplus(-logits) + F.softplus(logits)).mean()

# WLL duration decoding
p = torch.sigmoid(logits)
pred_duration = p / (1.0 - p + 1e-8)
```

实验中可以进一步记录训练过程中的 MAE，并在输入特征空间上绘制真实条件期望观看时长、MSE 预测值以及 WLR 预测值。图23.2给出了这种实验的一个示例。从图中可以看到，第一张图比较了两种方法训练过程中的 MAE，可以用于观察模型对实际观看时长的拟合速度和稳定性。第二张图进一步固定输入特征 X，直接比较真实条件期望 $E[T|X]$ 与两种模型的预测结果。理想情况下，两种模型都应该逐渐逼近真实的条件期望，这也与前面的理论分析相一致。第三张图则展示真实观看时长以及两种模型预测时长的分布，可以更加直观地观察两种方法对零时长样本以及长尾观看行为的刻画能力。

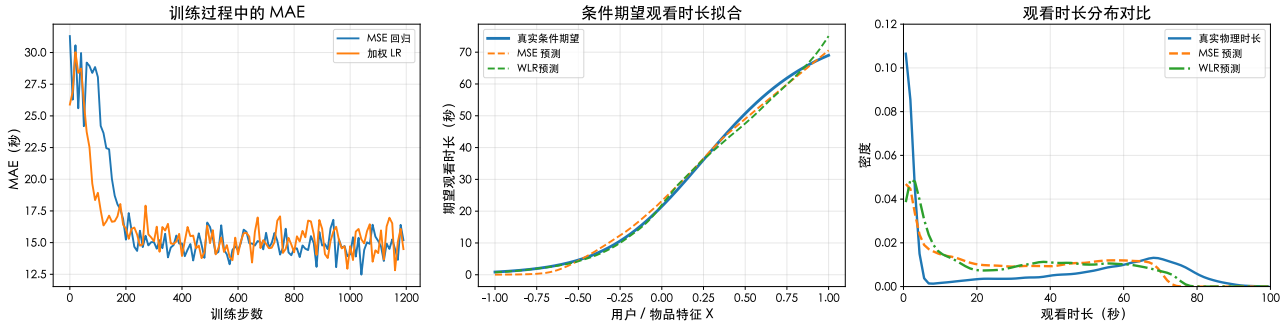

图 23.2: WLR 与 MSE 时长预估的对比示例。

需要特别注意的是，这类实验的目的并不是简单比较某一次实验中 WLR 的 MAE 是否一定低于 MSE。因为两种方法的优化目标和损失尺度本身并不完全一致，实际线上效果还会受到 Label 分布、时长截断、样本权重、模型结构以及排序场景等多种因素影响。更加重要的是理解两种方法背后的建模逻辑：**MSE 直接在时长空间中进行回归，而 WLR 则借助二分类框架，通过概率 Odds 间接编码条件期望观看时长。**

从工业推荐系统的角度来看，WLR 的一个重要优势在于其与成熟的 CTR/二分类建模框架具有较强的兼容性。很多推荐模型本身已经具备稳定的二分类训练、样本加权、在线校准以及多任务学习能力，因此只需要改变 Label 和 Weight 的定义，就可以将已有的 CTR 类模型扩展为时长预估模型。这也是 WLR 在工业界长期被广泛使用的重要原因。

与此同时，MSE 也并非没有优势。它的模型输出具有非常直接的物理含义，不需要额外进行 Odds 反解，并且在一些观看时长分布相对稳定、长尾问题不严重的场景中，直接回归可以获得非常好的效果。因此，在实际工业系统中，并不存在绝对意义上的“WLR 一定优于 MSE”，更合理的方式是根据业务目标、Label 分布以及线上排序需求进行选择。此外，WLR 还有一个非常重要的工程问题，即**时长权重的定义**。如果直接使用原始观看时

长 t 作为正样本权重，那么极端长时长样本可能对模型产生过强的影响。因此，工业系统中通常会进一步设计：

$$w = f(t) \tag{23.66}$$

其中 $f(\cdot)$ 可以是截断、对数压缩或者其他业务相关的非线性函数。例如：

$$w = \min(t, T_{max}) \tag{23.67}$$

也可以采用：

$$w = \log(1 + t) \tag{23.68}$$

不同的权重函数实际上对应了不同的业务偏好。如果希望模型更加关注长时长用户，可以适当提高长时长样本的权重；如果担心极端长时长样本导致模型被少数样本主导，则可以对权重进行截断或者压缩。因此，在实际工业系统中，**Weighted Logistic Regression 中的 Weight 本身也是一个非常重要的优化空间**。

总体而言，Weighted Logistic Regression 可以看作连接“分类建模”和“连续时长建模”的一个重要桥梁。它并没有直接使用复杂的连续分布模型，而是借助成熟的二分类模型框架，通过观看时长对正样本进行加权，再利用 Odds 反解得到最终的时长预估：

$$t \to w(t) \to \text{Weighted Logistic} \to p \to \frac{p}{1-p} \to \widehat{t} \tag{23.69}$$

因此，从工业推荐系统的角度来看，WLR 的价值并不仅仅在于它能够预测观看时长，更重要的是它在**模型稳定性、工程复用以及时长预估能力之间取得了较好的平衡**。相比直接对原始观看时长进行 MSE 回归，这种方式能够更加自然地利用推荐系统中海量的曝光样本，并且可以通过时长权重函数进一步控制长尾行为。当然，WLR 本质上仍然是在学习条件期望 $E[t|x]$。当业务真正关心的不再只是“平均能够观看多长时间”，而是希望进一步刻画观看时长的完整分布，例如区分用户观看 3 秒、10 秒、30 秒和 100 秒的不同概率时，仅仅预测一个条件期望就可能存在信息损失。此时，就需要进一步考虑将连续观看时长进行离散化，或者直接对完整的观看时长分布进行建模，这也将引出后续更加细粒度的时长分布建模方法。

23.3 基于离散化分桶的时长建模

除了直接对连续时长进行回归之外，近年来工业推荐系统中也广泛采用连续时长离散化（Discretization）的方式进行建模。其核心思想如下：

定义 23.2

离散化分桶建模是将原本连续的时长预测问题转化为离散概率分布预测问题：首先根据时长分布将连续取值划分为若干个时间桶，然后通过分类模型预测用户观看时长落入各个桶的概率，最后再根据预测概率分布恢复连续时长。

相比直接使用 MSE 对连续时长进行回归，离散化建模能够将具有明显长尾特征的连续变量转化为相对稳定的分类问题，同时还可以进一步利用概率分布刻画用户观看时长的不确定性。离散化分桶建模的具体做法相对比较简单。假设用户观看时长为 t，其取值范围为：$t \in [0, T_{\max}]$。首先将时长划分为 N 个桶：

$$B_1, B_2, \cdots, B_N \tag{23.70}$$

常见的分桶方式包括等距分桶（Equal Width Binning）、等频分桶（Equal Frequency Binning）、分位数分桶（Quantile Binning）以及对数分桶（Logarithmic Binning）等。其中，推荐系统实践中通常更加关注基于数据分布的非均匀分桶，例如分位数分桶，因为观看时长往往具有明显的长尾特征，大量样本集中在较短时长区域，而长时长区域的样本逐渐稀疏。如果采用完全等距的时间桶，则短时长区域的有效样本可能过于集中，而长时长区域又容易出现大量样本稀疏的桶。因此，实际工业系统通常会结合数据分布与业务经验，在短时长区域采用更加密集的分桶，在较长时长区域逐渐扩大桶宽，并对极端长时长进行截断或者单独设置尾部桶。

23.3.1 One-Hot 与高斯 Soft Label 时长建模

最直接的离散化方法是将时长分桶后转换为一个普通的多分类问题。假设真实观看时长 t 落入第 k 个时间桶，则其 One-Hot 标签可以表示为：

$$y_i = \mathbb{I}(i = k), \quad i = 1, 2, \cdots, N \tag{23.71}$$

模型经过 Softmax 后输出每个时间桶对应的预测概率：

$$\hat{\mathbf{p}} = (\hat{p}_1, \hat{p}_2, \cdots, \hat{p}_N) \tag{23.72}$$

其中：

$$\hat{p}_i = P(B_i|x) \tag{23.73}$$

训练时采用标准交叉熵损失：

$$\mathcal{L}_{\text{OneHot}} = -\sum_{i=1}^{N} y_i \log \hat{p}_i \tag{23.74}$$

由于 One-Hot 标签只有真实桶对应的位置为 1，其他位置均为 0，因此模型的训练目标非常明确，即尽可能将概率集中到真实时长所在的桶。

在完成预测后，可以使用每个桶对应的代表时长 c_i 对预测概率进行加权求和，从而恢复连续时长：

$$\hat{t} = \sum_{i=1}^{N} \hat{p}_i c_i \tag{23.75}$$

其中 c_i 可以取第 i 个桶的中心值、桶内样本均值或者其他具有代表性的统计量。当分桶足够细时，这种方式可以较好地逼近连续时长预测。

One-Hot 方法虽然简单，但存在一个比较明显的问题，即没有显式利用时长本身具有的连续性和有序关系。例如，假设真实观看时长为 120 秒附近的一个时间桶，那么预测到 119 秒附近的桶显然比预测到 10 秒附近的桶更加合理。然而，对于标准的 One-Hot 分类损失而言，这两种错误都只是“预测到了错误的类别”，并不会因为后者距离真实时长更远而受到更大的惩罚。此外，如果真实时长恰好位于两个时间桶的边界附近，那么只需要发生非常小的时长变化，就可能使标签从一个 One-Hot 向量突然变成另一个完全不同的 One-Hot 向量。这种标签的不连续性与观看时长本身的连续属性并不完全匹配。

因此，可以进一步将 One-Hot 标签扩展为 Soft Label，即不再将全部概率质量集中在真实桶，而是将一定概率分配给真实桶附近的其他时间桶，并且距离真实位置越远，分配到的概率越低。最常见的一种方式就是采用高斯分布进行平滑。假设真实时长对应离散位置 k，则第 i 个桶对应的未归一化权重可以定义为：

$$\tilde{y}_i = \exp\left(-\frac{(i-k)^2}{2\sigma^2}\right) \tag{23.76}$$

其中 σ 为高斯平滑参数。σ 越小，概率越集中在真实时长附近，Soft Label 越接近 One-Hot Label；σ 越大，概率向周围时间桶扩散得越明显。

进一步进行归一化，可以得到最终的高斯 Soft Label：

$$y_i = \frac{\tilde{y}_i}{\sum_{j=1}^{N} \tilde{y}_j} \tag{23.77}$$

从而满足：

$$\sum_{i=1}^{N} y_i = 1 \tag{23.78}$$

此时，一个真实时长对应的不再是单一类别，而是围绕真实时长位置形成的局部概率分布。模型训练时仍然可以使用交叉熵，只是原来的 One-Hot 标签被替换成了 Soft Label：

$$\mathcal{L}_{\text{Gaussian}} = -\sum_{i=1}^{N} y_i \log \hat{p}_i \tag{23.79}$$

因此，高斯 Soft Label 的本质并不是改变 Softmax 的输出形式，而是改变监督信号的构造方式。One-Hot 监督要

求模型将概率全部集中到一个离散桶，而高斯 Soft Label 则要求模型学习一个以真实时长为中心、随距离逐渐衰减的局部概率分布。这样能够显式地将“预测距离真实值的远近”引入训练过程。

对于上述 Label 构造过程，可以直接使用如下代码实现。One-Hot Label 首先将连续时长转换为整数桶编号，并通过 `F.one_hot` 构造独热向量；高斯 Soft Label 则首先构造所有桶的位置，然后计算桶位置与真实时长之间的距离，最后利用高斯概率密度进行归一化：

代码 23.6: One-Hot 与高斯 Soft Label 时长标签构造代码。

```
def onehot_label(duration, num_bins):
    return F.one_hot(duration.long().clamp(0, num_bins - 1), num_bins).float()

def gaussian_soft_label(duration, num_bins, sigma):
    bins = torch.arange(num_bins, dtype=torch.float32, device=duration.device)
    diff = bins.unsqueeze(0) - duration.unsqueeze(1)
    softy = torch.distributions.Normal(0.0, sigma).log_prob(diff).exp()
    return softy / softy.sum(-1, keepdim=True)
```

从代码实现可以看到，高斯 Soft Label 并不是简单地对 One-Hot 向量做数值上的平滑，而是直接在整个离散时间轴上构造一个高斯概率分布。对于一个真实时长 t，距离 t 越近的时间桶，其高斯密度越大，因此获得的监督权重也越高；距离越远的时间桶则获得越小的权重。这样就自然地把连续时长的局部邻近关系引入到了原本的分类任务中。

为了更直观地观察两种 Label 对模型训练的影响，可以构造一个简单的受控实验。实验中设置 400 个离散时间桶，并使用一个简单的 MLP 作为时长预测模型。模型首先通过两层全连接网络得到隐向量，然后通过一个线性层输出 400 个时间桶对应的 Logits，并经过 Softmax 得到概率分布。最终预测时长仍然采用各个桶位置与预测概率的加权和：

$$\hat{t} = \sum_{i=1}^{400} \hat{p}_i i \tag{23.80}$$

对应的模型结构可以简化表示为：

代码 23.7: One-Hot 与高斯 Soft Label 对比实验中的时长预测模型。

```
class DurationModel(nn.Module):
    def __init__(self, input_dim, num_bins):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(input_dim, 256),
            nn.ReLU(),
            nn.Linear(256, 128),
            nn.ReLU()
        )
        self.head = nn.Linear(128, num_bins)

    def forward(self, x):
        logits = self.head(self.net(x))
        probs = F.softmax(logits, dim=-1)
        pred = (probs * BIN_VALS.to(x.device)).sum(-1, keepdim=True)
        return logits, probs, pred
```

实验分别使用 One-Hot 和高斯 Soft Label 训练两个结构完全相同的模型，仅改变 Label 与 Loss 的构造方式。

One-Hot 模型直接使用标准交叉熵，而高斯模型则使用高斯 Soft Label 与模型预测分布之间的交叉熵：

代码 23.8: One-Hot 与高斯 Soft Label 对应的训练损失。

```
def compute_loss(method, logits, duration_labels):
    if method == "onehot":
        return F.cross_entropy(logits, duration_labels.long().clamp(0, BUK_NUMS - 1))
    else:
        soft = gaussian_soft_label(duration_labels, BUK_NUMS, SIGMA_TRAIN)
        return -(soft * F.log_softmax(logits, -1)).sum(-1).mean()
```

为了使实验同时包含短时长和长时长样本，示例代码构造了两个具有不同中心位置的 Gaussian 数据分布，其中一部分样本集中在约 30 秒附近，另一部分样本集中在约 180 秒附近，并将时长限制在 $[0, 400]$ 的范围内：

代码 23.9: One-Hot 与 Gaussian Soft Label 对比实验中的示例时长分布。

```
N = 2000
INPUT_DIM = 128
BUK_NUMS = 400
X_all = torch.randn(N, INPUT_DIM)
labels_all = torch.cat([torch.randn(N // 2) * 10 + 30, torch.randn(N // 2) * 30 + 180,])
labels_all = labels_all.clamp(0, BUK_NUMS - 1)

dataset = TensorDataset(X_all, labels_all)
dataloader = DataLoader(dataset, batch_size=BATCH, shuffle=True)
```

需要特别说明的是，这里的数据分布是为了直观展示两种 Label 和训练方式差异而构造的受控实验，并不代表真实线上观看时长分布。因此，后续图中的 Loss、MAE 以及预测分布主要用于解释模型行为，而不能直接理解为线上业务效果。

实验结果如图 23.3所示。图的第一行分别从 Loss、MAE 和 Label 形状三个角度比较 One-Hot 与高斯 Soft Label。Loss 曲线可以反映两种监督方式在训练过程中的优化行为，而 MAE 则将模型最终的概率分布进一步映射为连续时长后进行评价。需要注意的是，由于两种方法优化的目标分布不同，因此不能简单根据 Loss 数值的大小判断哪种方法更优，更有意义的是观察两种监督方式最终形成的概率分布以及连续时长预测结果。

图中第三个子图进一步给出了真实时长为 120 秒时的 Label 分布。One-Hot 方法将全部概率集中在 120 秒对应的单一 Bin 上，而高斯 Soft Label 则以 120 秒为中心向左右相邻 Bin 扩散，并且距离真实值越远，概率越低。图中采用 $\sigma = 2.0$ 的高斯分布，因此概率主要集中在真实值附近的局部区域。图的第二、三行进一步展示了固定样本在不同训练阶段的预测概率分布。One-Hot 模型在 step=50、200 和 500 时的预测分布会逐渐形成较明显的局部峰值，而高斯模型的预测分布则相对更加平滑。图中还同时标记了样本真实时长与模型预测时长，从而可以观察模型概率质量从较分散状态逐渐向特定时长区域聚集的过程。在示例实验中，One-Hot 模型在不同训练阶段对应的预测结果分别约为 101.4、94.2 和 100.8 秒，而高斯模型对应约为 112.0、103.0 和 118.1 秒。

这一结果说明，高斯 Soft Label 最重要的作用并不是简单地让预测值始终更加接近真实值，而是使模型学习到更加平滑的时长概率分布。对于连续时长而言，这种分布式监督能够避免模型将所有概率强行压缩到单一离散桶，同时保留真实值附近多个桶之间的相对关系。因此，高斯 Soft Label 可以理解为在离散分类框架中重新引入连续目标的局部结构。

从工业推荐系统的角度来看，这种离散化建模还可以进一步结合实际数据分布进行优化。例如，在播放时长建模中，可以将时长划分为约 600 个箱，在较小的播放时长区域采用更加密集的分位点进行划分，而在较大的播放时长区域逐渐采用更加稀疏的分位点，并对特别长的视频进行截断。这样既能够保证短时长区域具有足够高的预测分辨率，又能够避免在长尾区域使用过多的离散类别。

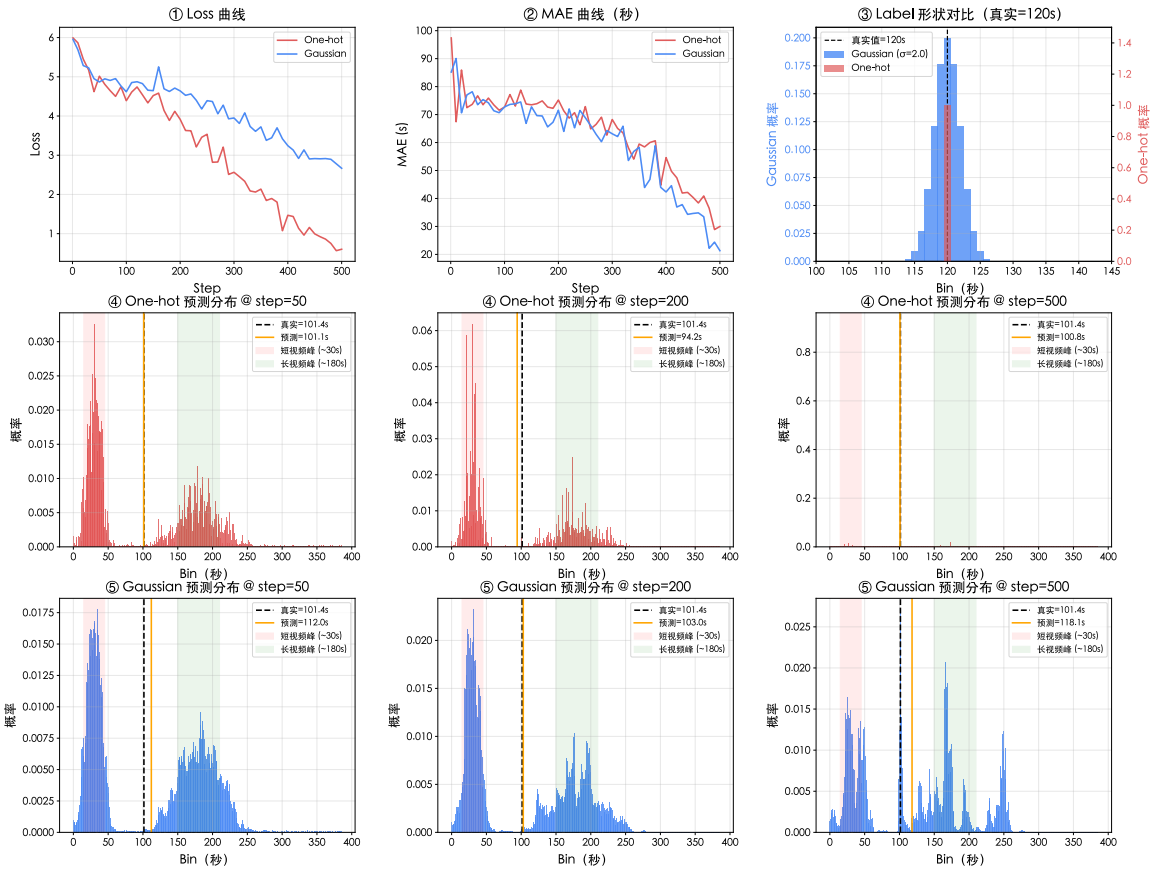

图 23.3: One-Hot 时长预估与 Gaussian 平滑时长预估对比。

23.3.2 自适应分桶与分位点 Soft Label 建模

在高斯 Soft Label 这种时长 Label 构造方法的基础上，还可以进一步利用真实数据的分布结构设计自适应的离散化方案。固定时间桶虽然能够通过高斯平滑缓解 One-Hot 的硬标签问题，但仍然存在一个潜在限制：不同时间区域的数据密度可能存在非常明显的差异。如果所有区域采用相同的离散粒度，那么在样本密集区域可能仍然不够精细，而在样本稀疏区域又会产生大量利用率较低的离散类别。

因此，可以首先根据数据分布对时长进行粗粒度的自适应分桶，在样本密集区域划分更加细的时间区间，而在样本稀疏区域使用更加宽的时间区间；然后，在每一个粗粒度时间桶内部进一步计算若干分位点，并将这些分位点作为该区域内部更加精细的离散坐标。与固定间隔的时间桶相比，这种方式实际上构造了一套由数据分布决定的非均匀时间坐标。

假设真实时长为 t，首先根据其所在位置确定粗粒度时间桶：

$$d = g(t), \quad d \in \{1, 2, \cdots, D\} \tag{23.81}$$

对于第 d 个时间桶，进一步计算其内部的 K_d 个时长分位点：

$$Q^{(d)} = \left\{ q_1^{(d)}, q_2^{(d)}, \cdots, q_{K_d}^{(d)} \right\} \tag{23.82}$$

由于分位点由桶内真实数据分布计算得到，因此数据密集区域中的分位点会更加密集，而数据稀疏区域中的分位点会更加稀疏。这样就能够同时利用“粗粒度分桶”来解决全局长尾问题，并利用“桶内分位点”来提高局部时长分辨率。

在确定当前样本对应的 Duration 分桶后，可以进一步沿用高斯 Soft Label 的思想，将真实时长与该桶内部各个分位点之间的距离转换为概率权重。对于第 i 个分位点，其未归一化权重可以统一写为：

$$\tilde{y}_i = K(t, q_i^{(d)}) \tag{23.83}$$

其中 $K(\cdot,\cdot)$ 为距离衰减核函数。最自然的选择仍然是高斯核：

$$K_{\text{Gaussian}}(t, q_i^{(d)}) = \exp\left(-\frac{(t-q_i^{(d)})^2}{2\sigma^2}\right) \tag{23.84}$$

当然也可以使用 Laplacian 核：

$$K_{\text{Laplace}}(t, q_i^{(d)}) = \exp\left(-\frac{|t-q_i^{(d)}|}{s}\right) \tag{23.85}$$

其中 σ 和 s 分别控制两种核函数的平滑程度。高斯核按照距离的平方进行衰减，因此远距离位置的概率下降更加迅速；Laplacian 核则按照绝对距离进行指数衰减，其尾部相对更加平缓。在观看时长具有明显长尾特征的场景中，可以根据实际数据分布选择更加合适的核函数。

在得到未归一化权重之后，需要对当前桶内所有分位点进行归一化：

$$y_i = \frac{\tilde{y}_i}{\sum_{j=1}^{K_d} \tilde{y}_j} \tag{23.86}$$

最终得到一个只定义在当前 Duration 分桶内的 Soft Label：

$$\mathbf{y}^{(d)} = (y_1, y_2, \cdots, y_{K_d}), \quad \sum_{i=1}^{K_d} y_i = 1 \tag{23.87}$$

此时，一个样本对应的监督信号不再是某个固定时间桶上的 One-Hot 标签，而是在当前数据分布自适应产生的分位点上形成一个局部概率分布。真实时长越接近某个分位点，该分位点获得的概率越高；距离越远，概率则逐渐降低。

在模型训练时仍然可以采用 Softmax + 交叉熵的方式进行优化：

$$\mathcal{L}_{\text{dist}} = -\sum_{i=1}^{K_d} y_i \log \hat{p}_i \tag{23.88}$$

因此，从模型结构角度看，该方法并不需要引入特别复杂的网络结构，其主要变化发生在 Label 构造和预测结果解码阶段。这也是这种方法比较适合工业推荐系统的原因之一：可以在复用已有多分类模型结构的基础上，通过重新设计离散化方式和监督分布来提高时长建模能力。

在模型推理阶段，最简单的解码方式仍然是直接计算分位点的概率加权平均：

$$\hat{t} = \sum_{i=1}^{K_d} \hat{p}_i q_i^{(d)} \tag{23.89}$$

不过，由于不同分位点之间的间隔并不相同，直接进行概率加权平均并不能充分利用非均匀分位点的时间间隔信息。因此，还可以进一步利用预测概率分布的累积分布函数（Cumulative Distribution Function，CDF）进行解码。假设当前桶中的分位点满足：

$$q_1^{(d)} < q_2^{(d)} < \cdots < q_{K_d}^{(d)} \tag{23.90}$$

模型预测概率对应的离散 CDF 为：

$$F_i = \sum_{j=1}^{i} \hat{p}_j \tag{23.91}$$

我们可以进一步定义离散的生存概率，即：

$$S_i = 1 - F_i \tag{23.92}$$

其中 S_i 可以理解为预测时长超过第 i 个分位点的概率。对于相邻两个分位点，其时间间隔为：

$$\Delta_i = q_{i+1}^{(d)} - q_i^{(d)} \tag{23.93}$$

于是，我们可以利用生存函数积分的思想对连续时长的期望进行离散近似：

$$\hat{t} = q_1^{(d)} + \sum_{i=1}^{K_d-1} \left(q_{i+1}^{(d)} - q_i^{(d)}\right)(1 - F_i) \tag{23.94}$$

这种近似方式的理论基础是，对于非负随机变量 T，其期望可以表示为生存函数的积分：

$$\mathbb{E}[T] = \int_0^{\infty} P(T > x)\mathrm{d}x \tag{23.95}$$

在这里模型并不存在连续形式的 $S(x)$，而只有定义在若干分位点上的离散概率分布，因此可以使用相邻分位点构成的区间对上述积分进行离散近似。与简单地使用桶中心进行加权相比，这种基于 CDF 和生存函数的解码方式能够显式利用不同分位点之间的非均匀间隔，因此更加适合自适应分桶场景。

从整体上看，时长离散化建模实际上可以形成一条逐步演进的技术路线。最基础的方法是将连续时长划分为固定时间桶，并使用 One-Hot 标签进行普通多分类；进一步可以使用高斯 Soft Label 将真实值附近的多个桶纳入监督，使模型学习更加平滑的局部概率分布；在此基础上，还可以根据真实时长的数据分布进行自适应分桶，并在每一个粗粒度时间桶内部进一步建立分位点概率分布，从而同时解决长尾分布下的全局离散化问题和局部时长分辨率问题。对于观看时长、停留时长以及会话时长等典型的非负长尾连续目标而言，这种从“点回归”到“离散概率分布”，再到“分布自适应的局部概率建模”的演进，也是工业推荐系统中较为实用的一类时长建模思路。

23.4 经典时长预估建模方法

前几个章节主要介绍了工业推荐系统中几类典型的时长建模方法，包括基于阈值化定义的时长建模、基于加权二分类的时长建模以及基于离散化分桶的时长建模。这些方法的共同特点是从不同角度对原始观看时长 Label 进行重新定义，使模型更加稳定地学习用户的观看行为。

除了上述方法之外，工业界和学术界还提出了许多针对观看时长绝对值预估的经典方法。例如，快手提出的 D2Q（Duration-Deconfounded Quantile-based Watch-time Prediction）[6]，从因果推断和分位数建模的角度对 Duration Bias 进行处理；CREAD、TPM 等方法则进一步从不同角度对观看时长的分布特征、长尾问题等进行建模。本节将对这些具有代表性的时长预估方法进行介绍。

23.4.1 D2Q 算法

近年来，快手科技提出的 D2Q（**D**uration-**D**econfounded **Q**uantile-based Watch-time Prediction Framework）方法 [6]，进一步从因果推断和分位数建模的角度对观看时长预估问题进行优化。与直接拟合原始观看时长不同，D2Q 的核心思想如下：

定义 23.3

按照视频物理 Duration 对训练数据进行分组，并在每个 Duration Group 内将观看时长转换为分位数 Label，再利用一个共享的模型预测该分位数，最后通过对应分组的经验分布将分位数反变换为实际观看时长。

传统时长回归模型通常直接学习用户观看时长的条件期望：

$$E[W|U, V, D] \tag{23.96}$$

其中 U 表示用户特征，V 表示视频特征，D 表示视频的物理 Duration，W 表示用户实际观看时长。这样做虽然比较直接，但在工业推荐系统中，视频的 Duration 与用户的 Watch Time 通常存在非常强的相关性。对于用户兴趣相近的两个视频，物理时长更长的视频通常更容易获得更长的观看时长，因此时长预估模型很容易学习到**“Duration 越长，Watch Time 越长”**的相关关系。

与此同时，Duration 不仅影响用户最终的观看时长，还可能影响视频在推荐系统中的曝光分布。由于推荐系统通常希望最大化用户观看时长，长视频天然更容易获得较高的 Watch Time，从而在训练数据中获得更大的样本权重和更多的曝光机会，进一步导致模型更加偏好长视频。这种由推荐系统反馈循环不断放大的现象，也就是工业推荐系统中常见的 Duration Bias。

如图23.4所示，从因果关系的角度来看，可以将用户、视频、物理 Duration 和观看时长之间的关系简化表示为：

$$U, V \rightarrow W, \qquad D \rightarrow W, \qquad D \rightarrow V \rightarrow W \tag{23.97}$$

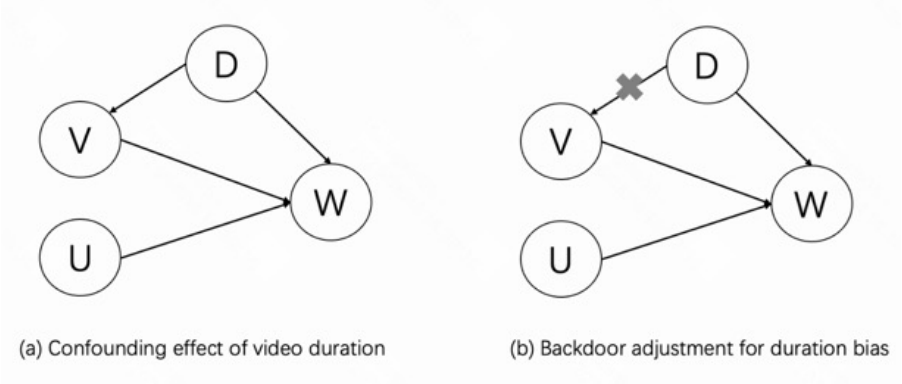

图 23.4: Duration Bias 示意图。

其中，U 和 V 主要刻画用户兴趣对观看时长的影响，$D \to W$ 表示视频物理时长本身对观看时长的直接影响，而 $D \to V \to W$ 则反映 Duration 对视频曝光分布产生影响后进一步影响 Watch Time 的路径。前者属于推荐系统真正需要保留的时长效应（Duration Effect），而后者则可能导致视频曝光分布产生偏置，并进一步形成反馈循环。D2Q 的基本思路就是通过基于 Duration 的数据划分（Duration-based Data Splitting，DDS），尽可能地削弱 Duration 对视频曝光分布产生的偏置，同时保留 Duration 对真实观看时长的合理影响。

具体而言，首先根据训练数据中的视频 Duration 分布计算经验分位数，将所有视频划分为 M 个 Duration 分组：

$$\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_M\} \tag{23.98}$$

其中，不同分组在训练数据中具有近似相等的样本量。与直接按照固定秒数进行分桶相比，这种基于 Duration 分位数的分桶方式能够避免某些 Duration 区间样本过多、而其他区间样本过少的问题。

对于每一个 Duration 分组 $\mathcal{D}_k$，D2Q 算法不再直接学习原始的 Watch Time，而是统计该分组内部的 Watch Time 累积分布函数（Cumulative Distribution Function，CDF），即：

$$\hat{\Phi}_k(w) = P(W \le w \mid D \in \mathcal{D}_k) \tag{23.99}$$

随后，对于一个训练样本 (u_i, v_i, d_i, w_i)，首先确定其所属的 Duration 分组 $\mathcal{D}_{k_i}$，再利用该分组的经验 CDF 将原始观看时长转换为分位数 Label：

$$q_i = \hat{\Phi}_{k_i}(w_i) \tag{23.100}$$

其中，$q_i \in [0, 1]$ 表示该样本在对应 Duration 分组内的相对观看时长位置。例如，对于一个物理 Duration 较短的视频，如果用户观看时长为 8 秒，但该时长已经处于该 Duration 分组的 80% 分位点，那么其 Label 就是 $q = 0.8$；对于一个更长的视频，即使用户观看了 20 秒，只要其在对应 Duration 分组中也处于 80% 分位点，其 Label 同样可以定义为 $q = 0.8$。

这样一来，模型学习的目标就从原始 Watch Time 转换成了 Duration-aware 的相对观看水平。D2Q 算法使用一个共享的模型学习：

$$\hat{q} = h(U, V, D) \tag{23.101}$$

其中，h 是所有 Duration 分组共享的播放时长分位点预估模型（Watch-time Quantile Prediction Model）。D2Q 算法的论文中采用 MSE 对分位数 Label 进行学习：

$$L_{D2Q} = \frac{1}{N} \sum_{i=1}^{N} (q_i - \hat{q}_i)^2 \tag{23.102}$$

这里需要特别注意，D2Q 中的“Quantile-based”并不意味着必须使用传统的分位数回归（Quantile Regression）中的分位数损失（Quantile Loss）。其核心在于将原始观看时长转换为不同 Duration 分组内的分位数 Label，而论文的具体实现采用的是 MSE 损失来学习这一归一化后的分位数 Label。

从模型的角度来看，这种数据处理方式解决了一个比较重要的工程矛盾。如果直接针对不同 Duration 分组

分别训练独立的 Watch Time 模型，那么确实可以减少不同 Duration 区间之间的干扰，但模型参数规模会随着 Duration 分组数量增加而线性增长，在工业推荐系统中并不现实。反过来，如果所有 Duration 分组直接共享同一个模型并使用原始 Watch Time 作为 Label，那么长 Duration、长 Watch Time 样本仍然会主导模型的梯度更新，时长去混杂（Duration Deconfounding）的效果就会明显减弱。

D2Q 算法通过将不同 Duration 分组中的 Watch Time 转换到统一的分位数空间，使不同 Duration 分组可以共享同一套模型参数。模型学习的是“用户在当前 Duration 分组中处于什么样的观看水平”，而不是直接学习跨 Duration 分组的绝对观看秒数，因此能够在一定程度上减少不同 Duration 样本之间的训练干扰。

在推理阶段，首先根据待预测视频的物理 Duration 找到对应的 Duration 分组 $\mathcal{D}_{k_0}$，然后利用共享模型预测其 Watch-time 的分位数：

$$\hat{q}_0 = h(u_0, v_0, d_0) \tag{23.103}$$

最后通过该 Duration 分位数对应的经验分位数函数将预测结果再反变换回原始 Watch Time 空间：

$$\hat{w}_0 = \hat{\Phi}_{k_0}^{-1}(\hat{q}_0) \tag{23.104}$$

其中，$\hat{\Phi}_{k_0}^{-1}$ 表示第 k_0 个 Duration 分组对应的经验分位数函数。整个过程可以概括为“**Duration 分组 → 组内分位数 Label → 共享模型预测 → 分位数反变换**”。

需要注意的是，由于模型需要区分不同 Duration 分组中具有相同分位数 Label 的样本，因此 Duration 信息仍然需要作为模型输入的一部分。否则，例如两个不同 Duration 分组的样本都具有 $q = 0.8$ 的 Label，模型仅根据用户和视频特征进行预测时，很难判断最终应该映射到哪个 Watch Time 区间。在此基础上，还可以进一步增加独立的 Duration 塔或 Duration-aware 的特征提取模块，使模型能够更加充分地利用视频物理时长信息，这也是论文中进一步提出 Res-D2Q 等改进方法的基础，如图23.5所示。

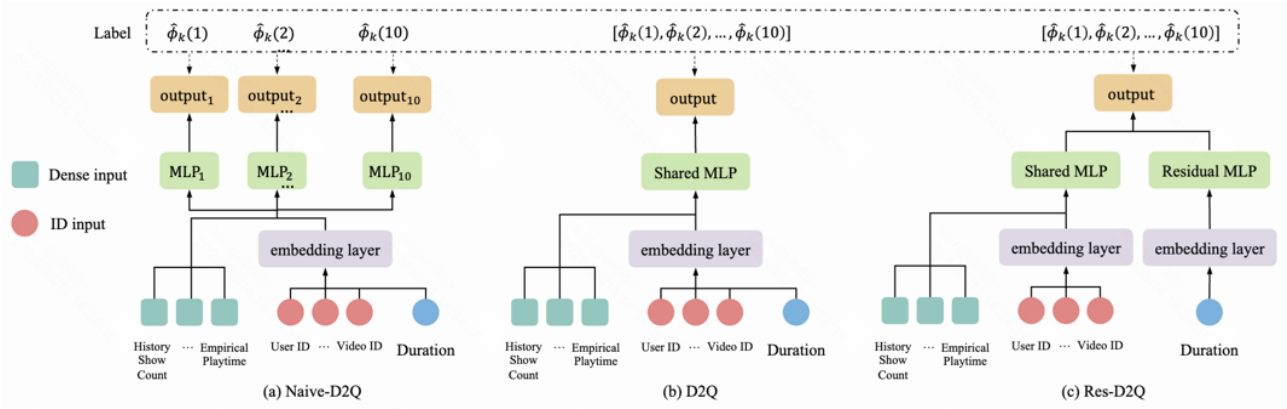

图 23.5: D2Q 及其变种方法的网络结构对比。

从工程实践来看，D2Q 算法还有一个需要重点关注的问题，即 Duration 分组的数量并不是越多越好。当 Duration 分组较少时，不同 Duration 区间内部的分布仍然存在一定差异，Duration Bias 的消除可能不够充分；随着分组数量增加，不同区间之间的差异能够被更加精细地刻画。但与此同时，每个 Group 中的样本量也会逐渐减少，使经验 CDF $\hat{\Phi}_k$ 的估计误差增大。因此，D2Q 算法的实际效果通常存在一个折中点：Duration 分组过少时去偏效果不足，分组过多时又容易受到样本量和分布估计误差的影响。

总体而言，D2Q 算法并没有通过设计非常复杂的网络结构来解决时长预估问题，而是从 **Label 定义和数据分布**入手，将原始观看时长转换为 Duration-aware 分位数，并通过共享模型实现不同 Duration 区间之间的参数共享。对于工业推荐系统而言，这种思路具有比较强的工程价值：一方面能够缓解长 Duration、长 Watch Time 样本对模型训练的主导作用，另一方面又避免了针对不同 Duration 区间分别训练独立模型所带来的模型规模和线上部署成本。因此，D2Q 算法的核心启发并不只是“使用分位数进行时长预测”，更重要的是通过重新定义时长 Label，将 Duration Bias 的处理融入到数据构造和模型训练过程中。

23.4.2 DVR 算法

DVR（Debiased Video Recommendation）算法源自 MM 2022 会议论文《DVR: Micro-Video Recommendation Optimizing Watch-Time-Gain under Duration Bias》[9]。与 D2Q 从损失函数层面对时长偏差进行建模不同，DVR 从评价指标、预测目标和模型结构三个层面同时进行去偏，其核心思想是：视频时长本身会对观看时长产生强烈影响，因此不能直接将 Watch Time（WT）作为用户兴趣的衡量指标，而应该在相似视频时长条件下比较用户的相对观看表现。基于这一观察，DVR 首先提出无偏的 Watch Time Gain（WTG）指标，将观看时长转换为相对于同一时长区间平均水平的标准化增益；随后以 WTG 作为模型预测目标，并通过对抗学习进一步去除预测结果中隐含的视频时长信息，从而降低 Duration Bias 对推荐模型的影响。

在短视频推荐中，较长的视频天然具有更大的观看时长上限，因此直接比较不同视频的 Watch Time 会使模型倾向于推荐长视频。例如，一个用户观看 60 秒视频 50 秒，而另一个 60 秒视频仅观看 5 秒，前者显然更能体现用户兴趣；但如果一个 30 秒视频被观看 29 秒、一个 120 秒视频被观看 60 秒，仅从 Watch Time 来看后者更高，却无法说明用户对后者具有更强的偏好。因此，DVR 算法认为 Watch Time 只有在控制视频 Duration 之后才具有较好的用户兴趣表征能力。

具体而言，DVR 算法首先按照视频 Duration 将所有视频划分为若干个等宽的区间，然后将每个视频映射到对应的 Duration 分桶上，即：

$$B(v) = f_b(d_v), \quad B = [b_1, \ldots, b_m] \tag{23.105}$$

其中，d_v 表示视频 v 的 Duration，$f_b(\cdot)$ 表示 Duration 到对应时长分桶的映射函数。对于每一个 Duration 分桶，分别统计其中所有用户行为记录的平均 Watch Time μ_B 和标准差 σ_B，然后将原始 Watch Time 进行标准化，就得到 WTG 指标：

$$WTG = \frac{WT - \mu_{B(v)}}{\sigma_{B(v)}} \tag{23.106}$$

因此，WTG 指标衡量的并不是用户实际观看了多少秒，而是该用户对当前视频的观看表现相对于相似时长视频平均水平高出了多少。这种定义方式跟 DeepSeek 团队提出的 GRPO 方法中的 Advantage 定义方式是比较像的。在经过 Duration 分桶内的标准化处理之后，不同时长的视频可以在相对统一的尺度下进行比较。例如，WTG 为 0 意味着该视频的观看表现接近相同时长视频的平均水平，而 WTG 为 1 则意味着其观看表现明显高于同 Duration 视频。相比直接使用 Watch Time，WTG 指标能够显著削弱视频 Duration 对用户兴趣评价的主导作用。

在得到 WTG 指标后，DVR 算法进一步将 WTG 指标作为推荐模型的预测目标，而不是直接预测 Watch Time。具体地，给定用户和视频特征 X，Φ 网络输出预测 WTG 指标：

$$\hat{Y}_{WTG} = \Phi(X) \tag{23.107}$$

线上推荐时，直接按照预测 WTG 指标对候选视频进行排序，并选择 Top-k 视频进行推荐。由于 WTG 指标已经在 Duration 分桶内部进行了标准化，因此模型学习到的目标更加接近用户相对兴趣，而不是简单学习视频 Duration 与 Watch Time 之间的相关关系。

然而，仅仅移除 Duration 特征并将 WTG 作为 Label，并不能完全消除 Duration Bias。原因在于 Duration 的信息可能通过其他特征间接进入模型，例如视频 Category、内容类型以及 Creator 等特征都可能与视频 Duration 存在相关性。因此，即使输入特征中不包含 Duration 特征，模型仍然可能从其他 Feature 中间接恢复 Duration 信息。

为进一步消除这种隐式关联，DVR 算法在 WTG 预测模型 Φ 之后增加一个 Duration 预测网络 Ψ，尝试仅根据 Φ 网络输出的 WTG 指标来预测视频的 Duration：

$$\hat{Y}_D = \Psi(\hat{Y}_{WTG}) \tag{23.108}$$

其中，Ψ 预测网络的目标是尽可能准确地从预估的 WTG 指标中恢复 Duration，而 Φ 网络则通过对抗训练尽可能隐藏 Duration 相关信息。换句话说，Ψ 网络负责挖掘 Duration 信息，而 Φ 网络负责消除 Duration 信息，两个网络形成了某种对抗的关系。DVR 算法采用 Gradient Reversal Layer（GRL）来实现这一对抗过程，使得 Duration 预测网络正常优化，而梯度经过 GRL 反向传播到推荐模型时，会推动 Φ 网络学习到时长无关（Duration-invariant）

的用户兴趣表示。

因此，对应的 WTG 预测损失和 Duration 预测损失分别定义如下：

$$L_{WTG} = \text{MSE}(\hat{Y}_{WTG}, Y_{WTG}) \tag{23.109}$$

$$L_D = \text{MSE}(\hat{Y}_D, Y_D) \tag{23.110}$$

其中，Y_{WTG} 和 Y_D 分别表示真实 WTG 指标和视频 Duration。最终，WTG 预测模型 Φ 和 Duration 预测模型 Ψ 采用对抗式目标进行联合优化：

$$L_\Phi = L_{WTG} - \alpha L_D, \qquad L_\Psi = \alpha L_D \tag{23.111}$$

其中 α 用于控制 Duration 纠偏的强度。对于 Ψ 网络而言，希望最小化 L_D 损失函数来提高 Duration 预测能力；而对于 Φ 网络而言，则希望在保证 WTG 指标预测准确性的同时去最大化 L_D 损失函数，使得 Duration 预测网络无法从 $\hat{Y}_{WTG}$ 中获取有效的 Duration 信息。通过这种方式，整个时长预估模型最终能够得到更加独立于视频 Duration 的用户兴趣预测结果。

为了衡量基于 WTG 指标进行推荐的效果，DVR 算法进一步定义了 WTG@k，即 Top-k 推荐视频真实 WTG 的平均值：

$$WTG@k = \frac{1}{k}\sum_{i=1}^{k} WTG(l_i) \tag{23.112}$$

其中 l_i 表示按照预测 WTG 排序后第 i 个推荐视频。WTG@k 越高，说明模型选择的视频相对于相同 Duration 的视频具有更高的用户观看表现。由于 WTG@k 不考虑 Top-k 内部的具体排序，因此 DVR 进一步引入带位置折扣的 DCWTG@k，使排名靠前的视频获得更高权重：

$$DCWTG@k = \sum_{i=1}^{k} \frac{WTG(l_i)}{\log_2(1+i)} \tag{23.113}$$

这两种基于 WTG 的指标与传统的 DCG 和 NDCG 指标类似，但更关注用户在相同 Duration 条件下的相对观看表现，而不是绝对观看时长。

总的来说，DVR 算法的核心并不是设计一个复杂的网络结构，而是围绕 Duration Bias 重新定义什么是值得优化的观看时长，也就是我们前面几个章节一直讨论的问题：

笔记 如何定义一个合理的时长 Label 来指导时长预估模型的学习？

DVR 算法首先通过 WTG 指标来将绝对的 Watch Time 转化为 Duration 条件下的相对观看增益，再通过去除 Duration 特征输入和引入对抗学习的范式来进一步抑制时长预估模型对 Duration 信息的依赖。因此，DVR 算法可以直接作为一个 Debiasing 框架叠加到不同的粗精排模型或者 LTR 模型上，也是工业短视频推荐中 Duration Bias 建模领域的经典代表方法。

23.4.3 D^2Co 算法

D^2Co 算法源自于中国人民大学高瓴人工智能学院与华为诺亚方舟实验室合作的 RecSys 2023 会议论文《Uncovering User Interest from Biased and Noised Watch Time in Video Recommendation》[7]。该方法进一步研究了观看时长作为用户兴趣代理标签时存在的问题。前面章节中的 D2Q 算法主要通过时长分桶与分位数变换来缓解 Duration Bias，而 D^2Co 算法则进一步指出，用户的观看时长中除了视频时长带来的系统性偏差之外，还存在着大量与用户真实兴趣无关的 Noisy Watching 行为。例如，用户可能因为误触、视频开头吸引力、滑动延迟等原因短暂观看一个实际上并不感兴趣的视频。因此，用户的观看时长会同时受到 Duration Bias 和 Noisy Watching 的影响，只是进行简单的归一化或分桶排序仍然难以得到真正反映用户兴趣的监督信号。因此，D^2Co 算法的核心思想是从观看时长的生成过程出发，对这两类因素进行统一建模，并进一步从观测观看时长中恢复潜在的用户兴趣。

对于用户 u 和视频 v 构成的样本 (u, v)，我们记其特征表示为 x，视频时长为 d，用户的实际观看时长为 w，用户对视频的真实兴趣为不可观测变量 $R \in \{0, 1\}$。在理想情况下，推荐模型应该直接学习用户兴趣概率

$P(R=1|x)$，其对应的理想目标可以写成如下形式：

$$L_{\text{ideal}} = -\frac{1}{|\mathcal{D}|}\sum_{(x,r)\in\mathcal{D}}[r\log\sigma(f(x)) + (1-r)\log(1-\sigma(f(x)))] \tag{23.114}$$

其中，r 表示用户对视频的真实兴趣，$f(x)$ 表示模型对用户兴趣的预测值。然而，在真实推荐系统中用户对视频的真实兴趣 R 通常是无法直接观测的，因此我们只能使用用户的观看时长 w 来作为用户兴趣的代理标签。最直接的做法是将观看时长归一化后作为训练目标，但这种方式实际上默认观看时长能够准确反映用户兴趣。在现实中，观看时长还会受到视频时长和非兴趣观看行为的共同影响，比如用户只是将手机放在一边纯播放视频但不是真正的观看。因此，直接拟合观看时长会使模型学习到大量与真实兴趣无关的信息。

为了分析观看时长中不同因素的作用，D^2Co 算法从因果关系角度将用户兴趣、视频时长与观看时长联系起来。对于给定的用户与视频特征 x，视频时长 D 和用户兴趣 R 共同决定最终的观看时长 W。由于一个视频具有确定的视频时长 d，观看时长的条件期望可以进一步分解为

$$w = P(R=1|x)E(W|d,R=1) + P(R=0|x)E(W|d,R=0) \tag{23.115}$$

记 $p_x^r = P(R=1|x)$，$w_d^+ = E(W|d,R=1)$，$w_d^- = E(W|d,R=0)$，则上述关系可以简化为

$$w = p_x^r w_d^+ + (1-p_x^r)w_d^- \tag{23.116}$$

这个分解是 D^2Co 算法的基础。其中，p_x^r 表示用户对当前视频的真实兴趣概率，是模型真正希望学习的目标；w_d^+ 表示用户对视频感兴趣时，在视频时长为 d 的情况下产生的平均观看时长，其中包含明显的 Duration Bias；而 w_d^- 则表示用户实际上不感兴趣时仍然产生的平均观看时长，主要反映误触、短暂停留等 Noisy Watching 行为。因此，观测到的观看时长并不是单纯的用户兴趣信号，而是兴趣驱动观看、时长偏差和噪声观看三者共同作用后的混合结果。

这一分解也解释了为什么传统的时长归一化方法存在局限。例如，Play Complete Rate（PCR）指标直接使用观看时长与视频时长的比值作为兴趣标签，即：

$$r_x^{\text{PCR}} = \frac{w}{d} \tag{23.117}$$

PCR 指标能够在一定程度上消除不同视频时长带来的量纲差异，但它隐含假设感兴趣和不感兴趣状态下的观看时长都与视频时长呈线性关系，即 $w_d^+ = C_1 d$ 且 $w_d^- = C_2 d$。实际数据中的观看行为通常并不满足这种简单的线性关系，因此 PCR 无法从根本上恢复真实用户兴趣。

D2Q 算法和 WTG 指标进一步通过时长分组后的相对位置消除 Duration Bias，但它们仍然没有显式建模 Noisy Watching。特别是 WTG 指标假设不同视频时长下用户兴趣分布具有相同的统计特征，而 D2Q 算法则隐含假设不同视频时长分组中的用户兴趣排序具有一致性。当视频时长与用户兴趣本身存在相关性时，这些假设并不一定成立。D^2Co 算法因此不再直接对观看时长进行简单的归一化或排序，而是尝试估计式中的两个潜在观看时长分量 w_d^+ 和 w_d^-，再利用它们反推出用户兴趣。

从概率分布的角度来看，对于给定的用户与视频样本 x，观看时长同样可以被理解为两个潜在分布的混合：

$$P(W=w|x) = P(R=1|x)P(W=w|d,R=1) + P(R=0|x)P(W=w|d,R=0) \tag{23.118}$$

其中，$P(W|d,R=1)$ 对应用户真正感兴趣时产生的观看时长分布，主要体现 Duration Bias；$P(W|d,R=0)$ 对应用户不感兴趣时产生的观看时长分布，主要体现 Noisy Watching。理论上，如果能够分别估计这两个潜在分布，就能够进一步得到 w_d^+ 和 w_d^-，并根据前面的混合关系恢复 p_x^r。

但是，在单个用户与视频样本上通常只有一次观测到的观看时长，无法直接利用高斯混合模型（Gaussian Mixture Model, GMM）估计两个潜在分布。因此，D^2Co 算法进行了一个非常关键的转换，即：将个体级别的混合分布提升到视频时长级别进行估计。具体而言，对于相同视频时长 d 的所有用户与视频交互记录，其整体观看时长分布可以看作两个潜在观看分布的混合：

$$P(W=w|d) = p_d P(W=w|d,R=1) + (1-p_d)P(W=w|d,R=0) \tag{23.119}$$

其中 p_d 表示视频时长为 d 时整体用户的平均兴趣水平。由于同一视频时长下能够收集大量不同用户的观看记录，因此可以基于视频时长使用两成分的 GMM 来对用户观看时长的分布进行拟合，从而估计出两个潜在高斯

分布的均值，并分别得到 $\hat{w}_d^+$ 和 $\hat{w}_d^-$。

这里的关键在于：

命题 23.4

D^2Co 算法并不是试图为每个用户单独估计一套观看时长分布，而是利用不同用户在相同视频时长下产生的大量交互数据，共同估计该时长对应的两种潜在观看行为。

D^2Co 算法论文在实际数据集（比如 KuaiRand）中观察到，固定视频时长下的观看时长分布往往呈现较明显的双峰结构，这与“感兴趣观看”和“不感兴趣但产生观看”两类行为的混合假设相吻合，因此采用两成分的 GMM 具有一定的数据依据。

不过，由于不同视频时长下的数据量存在明显差异，直接使用 GMM 估计得到的 $\hat{w}_d^+$ 和 $\hat{w}_d^-$ 仍然可能存在较大的统计噪声。同时，相邻视频时长对应的用户观看行为通常具有较强的连续性，例如 20 秒和 21 秒视频的感兴趣观看时长不应该出现剧烈变化。因此，D^2Co 算法进一步对不同 Duration 上的 GMM 估计结果进行频次加权的双向移动平均。具体来说，对于时长 d_i，其平滑后的估计值可以表示为

$$\tilde{w}_{d_i}^+ = \frac{\sum_{j=i-T}^{i+T} |D_j| \hat{w}_{d_j}^+}{\sum_{j=i-T}^{i+T} |D_j|} \tag{23.120}$$

$$\tilde{w}_{d_i}^- = \frac{\sum_{j=i-T}^{i+T} |D_j| \hat{w}_{d_j}^-}{\sum_{j=i-T}^{i+T} |D_j|} \tag{23.121}$$

其中，T 表示移动平均窗口大小，$|D_j|$ 表示视频时长为 d_j 时对应的样本数量。通过这种方式，样本量较大的 Duration 对估计结果具有更高的贡献，同时利用相邻 Duration 之间的连续性降低 GMM 估计带来的随机波动，最终得到更加平滑的 $\tilde{w}_d^+$ 和 $\tilde{w}_d^-$。

在获得偏差项和噪声项之后，最直接的做法是根据观看时长的混合关系进行反解，从而得到 D^2Co(A) 的兴趣估计：

$$r_x^{D^2Co(A)} = \frac{w - \tilde{w}_d^-}{\tilde{w}_d^+ - \tilde{w}_d^-} \tag{23.122}$$

当 $\tilde{w}_d^+ = w_d^+$ 且 $\tilde{w}_d^- = w_d^-$ 时，将式中的 $w = p_x^r w_d^+ + (1 - p_x^r) w_d^-$ 代入即可得到 $r_x^{D^2Co(A)} = p_x^r$。因此，从理论上讲，如果两个潜在观看时长分量能够被准确估计，该线性校正可以无偏地恢复用户兴趣。

然而，实际情况下 w_d^+ 和 w_d^- 无法被精确估计，GMM 的参数估计误差会进一步传递到最终的兴趣标签中。特别是在 $\tilde{w}_d^+$ 和 $\tilde{w}_d^-$ 比较接近时，分母较小，线性校正会对两个估计量的误差非常敏感。此外，这种敏感性还与当前样本的观看时长有关：对于观看时长较短的样本，兴趣估计对噪声项 $\tilde{w}_d^-$ 更加敏感；而对于观看时长较长的样本，则更加容易受到偏差项 $\tilde{w}_d^+$ 估计误差的影响。因此，简单的线性反解虽然具有较好的理论可解释性，但在实际数据存在估计误差时可能产生较大的标签波动。

为提高兴趣恢复过程对估计误差的鲁棒性，D^2Co 算法进一步提出了敏感度控制校正（Sensitivity-controlled Correction）策略，将线性校正替换为指数空间中的非线性校正函数：

$$r_x^{D^2Co(S)} = \frac{\exp(\alpha w) - \exp(\alpha \tilde{w}_d^-)}{\exp(\alpha \tilde{w}_d^+) - \exp(\alpha \tilde{w}_d^-)} \tag{23.123}$$

其中，α 为敏感性控制参数。通过调整 α，可以改变校正函数对 $\tilde{w}_d^+$ 和 $\tilde{w}_d^-$ 估计误差的敏感程度。当 α 取不同方向和不同幅度的值时，可以针对数据集中短观看时长或长观看时长样本占比较高的情况调整误差传播特性，从而降低潜在观看时长估计误差对最终兴趣标签的影响。相比 D^2Co(A) 的直接线性反解，D^2Co(S) 更加关注实际估计误差下的稳定性，因此最终采用 D^2Co(S) 作为用户兴趣标签的生成方式。

最终，D^2Co 算法不需要修改下游推荐模型的网络结构，而是将经过 Debias 和 Denoise 处理后的 $r_x^{D^2Co(S)}$ 作为新的监督信号，来训练任意能够进行实值预测的时长预估模型。其整体流程可以概括为：

- 首先按照视频 duration 统计观看时长分布，并利用两成分 GMM 估计用户感兴趣和不感兴趣两种状态下的潜在观看时长，分别对应 Duration Bias 和 Noisy Watching；

- 随后利用频次加权移动平均对不同 duration 上的估计结果进行平滑，降低单个 duration 数据量不足带来的估计噪声；
- 最后通过敏感度控制校正策略从原始观看时长中分离出用户兴趣，并将其作为推荐模型的训练标签。

由此，D^2Co 算法将 Duration Bias 和 Noisy Watching 统一纳入同一观看时长生成框架，在不依赖具体下游模型结构的情况下，将原始观看时长转化为更加接近真实用户兴趣的监督信号。相比仅针对 Duration Bias 进行校正的 D2Q 算法和 WTG 算法，D^2Co 算法进一步考虑了用户在低兴趣状态下产生的 Noisy Watching，并通过对潜在偏差项和噪声项进行显式建模与校正，能够在降低时长偏差的同时减少无兴趣观看行为对用户兴趣建模的干扰，从而获得更加准确且更加符合真实用户偏好的推荐目标。

23.4.4 OR 算法

Ordinal Regression（OR），即经典的有序回归方法，其核心思想是利用目标变量天然存在的顺序关系，将原本的连续回归问题转化为多个具有序关系的二分类任务。与直接对连续观看时长进行回归不同，OR 算法首先将观看时长划分为若干个有序区间，并围绕不同的时长阈值判断样本是否超过该阈值，从而将复杂的连续值预测转化为一系列相对简单的分类问题。与普通多分类方法将不同类别视为相互独立的类别不同，OR 算法显式利用了类别之间的顺序信息，使相邻时长区间之间的关系能够参与模型学习。例如，若一个用户对视频的实际观看时长为 80 秒，那么其必然同时满足“观看时长超过 20 秒”、“超过 40 秒”和“超过 60 秒”等条件，因此不同阈值对应的二分类 Label 之间具有天然的嵌套和单调关系。

具体而言，首先将连续观看时长划分为 K 个有序区间，并设置 $K-1$ 个递增阈值：

$$0 = t_0 < t_1 < \cdots < t_{K-1} < t_K = T_{\max} \tag{23.124}$$

对于第 k 个阈值，可以构造累计二分类 Label：

$$z_k = \mathbb{I}(y > t_k) \tag{23.125}$$

其中，$z_k = 1$ 表示用户观看时长超过阈值 t_k，否则为 0。由于不同阈值之间存在天然的序关系，因此对于同一个样本，其 Label 通常具有类似 $(1,1,1,0,0,\ldots)$ 的结构，而不会出现 $(1,0,1,0,\ldots)$ 这样的非单调模式。

然后时长预估模型针对每一个阈值预测对应的累计概率：

$$p_k(x) = P(y > t_k \mid x) \tag{23.126}$$

从概率分布的角度来看，$p_k(x)$ 可以理解为条件 Survival Function 在离散阈值 t_k 处的取值。进一步地，如果定义条件累积分布函数（Cumulative Distribution Function，CDF），则有：

$$F(t_k \mid x) = P(y \le t_k \mid x) = 1 - p_k(x) \tag{23.127}$$

因此，OR 算法实际上是在多个离散阈值上估计观看时长的条件累计分布，而不是直接预测一个连续的点估计。由于生存概率随阈值增大应该单调下降，理论上应满足：

$$p_1(x) \ge p_2(x) \ge \cdots \ge p_{K-1}(x) \tag{23.128}$$

在最直接的实现中，可以将每一个累计阈值视为一个二分类任务，并使用二元交叉熵（Binary Cross-Entropy）损失函数来进行联合优化：

$$L_{\mathrm{OR}} = -\sum_{k=1}^{K-1} \left[z_k \log p_k(x) + (1 - z_k) \log(1 - p_k(x)) \right] \tag{23.129}$$

因此，Ordinal Regression 算法的本质并不是简单地将连续值转换成普通的多分类问题，而是将一个连续有序变量转换为多个具有累计关系的二分类问题。相比直接回归，这种方式能够利用成熟的分类损失学习不同时间尺度上的观看行为，同时通过多个阈值共同描述观看时长的相对位置。

在预测阶段，可以进一步利用这些累计概率恢复连续观看时长。对于非负随机变量，其条件期望可以表示为生存函数的积分：

$$E[y \mid x] = \int_0^{T_{\max}} P(y > t \mid x)\, dt \tag{23.130}$$

因此，在离散阈值下可以采用分段积分近似：

$$\hat{y} = \sum_{k=1}^{K} p_k(x)(t_k - t_{k-1}) \tag{23.131}$$

其中，最后一个区间的概率可以根据实际截断方式进行处理。由此，Ordinal Regression 可以形成完整的“**累计概率预测 → 生存概率 → 连续时长恢复**”建模流程。

在观看时长预估场景中，Tree-based Progressive Regression Model（TPM）[2] 的工作进一步将 Ordinal Regression 作为观看时长预估的基础方法进行实验，并直接将连续观看时长划分为多个有序等级，再利用多个分类器预测观看时长是否超过对应 Rank。CREAD[5] 算法同样将 OR 作为重要的 Baseline，并在实验中通过不同离散化粒度进行比较，最终选择 $K = 80$ 作为较优设置。由此可见，Ordinal Regression 已经成为长尾连续时长预估中较为典型的“Regression by Classification”建模范式。

不过，传统 Ordinal Regression 算法的一个核心问题在于离散化策略。由于模型最终依赖一组预先确定的阈值，因此阈值的位置和数量会直接影响模型的学习难度以及连续值恢复精度。当分桶数量较少时，不同观看时长会被压缩到较宽的区间中，导致较大的修复误差（Restoration Error）；而当分桶数量过多时，每个分类任务能够获得的有效样本减少，又容易增加学习误差（Learning Error）。与此同时，传统 OR 算法通常分别优化不同阈值对应的分类任务，不同阈值之间虽然在 Label 定义上具有序关系，但模型输出之间未必能够严格满足单调约束。

23.4.5 CREAD 算法

CREAD（Classification and REconstruction-based Accurate Duration prediction）算法源自于快手科技在 AAAI 2024 上发表的论文《CREAD: A Classification-Restoration Framework with Error Adaptive Discretization for Watch Time Prediction in Video Recommender Systems》[5]。CREAD 的核心思想如下：

定义 23.4 (CREAD 算法核心思想)

CREAD 算法将连续观看时长预估转化为多个有序二分类任务，并进一步利用这些分类概率恢复连续观看时长。相比前文介绍的阈值化时长建模，前面的阈值化方法通常针对 EVTR、LVTR 等具体业务指标定义少量具有明确业务含义的时长阈值，而 CREAD 算法则尝试使用一组覆盖整个观看时长空间的阈值，将连续回归问题转化为更加细粒度的多阈值分类问题，最终再通过概率积分恢复绝对观看时长。

CREAD 算法是一种基于 Ordinal Regression 方法的改进方法。从方法关系上看，OR 算法与 CREAD 算法并不是两个完全独立的建模范式，而更适合理解为“**基础范式与增强框架**”的关系。CREAD 算法保留了 OR 算法中多阈值分类和连续值恢复的基本思想，但进一步引入 Error-Adaptive Discretization（EAD）解决离散化策略问题，同时增加 Restoration Loss 直接优化最终观看时长误差，并通过 Ordinal Regularization 显式约束不同阈值预测概率之间的单调关系。相比之下，标准 OR 更强调利用观看时长的有序结构将回归问题转化为累计分类问题，其性能更加依赖于预先设计的离散化策略。

因此，若从整个观看时长预估的技术演进来看，可以将 OR 算法视为“Regression by Classification”在连续时长建模中的基础形式，而 CREAD 算法则是在这一范式上进一步针对离散化误差、恢复误差以及输出单调性进行系统优化的方法。后续的 TPM 算法则进一步从模型结构层面出发，通过树状的条件分类过程显式建模不同观看时长区间之间的依赖关系，从而形成与 CREAD 算法不同的另一条优化路线。

下面我们重点介绍 CREAD 算法的数学细节。假设训练数据为 $\{(x_i, y_i)\}_{i=1}^{N}$，其中 x_i 表示用户、视频以及上下文等输入特征，$y_i \in \mathbb{R}_+$ 表示真实观看时长。首先将观看时长限制在 $[0, T_{\max}]$ 范围内，并定义 $M-1$ 个时长阈值：

$$0 = t_0 < t_1 < \cdots < t_{M-1} < t_M = T_{\max} \tag{23.132}$$

这些阈值将整个观看时长空间划分为 M 个互不重叠的桶：

$$d_m = [t_{m-1}, t_m), \qquad m = 1, \ldots, M \tag{23.133}$$

在此基础上，CREAD 同样不直接预测样本属于哪一个桶，而是针对每一个阈值 t_m 构造一个二分类 Label：

$$y_m = \mathbb{I}(y > t_m) \tag{23.134}$$

其中，$\mathbb{I}(\cdot)$ 为指示函数。当真实观看时长超过某个阈值时，对应的 Label 为 1，否则为 0。例如，当一个样本的真实观看时长为 35 秒，并设置 10、20、30、40 秒等多个阈值时，该样本对应的 Label 序列大致为 $(1, 1, 1, 0, \ldots)$。因此，这些二分类 Label 并不是相互独立的，而是天然具有有序的结构。

对于每一个阈值 t_m，CREAD 算法使用一个分类器预测用户观看时长超过该阈值的概率：

$$\hat{\phi}_m(x) = P(y > t_m \mid x) \tag{23.135}$$

这样一个连续的观看时长预估问题就被转换成多个具有明确物理意义的概率预测问题。这里的 $\hat{\phi}_m(x)$ 还可以理解为观看时长在时间 t_m 之后仍然存活的概率，即生存概率（Survival Probability）。随着阈值不断增大，理论上的生存概率应当单调下降，因此 CREAD 算法在训练过程中还显式利用了这种有序关系。

得到多个分类概率之后，CREAD 算法并不是简单地选择概率最大的分桶作为最终预测结果，而是进一步设计 Watch Time Restoration 模块，将整个生存概率曲线恢复为连续观看时长。对于非负随机变量，其期望可以表示为生存函数（Survival Function）的积分：

$$E[y \mid x] = \int_0^{T_{\max}} P(y > t \mid x)\, dt \tag{23.136}$$

因此，将连续积分按照离散阈值进行近似，可以得到：

$$\hat{y} = \sum_{m=1}^{M} \hat{\phi}_m(x)(t_m - t_{m-1}) \tag{23.137}$$

这也是 CREAD 算法与普通多分类方法的一个重要区别：模型并不是输出一个离散的时长类别，而是利用不同时间阈值对应的概率共同恢复连续的 Watch Time。直观来看，每一个 $\hat{\phi}_m$ 都可以理解为生存曲线（Survival Curve）在对应时间位置上的一个采样点，而最终预测值则近似等于整个生存曲线下方的面积。

在模型训练过程中，CREAD 算法同时考虑分类准确性、最终时长恢复准确性以及不同阈值之间的序关系。首先，对于 M 个二分类任务，使用标准的交叉熵损失函数：

$$L_{\text{CE}} = -\sum_{m=1}^{M} \Big[y_m \log \hat{\phi}_m + (1 - y_m) \log(1 - \hat{\phi}_m) \Big] \tag{23.138}$$

其次，由于线上真正使用的是恢复后的连续观看时长，因此 CREAD 算法进一步对 $\hat{y}$ 与真实观看时长 y 之间的误差进行直接约束：

$$L_{\text{restore}} = \ell(\hat{y}, y) \tag{23.139}$$

其中，CREAD 算法采用 Huber Loss 来作为恢复损失，以降低长尾观看时长中极端样本对训练过程的影响。

此外，由于不同阈值对应的生存概率具有明确的单调关系，因此 CREAD 算法增加了 Ordinal Regularization，对相邻阈值概率出现反常的情况进行惩罚：

$$L_{\text{ord}} = \sum_{m=1}^{M-1} \max(\hat{\phi}_{m+1} - \hat{\phi}_m, 0) \tag{23.140}$$

CREAD 算法的最终训练目标可以表示为：

$$L_{\text{CREAD}} = \lambda_{\text{CE}} L_{\text{CE}} + \lambda_{\text{restore}} L_{\text{restore}} + \lambda_{\text{ord}} L_{\text{ord}} \tag{23.141}$$

其中，λ_{CE}、λ_{restore} 和 λ_{ord} 分别用于控制三部分损失的权重。

从上述过程可以看到，CREAD 算法与 OR 的核心框架实际上非常接近，二者都采用“**连续时长离散化 → 多阈值分类 → 连续值恢复**”的思路。二者真正的区别主要体现在三个方面。首先，OR 算法更关注如何通过多个有序分类任务完成连续值预测，而 CREAD 算法进一步将最终的连续时长恢复误差纳入训练目标，通过 L_{restore} 直接优化最终的 Watch Time。其次，CREAD 算法显式加入了 Ordinal Regularization，使不同阈值的预测概率满足更加稳定的单调关系。更重要的是，OR 算法通常将分桶策略作为固定的预处理过程，而 CREAD 算法进一步研究了“如何分桶”本身对模型性能的影响，并提出了 Error-Adaptive Discretization（EAD）方法。因此，可以将 OR

算法理解为 CREAD 算法的基础建模范式，而 CREAD 算法则是在此基础上进一步解决离散化误差和连续值恢复问题。

CREAD 算法中一个非常关键的问题正是如何进行 Watch Time 的离散化处理。直观来看，桶数量越多，阈值越密集，离散化之后对原始生存概率的近似就越精细，但每个桶能够获得的有效训练样本也会减少，使对应的分类任务更加难以训练。反之，如果分桶数量过少，每个分类任务虽然更加容易学习，但较大的桶宽又会使最终的积分近似产生更大的恢复误差。因此，CREAD 算法将离散化带来的误差概括为两类：一类是由于每个桶的样本量有限而产生的学习误差（Learning Error），另一类是由于连续时长分布被离散化后进行积分近似而产生的恢复误差（Restoration Error）。两者之间存在明显的权衡关系。

传统的分桶方式通常采用等距（Equal-width）或等频（Equal-frequency）分桶。等距（Equal-width）分桶将整个时长范围等间隔划分：

$$t_m = \frac{m}{M} T_{\max} \tag{23.142}$$

这种方法能够保证每个桶具有相同的物理时长宽度，但对于典型的长尾观看时长分布而言，长时长区域往往样本非常稀疏，导致部分桶的训练样本过少。另一种常见方式是等频，即按照 Watch Time 的经验分布进行等频分桶：

$$t_m = \Psi^{-1}\left(\frac{m}{M}\right) \tag{23.143}$$

其中，Ψ 为 Watch Time 的累积分布函数。该方法可以保证不同桶中具有近似相同的样本量，但由于长尾区域的概率密度较低，可能导致部分桶的物理时长跨度过大，从而增加 Restoration Error。

针对上述问题，CREAD 算法进一步提出 EAD（Error-Adaptive Discretization）策略，在等距和等频分桶之间寻找更加合理的折中。其核心思想是通过一个校准函数（Calibration Function）γ 对分位数空间进行重新映射：

$$t_m = \Psi^{-1}\left[\gamma\left(\frac{m}{M}\right)\right] \tag{23.144}$$

其中，$\gamma : [0,1] \to [0,1]$ 且满足 $\gamma(0) = 0, \gamma(1) = 1$。当 $\gamma(z) = z$ 时，该方法退化为等频分桶；而当 $\gamma(z) = \Psi(zT_{\max})$ 时，则可以得到等距分桶。因此，通过调整 γ 的具体形式，可以在两种极端分桶方式之间进行连续调节。

CREAD 算法进一步从误差角度分析了不同离散化策略的影响。论文将与模型学习能力相关的误差记为学习误差（Learning Error），将由离散化和恢复过程本身引入的误差记为修复误差（Restoration Error），并证明二者与分桶策略具有直接关系。这里不再展开具体的误差边界推导，其核心结论比较直观：分桶中的样本量过少会导致分类模型难以准确学习，从而增加学习误差；而分桶的时间跨度过大，则会降低离散积分对真实生存曲线的近似精度，从而增加修复误差。因此，分桶并不是越细越好，也不是简单采用等频或等宽即可。

基于这一观察，CREAD 算法将 EAD 的目标概括为同时降低学习误差和修复误差：

$$\min_{\mathcal{D}} J(\mathcal{D}) = A_w(\mathcal{D}) + \beta A_b(\mathcal{D}) \tag{23.145}$$

其中，A_w 主要刻画分桶样本量不足带来的学习误差，A_b 则主要刻画分桶宽度过大带来的修复误差，β 用于控制两类误差之间的权衡。在实际实现中，CREAD 算法并不直接求解复杂的高维优化问题，而是设计带有少量超参数的校准函数，并通过网格搜索选择更加合适的分桶策略。

总体而言，CREAD 算法的核心贡献可以概括为对 OR 算法这种“Regression by Classification”方法的一次系统增强。OR 算法解决了如何利用多个有序分类任务近似连续时长回归的问题，而 CREAD 算法在此基础上进一步引入了“分类 + 恢复”的联合优化机制，并重点解决了连续时长离散化过程中学习误差与修复误差的权衡问题。最终形成了“**自适应分桶 → 多阈值分类 → 生存概率 → 连续时长恢复**”的完整建模流程。对于工业推荐系统而言，这种方法既保留了分类模型在长尾分布下相对稳定的学习能力，又能够通过概率恢复得到连续的绝对观看时长，因此特别适合用于需要直接预测 Watch Time 的推荐排序和融合模型。

23.4.6 TPM 算法

TPM（Tree based Progressive Regression Model）源自于快手科技发表在 KDD 2023 会议上的论文《Tree based Progressive Regression Model for Watch-Time Prediction in Short-video Recommendation》[2]。与直接对连续观看时

长进行回归不同，TPM 算法将观看时长预估转化为一个基于树结构的渐进式搜索过程：首先将连续观看时长离散为多个有序的 Rank，然后通过树中的一系列分类器逐层判断观看时长所属的区间，最终得到各个 Rank 的概率分布，并进一步根据该分布恢复连续观看时长。相比于传统的直接回归方法，TPM 算法能够利用分类任务相对稳定的学习特性，同时通过树结构将原本一次性的预测问题分解为多个条件依赖的局部决策。

假设输入特征为 X，真实观看时长为 T。首先将观看时长空间离散为有序的 Rank：

$$\gamma_0 < \gamma_1 < \cdots < \gamma_m \tag{23.146}$$

其中，每个 Rank 对应一个连续的观看时长区间，如图23.6所示。对于一个具体样本，观看时长预测实际上可以看作在这些有序 Rank 中进行搜索。例如，最简单的方式是按照从小到大的顺序依次判断 T 是否超过当前阈值，这相当于线性搜索；而如果每次选择中间位置的阈值进行判断，则可以通过不断缩小搜索空间实现二分搜索。TPM 算法的核心思想正是将这一搜索过程显式表示为一棵树，使每一个非叶节点对应一个分类决策，最终从根节点逐步走向某一个叶节点。

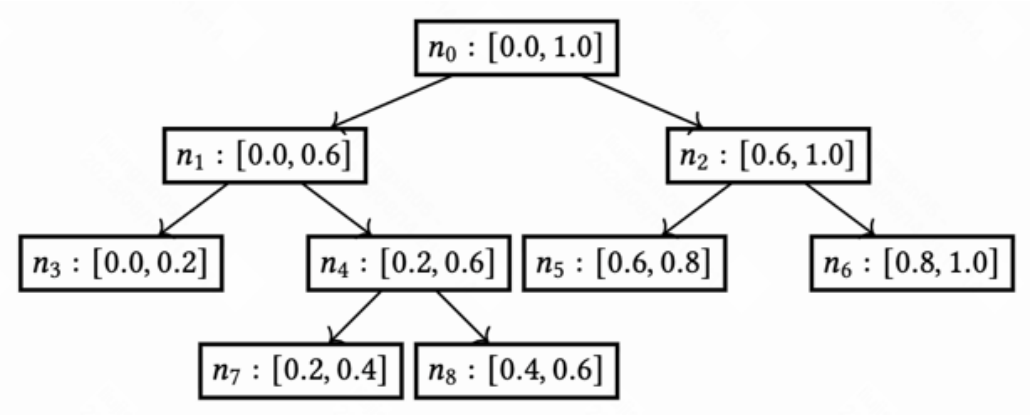

图 23.6: TPM 的树状结构示意图。

具体而言，TPM 算法构建一棵由非叶节点和叶节点组成的分解树 $\mathcal{T}$。根节点表示完整的观看时长空间，每个非叶节点表示一个连续的 Rank 区间，而叶节点则对应最终的某一个 Rank。例如，在二叉树结构下，父节点可以将当前观看时长空间划分为左右两个子空间，分类器负责判断当前样本应该进入左子节点还是右子节点。经过多层连续决策后，样本最终到达某一个叶节点，从而确定其对应的观看时长区间。这样，原本需要在大量 Rank 中进行一次性预测的问题，就被转化为多个条件依赖的二分类任务。

对于从根节点到某一个叶节点 l_k 的路径，记该路径上的节点依次为 ϕ_{l_k}。TPM 中每一个非叶节点都对应一个分类器，用于预测在已经到达当前父节点的条件下，样本属于不同子节点的概率。因此，一个叶节点最终被选中的概率并不是由单个分类器直接给出，而是由路径上所有分类决策的条件概率相乘得到：

$$p(T \in l_k \mid X, \mathcal{T}) = \prod_{i=1}^{d(l_k)} p\left(T \in n_{\phi_{l_k}(i)} \mid X, \mathcal{T}, T \in n_{\phi_{l_k}(i-1)}\right) \tag{23.147}$$

其中，$d(l_k)$ 表示叶节点 l_k 的深度。由此，TPM 最终能够得到所有叶节点对应的概率分布，而不是仅输出一个确定的观看时长。

在获得叶节点概率之后，TPM 算法进一步利用各个 Rank 的代表值计算观看时长的期望。对于叶节点 l_k，论文采用对应区间上下界的中点作为该区间的代表值：

$$c_k = \frac{\gamma_k + \gamma_{k+1}}{2} \tag{23.148}$$

因此，最终的观看时长预测可以表示为：

$$\hat{T} = E(T \mid X, \mathcal{T}) = \sum_{l_k \in \mathcal{L}_\mathcal{T}} c_k p(T \in l_k \mid X, \mathcal{T}) \tag{23.149}$$

其中，$\mathcal{L}_\mathcal{T}$ 表示树中的叶节点集合。由此可以看出，TPM 算法本质上是在预测一个离散的观看时长概率分布，再利用该分布的期望恢复连续观看时长。如果进一步构建多棵不同的分解树，还可以将不同树的预测结果进行集

成，但论文中的主要实验采用单棵树进行预测。

TPM 算法中树结构的设计直接影响不同分类器的训练难度。由于观看时长通常具有明显的长尾分布，如果直接按照固定宽度进行划分，会导致部分长时长区间中的样本数量非常少，从而产生严重的样本不平衡（Label Imbalance）问题。为此，TPM 算法首先根据观看时长分布计算分位数，并将分位数对应的位置作为 Rank 划分点，然后采用递归二分的方式构建一棵近似平衡的二叉树。这样，每个非叶节点对应的左右子空间具有相对均衡的样本量，使得每一个分类器都能够获得更加平衡的正负样本。相比于直接针对所有 Rank 训练多个独立分类器，这种树结构还能够通过逐层缩小搜索空间降低单个分类任务的难度。

TPM 算法的另一个重要特点是其并不局限于输出单点观看时长，而是能够显式获得观看时长的概率分布。由于所有叶节点均具有对应的概率 $p(T \in l_k \mid X, \mathcal{T})$，因此可以进一步计算预测分布的方差：

$$\mathrm{Var}(T \mid X, \mathcal{T}) = E(T^2 \mid X, \mathcal{T}) - E(T \mid X, \mathcal{T})^2 \tag{23.150}$$

该方差可以反映模型对于当前观看时长预测的不确定性。例如，即使两个样本具有相同的预测期望，如果一个样本的概率集中在少数相邻 Rank 上，而另一个样本的概率分布分散在多个远距离 Rank 上，后者显然具有更高的不确定性。TPM 算法因此将预测不确定性也纳入训练目标，使模型不仅关注预测值是否准确，同时尽可能降低预测分布的方差。

在训练过程中，对于每一个训练样本 (X, T)，首先根据真实观看时长确定其对应的叶节点以及从根节点到该叶节点的路径，然后只在该路径上的分类器上进行训练。对于二叉树而言，如果真实样本位于右侧子节点，则对应分类任务的 Label 为正样本，否则为负样本。模型训练主要包含三个方面：首先，通过最大化真实叶节点对应路径的概率，提高树结构对于真实观看时长区间的分类能力；其次，通过加入预测分布的标准差约束降低模型的不确定性；最后，通过直接约束最终预测观看时长与真实观看时长之间的差异，提高连续值预测精度。其目标函数可以概括为：

$$\mathcal{L}_{\mathrm{TPM}} = -\alpha_1 \log p(\hat{T} \in l_k(T) \mid X, \mathcal{T}) + \alpha_2 \,\mathrm{Var}(\hat{T} \mid X, \mathcal{T})^{\frac{1}{2}} + \alpha_3 \left\| E(\hat{T} \mid X, \mathcal{T}) - T \right\|_2 \tag{23.151}$$

其中，第一项对应路径分类损失，第二项用于抑制预测分布的不确定性，第三项则直接优化最终的观看时长回归误差，$\alpha_1, \alpha_2, \alpha_3$ 为对应的损失权重。

除了利用树结构解决观看时长的长尾分布和预测不确定性问题外，TPM 算法还进一步结合后门调整（Backdoor Adjustment）处理推荐系统中的混淆偏差（Confounding Bias）。其基本动机是，观看时长不仅受到用户和视频特征 X 的影响，还可能受到一些混杂因素 D 的影响，例如曝光环境、流量分布以及其他影响用户观看行为的因素。论文将其建模为：

$$D \to X, \qquad D \to T, \qquad X \to T \tag{23.152}$$

其中，D 同时影响输入特征和观看时长，因此直接学习 $P(T \mid X)$ 可能将混杂因素带来的相关性错误地归因于 X，进一步造成偏差放大（Bias Amplification）。

为此，TPM 算法将不同混杂因素划分为若干组，并针对不同的 D 分组分别构建和训练对应的树模型。最终通过后门调整按照混杂因素在总体数据中的先验分布进行加权：

$$E(T \mid do(X)) = \sum_d P(D = d) E(T \mid X, D = d) \tag{23.153}$$

其中，每个条件分布 $E(T \mid X, D = d)$ 均可以利用 TPM 算法的树结构进行建模。因此，TPM 算法可以在原有的“树搜索 $\to$ 概率分布 $\to$ 期望恢复”框架中自然地引入因果去偏，而不需要改变其基本的预测形式。

总体而言，TPM 算法的核心思想可以概括为“**有序离散化 $\to$ 树结构渐进搜索 $\to$ 概率分布建模 $\to$ 连续时长恢复**”。与直接回归相比，TPM 算法将复杂的连续值预测问题分解为多个条件依赖的分类任务，并利用分位数构建的平衡二叉树缓解长尾观看时长带来的样本不均衡问题；与此同时，通过显式建模叶节点概率分布，TPM 算法不仅可以预测观看时长的期望，还能够进一步估计预测的不确定性。在此基础上，论文还结合后门调整对推荐系统中的混杂偏差进行处理，使 TPM 算法同时具备时长分布建模和因果去偏能力。

23.4.7 EGMN 算法

EGMN（Exponential-Gaussian Mixture Network）算法源自于小红书团队发表在 RecSys 2025 会议上的论文《Multi-Granularity Distribution Modeling for Video Watch Time Prediction via Exponential-Gaussian Mixture Network》[8]。与前面介绍的 TPM、CREAD 等方法主要围绕观看时长的离散化、分类或概率重构展开不同，EGMN 算法的核心思路是**直接对观看时长的条件概率分布进行建模**。传统回归模型通常通过 MSE、MAE 等损失函数直接学习一个点估计，而 EGMN 算法认为观看时长本身具有明显的长尾性和多模态特征，仅预测一个期望值容易丢失用户不同观看行为模式所包含的分布信息。因此，EGMN 算法使用一个由一个指数分布（Exponential Distribution）和多个高斯分布（Gaussian Distribution）组成的混合分布，对不同粒度下的观看时长分布进行统一建模，并通过神经网络根据用户、视频和上下文特征动态生成该分布的参数。

具体来说，假设 $\mathbf{x} \in \mathbb{R}^d$ 表示用户与视频交互样本的特征表示，$t \in \mathbb{R}_+$ 表示用户的真实观看时长。EGMN 算法不再直接学习一个回归函数 $\hat{t} = f(\mathbf{x})$，而是学习条件概率密度：

$$p(t \mid \mathbf{x}) \tag{23.154}$$

即给定用户、视频以及上下文特征之后，观看时长 t 所服从的概率分布。最终的观看时长点预测仍然可以通过该分布的期望得到，但模型在训练过程中学习的是完整的概率分布，因此能够同时保留观看时长的不同潜在行为模式。EGMN 算法对观看时长分布进行了一个非常直观的经验假设，即将其表示为一个 Exponential-Gaussian Mixture（EGM）分布：

$$p(t \mid \mathbf{x}) = \omega_0(\mathbf{x}) f_{\text{exp}}(t \mid \lambda(\mathbf{x})) + \sum_{k=1}^{K} \omega_k(\mathbf{x}) f_{\text{gauss}}(t \mid \mu_k(\mathbf{x}), \sigma_k^2(\mathbf{x})) \tag{23.155}$$

其中 $\omega_k(\mathbf{x})$ 表示第 k 个分布成分的混合权重，满足：

$$\sum_{k=0}^{K} \omega_k(\mathbf{x}) = 1 \tag{23.156}$$

指数成分（Exponential Component）的概率密度函数为：

$$f_{\text{exp}}(t \mid \lambda) = \lambda e^{-\lambda t} \tag{23.157}$$

高斯成分（Gaussian Component）的概率密度函数为：

$$f_{\text{gauss}}(t \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(t-\mu)^2}{2\sigma^2}\right) \tag{23.158}$$

EGMN 算法采用这两类分布具有明确的业务含义。对于较粗粒度的观看时长分布，短视频推荐中通常存在大量快速划走（Quick Skip）行为，使整体观看时长呈现明显的右偏和长尾特征，大量样本集中在接近 0 的区域。因此，指数分布可以利用其在零附近概率密度较高的特点，对这部分快速划走行为进行建模。另一方面，当进一步观察更加细粒度的观看行为时，不同用户兴趣、视频内容以及消费习惯会形成多个潜在的观看模式，例如快速浏览、正常观看以及深度观看等，使观看时长分布呈现明显的多峰特征。多个高斯分量则可以进一步对这些不同的局部模式进行建模。由此，EGMN 算法通过**“指数分布建模粗粒度长尾 + 高斯混合分布建模细粒度多模态”**的方式，来同时刻画观看时长分布的不同粒度特征。

如图23.7所示，在模型结构上，EGMN 算法本身并不限定具体的特征编码器，而是一个与 Backbone 解耦的分布建模框架（Distribution Modeling Framework）。EGMN 模型首先将用户特征、视频特征以及上下文特征经过 Embedding 等处理得到输入向量 $\mathbf{x}$，再通过共享的特征编码器得到隐藏表示：

$$\mathbf{h} = g_{\text{backbone}}(\mathbf{x}) \tag{23.159}$$

其中，$g_{\text{backbone}}(\cdot)$ 可以采用推荐系统中常见的 DCN、DIN、SENet 或 Transformer 等网络结构。因此，EGMN 算法的主要创新并不在于重新设计一个复杂的特征交互网络，而在于在已有推荐模型 Backbone 的基础上增加一个概率分布参数生成模块。

在得到隐藏表示 $\mathbf{h}$ 后，EGMN 模型通过不同的预测头（Prediction Head）分别预测 EGM 分布中的参数。首

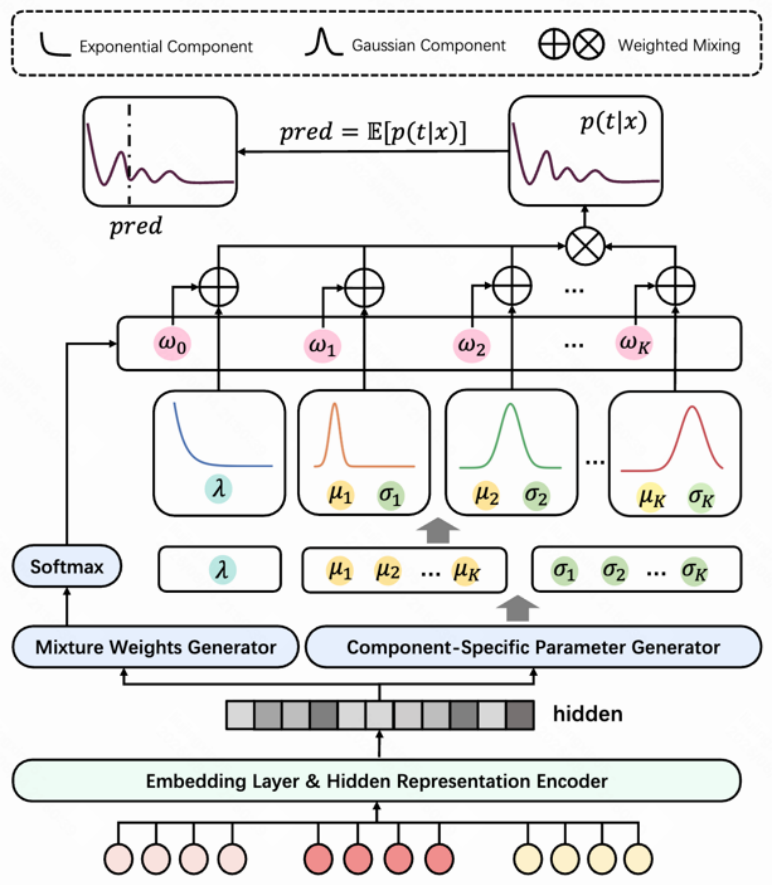

图 23.7: EGMN 的模型结构示意图。

先，对于指数成分，EGMN 模型利用 Softplus 函数来生成其速率参数（Rate Parameter）：

$$\lambda(\mathbf{x}) = \text{softplus}(\mathbf{W}_\lambda \mathbf{h} + \mathbf{b}_\lambda) \tag{23.160}$$

其中 Softplus 函数可以保证 $\lambda(\mathbf{x}) > 0$。对于第 k 个高斯成分，其均值和方差分别为：

$$\mu_k(\mathbf{x}) = \frac{1}{\lambda(\mathbf{x})} + \text{softplus}(\mathbf{W}_{\mu_k} \mathbf{h} + \mathbf{b}_{\mu_k}) \tag{23.161}$$

$$\sigma_k^2(\mathbf{x}) = \text{softplus}(\mathbf{W}_{\sigma_k} \mathbf{h} + \mathbf{b}_{\sigma_k}) \tag{23.162}$$

其中，对高斯均值（Gaussian Mean）增加 $1/\lambda(x)$ 的偏移，使得高斯成分的均值整体位于指数成分之后，从而减少不同分布成分之间的参数歧义。最后，通过 Softmax 函数生成所有分布成分的混合权重（Mixture Weight）：

$$[\omega_0(\mathbf{x}), \omega_1(\mathbf{x}), \ldots, \omega_K(\mathbf{x})] = \text{softmax}(\mathbf{W}_\omega \mathbf{h} + \mathbf{b}_\omega) \tag{23.163}$$

这样便可以得到完整的条件观看时长分布 $p(t \mid x)$。

与直接使用 MSE 或 MAE 损失函数来训练一个单点预估模型不同，EGMN 算法直接采用最大似然估计的方法来优化预测分布与真实观看时长之间的匹配程度，其核心损失可以表示为：

$$\mathcal{L}_{\text{MLE}} = -\frac{1}{N} \sum_{i=1}^{N} \log p(t_i \mid \mathbf{x}_i) \tag{23.164}$$

其中，$p(t_i \mid \mathbf{x}_i)$ 为模型在真实观看时长 t_i 处预测的概率密度。通过最大化真实样本所在位置的概率密度，模型能够学习更加符合实际数据分布的观看时长概率模型，而不再仅仅关注预测值与真实值之间的距离。由于混合模型在训练过程中可能出现某一个成分获得几乎全部概率质量、其他成分基本不参与建模的成分塌缩（Component

Collapse）问题，EGMN 算法进一步对混合权重引入熵正则（Entropy Regularization）：

$$\mathcal{L}_{\text{entropy}} = \frac{1}{N}\sum_{i=1}^{N}\sum_{k=0}^{K}\omega_k(\mathbf{x}_i)\log\omega_k(\mathbf{x}_i) \tag{23.165}$$

由于该项本身为负熵，最小化该损失相当于最大化混合权重的熵，从而鼓励模型在不同场景下合理利用多个分布成分，而不是退化为单一分布。

不过，仅仅优化概率密度并不能保证分布的期望具有较好的点预测精度。因此，EGMN 算法进一步增加回归损失，直接约束预测分布的期望与真实观看时长之间的差异。对于 EGM 分布，其期望可以直接计算为：

$$\hat{t} = \mathbb{E}[t \mid \mathbf{x}] = \omega_0(\mathbf{x})\frac{1}{\lambda(\mathbf{x})} + \sum_{k=1}^{K}\omega_k(\mathbf{x})\mu_k(\mathbf{x}) \tag{23.166}$$

论文采用 MAE 来作为回归损失：

$$\mathcal{L}_{\text{reg}} = \frac{1}{N}\sum_{i=1}^{N}|t_i - \hat{t}_i| \tag{23.167}$$

因此，EGMN 算法最终采用三部分损失的加权和进行训练：

$$\mathcal{L} = \mathcal{L}_{\text{MLE}} + \alpha\mathcal{L}_{\text{entropy}} + \beta\mathcal{L}_{\text{reg}} \tag{23.168}$$

其中，α 和 β 分别控制分布熵正则项和回归损失的权重。三部分目标分别从**分布拟合、成分利用以及点预测准确性**三个角度约束模型，使 EGMN 算法在学习完整观看时长分布的同时，也能够保持良好的连续值预测能力。

在线上推理的阶段，EGMN 算法不需要额外的离散化、桶重建或后处理步骤，而是直接由网络输出完整的条件概率分布 $p(t \mid x)$。对于标准观看时长预估任务，最终使用上述分布的期望 $\hat{t}$ 作为预测结果。与此同时，由于模型已经显式获得了整个观看时长分布，因此还可以进一步计算特定时间区间的累计概率、不同分位点以及预测方差等统计量。例如，可以通过累计分布判断用户快速划走的概率，或者利用不同分位点描述模型对于观看时长预测的不确定性。这使得同一个 EGMN 模型不仅能够服务于传统的点预测任务，也可以为后续的策略制定和多目标融合提供更加丰富的分布信息。

总结起来，EGMN 算法的核心思想可以概括为**“从预测一个观看时长，转向预测一个观看时长分布”**。其通过指数分量来捕获快速划走导致的粗粒度长尾特征，再利用高斯混合分量来建模不同用户和内容形成的细粒度多模态行为，并通过最大似然估计（MLE）、熵正则（Entropy Regularization）和回归损失（Regression Loss）来联合优化。与 CREAD 算法通过离散分桶和生存概率间接恢复观看时长不同，EGMN 算法直接对连续观看时长的概率分布进行参数化，因此能够在保持点预测能力的同时，为工业推荐系统提供更加丰富的分布级信息。但同样需要注意，这种对观看时长分布参数化的方法，也得结合具体业务场景中的数据表现进行分布建模，如果实际数据分布与 EGM 假设差异较大，可能需要进一步调整分布形式或增加更多的分布成分。

23.5 生成式时长预估

除了前文介绍的 D2Q、DVR、D^2Co、CREAD、TPM 和 EGMN 等方法之外，近年来随着生成式推荐的发展，也开始出现将生成式建模思想引入观看时长预估的方法。其核心思路与传统回归模型存在明显差异：传统方法通常直接学习从用户、视频和上下文特征到连续观看时长的映射，而生成式时长预估则尝试将连续观看时长重新表示为一个可生成的离散序列，并利用类似语言模型的序列建模能力逐步生成最终的时长结果。

这种思路一方面可以利用离散 Token 带来的分类学习优势，避免直接拟合长尾连续值；另一方面又能够通过多个 Token 的组合表达较大的时长范围，从而在离散建模和连续数值预测之间建立联系。代表性工作包括快手提出的 Generative Regression（GR）以及进一步采用连续生成建模思想的 FlowTime 等方法。其中，GR 算法将观看时长编码为具有数值含义的 Token 序列，并使用 Transformer Encoder-Decoder 自回归生成该序列，是生成式时长预估的代表性方法。

23.5.1 GR 算法

GR（Generative Regression）算法源自快手团队发表于 WWW 2026 的论文《Generative Regression Based Watch Time Prediction for Short-Video Recommendation》[4]。与传统的观看时长回归（Watch Time Regression）类的方法不同，GR 算法并不直接让模型输出一个连续数值，而是首先构建一个由不同“Time Slot”组成的词表，将连续观看时长编码为多个 Time Slot Token 的序列，再通过 Transformer Decoder 自回归地生成这些 Token，最后将生成的 Token 序列重新转换为连续观看时长。从建模角度来看，GR 算法的关键变化在于将原本的“**特征 → 连续数值回归**”转化为“**特征 → Token 序列生成 → 连续数值恢复**”。因此，GR 算法本质上是一种将回归问题重新表述为序列生成（Sequence Generation）问题的方法。

具体来说，假设训练样本为：

$$\{(\mathbf{u}_i, \mathbf{v}_i, y_i)\}_{i=1}^{N} \tag{23.169}$$

其中 $\mathbf{u}_i$ 和 $\mathbf{v}_i$ 分别表示用户侧和视频侧特征，$y_i \in \mathbb{R}_+$ 表示真实观看时长。GR 算法首先构造一个由不同 Time Slot 组成的词表：

$$\mathcal{V} = \{w_j\}_{j=1}^{V} \tag{23.170}$$

其中每一个 Token 都对应一个具有实际时间含义的数值。进一步定义映射函数 $g(w_j)$，表示 Token w_j 所对应的时长值。随后，GR 算法将连续观看时长 y_i 分解为一个 Token 序列：

$$\mathbf{s}_i = \{s_i^1, \ldots, s_i^{T_i}\} \tag{23.171}$$

并要求该序列能够通过 Token 对应的数值重新恢复原始观看时长，即：

$$y_i \approx \sum_{t=1}^{T_i} g(s_i^t) \tag{23.172}$$

因此，模型最终需要学习的是条件概率：

$$P(\mathbf{s}_i \mid \mathbf{u}_i, \mathbf{v}_i) \tag{23.173}$$

而不是直接预测：

$$P(y_i \mid \mathbf{u}_i, \mathbf{v}_i) \tag{23.174}$$

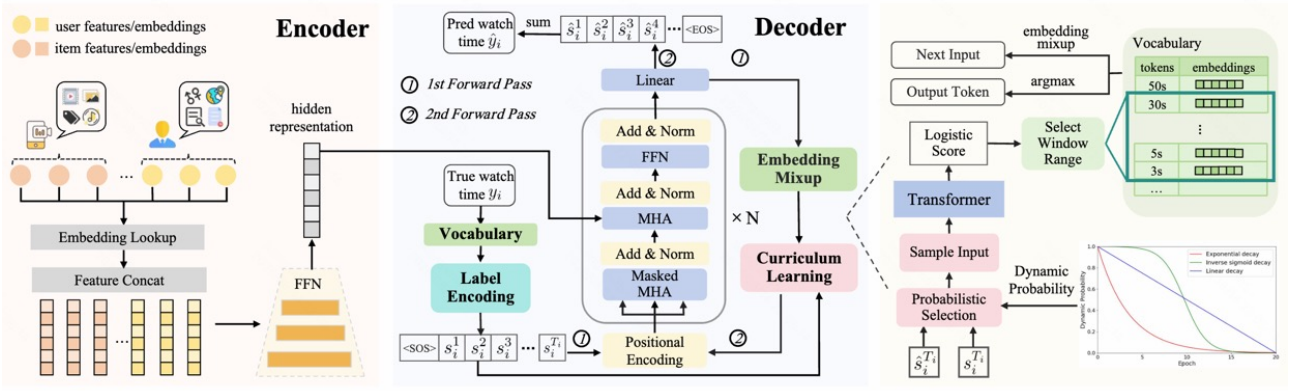

图 23.8: GR 的模型结构示意图。

如图23.8所示，GR 模型采用典型的 Transformer Encoder-Decoder 架构。其中 Encoder 负责将用户、视频以及上下文特征编码为一个固定长度的隐藏表示，Decoder 则以该隐藏表示为条件，自回归地生成观看时长 Token。GR 模型首先将用户侧、视频侧和上下文特征进行 Embedding 以及特征交叉，并得到输入特征 $\mathbf{x}_i = [\mathbf{u}_i; \mathbf{v}_i]$。随后通过一个前馈网络（Feed-Forward Network，FFN）得到共享的隐藏表示：

$$\mathbf{h}_i = g_{\text{encoder}}(\mathbf{x}_i) \tag{23.175}$$

这里的 Encoder 并不限定具体的网络结构，论文中采用 FFN 作为基础实现，但从工业推荐系统的角度来看，也可以替换为 DCN、DIN、Transformer 等更加复杂的特征建模结构。因此，GR 模型的核心创新并不依赖于特定的 Encoder，而在于后续的生成式时长建模方式。

在 Decoder 部分，GR 模型使用 Transformer Decoder 对时长 Token 进行自回归生成。给定 Encoder 输出 $\mathbf{h}_i$ 和已经生成的 Token 序列 $\mathbf{s}_i^{<t}$，模型在第 t 个位置预测下一个 Token：

$$P_\theta(s_i^t \mid \mathbf{h}_i, \mathbf{s}_i^{<t}) \tag{23.176}$$

通过链式法则，整个时长 Token 序列的生成概率可以表示为：

$$P_\theta(\mathbf{s}_i \mid \mathbf{h}_i) = \prod_{t=1}^{T_i} P_\theta(s_i^t \mid \mathbf{h}_i, \mathbf{s}_i^{<t}) \tag{23.177}$$

因此，GR 模型与传统语言模型在形式上具有较强的相似性，只不过语言模型生成的是具有语义含义的词，而 GR 模型生成的是具有明确数值含义的 Time Slot Token。模型在生成过程中使用 $\langle SOS \rangle$、$\langle EOS \rangle$ 和 $\langle PAD \rangle$ 等特殊 Token，其中 $\langle SOS \rangle$ 表示序列开始，$\langle EOS \rangle$ 表示序列结束。

在 GR 模型的设计中，将连续观看时长转化为 Token 序列的关键在于如何设计词表。如果直接按照固定时间间隔构造 Token，例如每 5 秒一个 Token，会导致短时长区域 Token 过于稀疏或者长尾区域 Token 分布严重不均衡；而如果直接从数据中选择若干具有代表性的时长值，又难以保证所有观看时长都能够被准确表示。因此，GR 从三个方面设计 Time Slot 词表。首先是**完整性（Completeness）**，即词表中的 Token 应当能够覆盖数据中的观看时长，并能够以较小误差恢复原始数值；其次是 **平衡性（Balance）**，即不同 Token 的出现频率不能过度失衡，从而避免严重的类别不平衡；最后是 **适应性（Adaptability）**，即词表构造方式应能够适应不同数据集的观看时长分布。

针对观看时长典型的长尾分布，GR 模型提出了基于动态分位点调整（Dynamic Quantile Adjustment，DQA）的数据驱动词表构造方法。其基本思想并不是简单采用固定分位点，而是从较高的分位点开始，逐步降低分位点，并动态调整候选 Time Slot，从而使词表既能够覆盖长尾区域，又能够避免少量极端长时长样本占据过多的 Token 空间。最终得到的词表可以表示为：

$$\mathcal{V} = \{w_1, w_2, \ldots, w_V\} \tag{23.178}$$

并按照对应的数值大小进行排列：

$$g(w_1) > g(w_2) > \cdots > g(w_V) \tag{23.179}$$

这里需要特别注意，GR 算法中的 Token 并不是传统 NLP 中完全离散且没有数值关系的符号。不同 Token 本身具有明确的数值含义，而且相邻 Token 对应的时间值通常也较为接近。因此，这种 Token 空间实际上保留了连续观看时长的部分数值结构。

在得到 Time Slot 词表之后，GR 模型还需要将一个连续观看时长 y_i 编码成 Token 序列。GR 模型希望这种编码过程同时满足三个要求：第一，Token 序列能够以较小误差恢复原始观看时长；第二，在满足恢复精度的情况下尽可能缩短序列长度；第三，Token 按照非递增顺序排列，从而与用户观看过程中逐渐衰减的注意力模式保持一致。因此，对于观看时长 y_i，GR 模型希望找到一个 Token 序列 $\mathbf{s}_i$ 能满足：

$$y_i = \sum_{t=1}^{T_i} g(s_i^t) + \epsilon_i, \qquad |\epsilon_i| \le 0.001 y_i \tag{23.180}$$

同时要求：

$$g(s_i^1) \ge g(s_i^2) \ge \cdots \ge g(s_i^{T_i}) \tag{23.181}$$

在具体实现中，GR 模型采用贪心分解（Greedy Decomposition）的方式进行编码，即从当前能够覆盖的最大 Time Slot 开始，不断从剩余观看时长中减去对应的 Token 值，直到剩余误差足够小。这样，一个连续观看时长就被转换成一组具有加法关系的 Token。例如，如果词表中存在 50、20、10 和 5 秒等 Time Slot，那么一个约 85 秒的观看时长可以被编码为类似 $(50, 20, 10, 5)$ 的 Token 序列。GR 模型最终需要学习的并不是直接输出 85 这个连续数

值，而是学习生成这样的 Token 组合。这种编码方式也是 GR 模型与普通分类式时长建模的重要区别。传统离散化方法通常将一个连续值映射到某一个分桶，而 GR 模型则允许一个观看时长由多个 Token 组合表示，因此可以在有限词表下表达更加丰富的连续数值空间。

在训练阶段，GR 模型首先采用类似语言模型的交叉熵损失函数对 Token 序列进行监督：

$$L_{\text{CE}} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T_i}\log P_\theta(s_i^t \mid \mathbf{h}_i, \mathbf{s}_i^{<t}) \tag{23.182}$$

但是仅优化 Token 生成准确率并不能保证最终恢复出的连续观看时长准确，因此 GR 模型进一步增加 Huber Loss。根据模型生成的 Token 序列，可以将其恢复为连续观看时长：

$$\hat{y}_i = \sum_{t=1}^{\hat{T}_i} g(\hat{s}_i^t) \tag{23.183}$$

然后直接约束 $\hat{y}_i$ 与真实观看时长 y_i 之间的差异：

$$L_{\text{Huber}} = \frac{1}{N}\sum_{i=1}^{N} L_\delta(y_i, \hat{y}_i) \tag{23.184}$$

其中，Huber Loss 在误差较小时采用平方损失，而在误差较大时转变为线性损失，因此相比普通 MSE 对观看时长长尾分布中的极端样本更加鲁棒。最终，GR 模型的训练目标为：

$$L = L_{\text{CE}} + \lambda L_{\text{Huber}} \tag{23.185}$$

其中，λ 用于平衡 Token 生成准确性和最终连续时长预测准确性。可以看到，GR 模型的训练目标实际上同时优化了两个层面的任务：一方面要求模型能够正确生成离散 Token，另一方面又要求这些 Token 组合起来之后能够得到准确的连续观看时长。

由于 GR 模型采用自回归 Decoder，因此训练阶段存在典型的 Teacher Forcing 问题。训练时，模型通常可以直接看到真实的历史 Token，而推理时只能依赖自己之前生成的 Token。当模型前面某一步生成错误 Token 后，这个错误可能继续传递到后续步骤，从而产生曝光偏差（Exposure Bias）。为缓解这一问题，GR 模型引入了 Curriculum Learning with Embedding Mixup（CLEM）。其核心思想是逐渐降低训练阶段对真实 Token 的依赖，并逐步增加模型自身预测 Token 的参与程度。训练初期主要使用真实 Token，使模型能够快速学习基本的序列生成能力；随着训练进行，再逐渐增加预测 Token，使训练条件逐步接近真实推理环境。

在此基础上，GR 模型进一步利用 Token Embedding 之间具有数值连续性的特点设计 Embedding Mixup。由于相邻 Time Slot Token 通常具有相近的数值含义，当模型预测某一个 Token 时，可以同时利用其邻近 Token 的 Embedding，并根据当前 Decoder 输出概率进行加权融合：

$$\mathbf{z}_i^t = \sum_{j=0}^{n_w} \sigma_j \mathbf{E}[\delta_{\hat{s}_i^t} + j - b, :] \tag{23.186}$$

其中，$\mathbf{E}$ 为 Token Embedding 矩阵，n_w 为局部窗口大小，$\delta_{\hat{s}_i^t}$ 表示当前预测 Token 在词表中的位置，σ_j 根据模型对局部 Token 的预测概率进行归一化得到。融合后的 $\mathbf{z}_i^t$ 被用于下一时间步的输入。Embedding Mixup 的直观意义在于：传统分类模型一旦选择了错误 Token，预测结果可能直接跳到一个离正确值较远的离散类别；而 GR 中相邻 Time Slot 具有天然的数值连续性，因此可以利用局部 Token 的概率分布形成更加平滑的数值表示，从而降低单个 Token 分类误差对最终 Watch Time 的影响。

在推理阶段，GR 模型首先通过 Encoder 得到用户、视频以及上下文特征的隐藏表示，然后以 $\langle SOS \rangle$ 作为 Decoder 的起始输入，逐步生成 Watch Time Token。每一步根据当前已经生成的 Token 序列预测下一个 Token，直到生成 $\langle EOS \rangle$ 为止。最终，将生成的 Token 序列重新映射为连续观看时长：

$$\hat{y}_i = \sum_{t=1}^{\hat{T}_i} g(\hat{s}_i^t) \tag{23.187}$$

因此，GR 完成了一个完整的“**连续值 → Token 序列 → 自回归生成 → 连续值**”闭环。从工业部署角度来看，GR 模型与传统回归模型相比最大的变化在于推理过程由一次前向预测变成了多次自回归的 Token 生成，因此其线上延迟与生成序列长度和 Decoder 结构直接相关。不过，由于 GR 模型的词表规模远小于通用语言模型，且生成

的序列通常较短，因此仍然具有在推荐场景中进行轻量化部署的可能性。

总体而言，GR 模型的核心贡献并不是简单地将 Transformer 用于观看时长预估，而是重新设计了连续观看时长的表示方式。它通过具有数值含义的 Time Slot Token 将连续回归问题转化为 Sequence Generation 问题，再利用 Token 的加法结构恢复连续观看时长。相比 CREAD 等基于离散分桶的方法，CREAD 算法更强调通过多个阈值概率近似生存函数，而 GR 算法则直接生成能够组合出观看时长的 Token 序列；相比 EGMN 等显式分布建模方法，GR 算法不需要预先假设观看时长服从某一种参数化概率分布，而是通过生成式模型直接学习观看时长的组合结构。因此，GR 算法代表了观看时长预估从“**数值回归**”向“**离散生成**”演进的一条重要路径。

23.5.2 FlowTime 算法

FlowTime 算法源自于快手科技在 KDD 2026 会议上的论文《FlowTime: Towards Continuous Generative Watch Time Prediction via Flow-based Personalized Priors》[3]。FlowTime 进一步将生成式时长预估从前面 GR 算法所采用的离散 Token 生成拓展到连续空间，核心思想如下：

定义 23.5

FlowTime 算法直接对用户观看时长的条件概率分布进行建模，而不是将观看时长离散化后再进行分类或 Token 生成。

FlowTime 算法主要基于 VAE 模型作为基础生成框架，并进一步利用基于流的个性化先验（Flow-based Personalized Prior）来构建与用户和视频历史观看行为相关的个性化潜变量先验，从而增强模型对 Watch Time 多模态分布和用户行为异质性的建模能力。通常来说，传统的回归模型通常通过 MSE 等损失直接学习一个确定性的 Watch Time 预测值。对于输入特征 $\mathbf{x}$，其优化目标可以表示为：

$$\min_{f} \mathbb{E}_{\mathbf{x},y}\left[(y - f(\mathbf{x}))^2\right] \tag{23.188}$$

该目标的最优解实际上是条件期望：

$$f^*(\mathbf{x}) = \mathbb{E}[y \mid \mathbf{x}] \tag{23.189}$$

因此，当同一个用户和视频特征下存在多个明显不同的观看时长模式时，例如一部分用户快速划走，而另一部分用户完整观看，MSE 倾向于将这些不同模式平均到一起，最终得到一个可能处于低概率密度区域的预测值，这也是传统点回归（Pointwise Regression）中的均值坍缩（Mean Collapse）问题。

前面介绍的 CREAD、TPM 等有序回归方法虽然可以通过多个阈值对 Watch Time 分布进行建模，但本质上仍然依赖离散化。GR 算法进一步将观看时长表示为多个 Token 组成的序列，通过自回归方式生成离散的时长 Token，从而能够显式建模不同 Token 之间的依赖关系。然而，这类方法仍然受到词表构建（Vocabulary Construction）和标签编码（Label Encoding）的限制，即：连续观看时长必须首先被映射到离散 Token 空间，最终预测结果也需要通过 Token 重新恢复。因此，FlowTime 模型进一步提出一个更加直接的思路，即不再对 Watch Time 进行离散化，而是在连续空间中直接学习条件分布 $p(y \mid \mathbf{x})$。

FlowTime 算法将观看时长预估（Watch Time Prediction）问题重新定义为条件概率分布学习问题。假设给定用户特征 u、视频特征 v 以及其他上下文特征 $\mathbf{x} = (u, v)$，目标不再是学习一个确定性的映射 $\hat{y} = f(\mathbf{x})$，而是学习真实的条件分布 $p_{\text{data}}(y \mid \mathbf{x})$。为此，FlowTime 引入潜变量 $\mathbf{z}$，并将条件分布表示为：

$$p_\theta(y \mid \mathbf{x}) = \int p_\theta(y \mid \mathbf{z}, \mathbf{x}) p_\psi(\mathbf{z} \mid \mathbf{x}) d\mathbf{z} \tag{23.190}$$

其中，$p_\psi(\mathbf{z} \mid \mathbf{x})$ 为条件潜变量先验，$p_\theta(y \mid \mathbf{z}, \mathbf{x})$ 为生成式 Decoder。这样，模型可以通过不同的潜变量 $\mathbf{z}$ 生成不同的 Watch Time 结果，从而避免传统回归模型只能输出单点预测的问题。

如图23.9所示，FlowTime 模型采用 VAE 作为基本生成框架。在训练阶段首先通过概率化的编码器来根据真实观看时长 y 和输入特征 $\mathbf{x}$ 推断潜变量 $\mathbf{z}$，随后 Decoder 网络根据 $\mathbf{z}$ 和 $\mathbf{x}$ 重构 Watch Time；在推理阶段则不再使用真实 y，而是直接从条件先验 $p_\psi(\mathbf{z} \mid \mathbf{x})$ 中采样潜变量，并通过 Decoder 生成对应的观看时长。

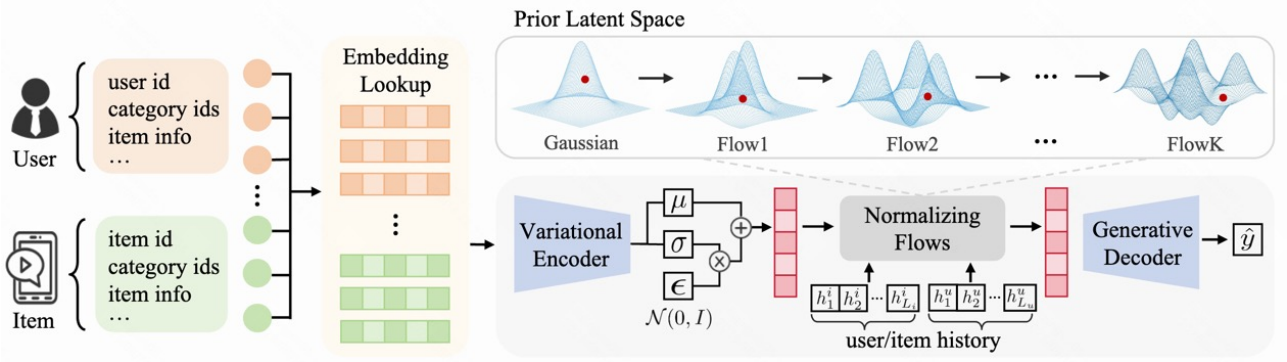

图 23.9: FlowTime 的模型结构示意图。

对于训练样本 $(\mathbf{x}_i, y_i)$，FlowTime 算法首先利用概率化的编码器构造潜变量的近似后验分布：

$$q_\phi(\mathbf{z} \mid y_i, \mathbf{x}_i) = \mathcal{N}(\mathbf{z}; \mu_\phi(y_i, \mathbf{x}_i), \sigma_\phi^2(y_i, \mathbf{x}_i) I) \tag{23.191}$$

其中，μ_ϕ 和 σ_ϕ 由 MLP 进行预测。为了使随机采样过程能够进行端到端反向传播，采用 VAE 经典的重参数化技巧（Reparameterization Trick）：

$$\mathbf{z} = \mu_\phi(y_i, \mathbf{x}_i) + \sigma_\phi(y_i, \mathbf{x}_i) \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I) \tag{23.192}$$

随后，生成式解码器（Generative Decoder）根据潜变量 $\mathbf{z}$ 和用户、视频特征 $\mathbf{x}_i$ 生成 Watch Time，并将条件概率建模为高斯分布：

$$p_\theta(y_i \mid \mathbf{z}, \mathbf{x}_i) = \mathcal{N}(y_i; \mu_\theta(\mathbf{z}, \mathbf{x}_i), \sigma_\theta^2(\mathbf{z}, \mathbf{x}_i)) \tag{23.193}$$

其中，μ_θ 和 σ_θ 由 Decoder 网络预测。在实际进行 Watch Time 预测时，可以直接使用生成分布的期望作为最终预测结果，即：

$$\hat{y}_i = \mu_\theta(\mathbf{z}, \mathbf{x}_i) \tag{23.194}$$

标准 VAE 通常将潜变量先验简单设置为固定的标准高斯分布 $p(\mathbf{z}) = \mathcal{N}(0, I)$。这种设计虽然简单，但对于观看时长预估这个任务而言存在明显限制，因为不同用户、不同视频以及不同历史行为模式对应的观看时长分布可能具有很强的异质性。使用所有用户共享的单一高斯先验，会限制潜变量空间的表达能力，使模型容易产生过度平滑的预测。为此，FlowTime 算法提出基于流的个性化先验，来根据用户和视频历史观看行为动态构建条件先验 $p_\psi(\mathbf{z} \mid \mathbf{x})$。其核心可以理解为：先从一个简单的基础分布中采样潜变量，再利用 Normalizing Flow 对潜变量空间进行连续的非线性变换，使原本简单的高斯分布逐渐变成与当前用户和视频观看模式相匹配的复杂分布。

具体而言，FlowTime 算法首先从用户和视频的历史 Watch Time 中提取多个分位数：

$$q^u = \{q_1^u, \ldots, q_L^u\}, \qquad q^v = \{q_1^v, \ldots, q_L^v\} \tag{23.195}$$

相比直接使用完整的历史观看序列，分位数表征更加关注历史观看时长的整体分布形状。例如，一个用户的观看时长主要集中在短视频区间，还是具有明显的长时长观看行为，都可以通过分位数序列得到较好的刻画。随后，FlowTime 算法使用 Transformer 分别对用户和视频的分位数序列进行编码，并通过 Mean Pooling 得到对应的分布表示：

$$h^u = \text{MeanPool}(\text{Transformer}(\text{Proj}(q^u) + P)), \qquad h^v = \text{MeanPool}(\text{Transformer}(\text{Proj}(q^v) + P)) \tag{23.196}$$

最终将二者拼接得到个性化先验的条件信息：

$$h_{\text{prior}} = [h^u \| h^v] \tag{23.197}$$

在此基础上，FlowTime 算法利用条件归一化流（Conditional Normalizing Flow，CNF）对基础潜变量 z_0 进行多次可逆变换：

$$z_k = f_k(z_{k-1}; h_{\text{prior}}), \quad k = 1, \ldots, K \tag{23.198}$$

论文中采用 Conditional Planar Flow 作为具体实现，其基本形式为：

$$z_k = z_{k-1} + u_k(h_{\text{prior}}) \tanh(w_k(h_{\text{prior}})^\top z_{k-1} + b_k(h_{\text{prior}})) \tag{23.199}$$

在经过多次变换后得到最终潜变量 $\mathbf{z}_K$，并根据 Normalizing Flow 的 Change of Variables 定理计算其概率密度：

$$\log p_\psi(\mathbf{z}_K \mid \mathbf{x}) = \log p_0(\mathbf{z}_0) - \sum_{k=1}^{K} \log \left| \det \frac{\partial f_k}{\partial \mathbf{z}_{k-1}} \right| \tag{23.200}$$

因此，FlowTime 算法不再让所有用户共享一个固定的标准高斯先验，而是根据用户和视频历史观看行为，将基础潜变量空间动态变换为不同的个性化潜变量分布。这样，潜变量中的不同区域可以对应不同的观看时长模式，为多模态 Watch Time 分布提供了更加灵活的表示空间。

FlowTime 算法的训练目标主要包含三个部分，分别用于保证预测准确性、潜变量空间稳定性以及生成分布与历史行为分布的一致性。首先，在 Watch Time 空间中采用 Huber Loss 作为准确性锚（Accuracy Anchor），直接约束生成结果与真实观看时长之间的误差：

$$L_{\text{base}} = L_{\text{Huber}}(y, \hat{y}) \tag{23.201}$$

相比 MSE 损失函数，Huber Loss 在误差较小时具有类似平方损失的优化特性，而在面对长尾 Watch Time 中的极端样本时具有更好的鲁棒性。其次，在潜变量空间中使用 KL 散度来约束 Encoder 得到的后验分布，使其保持良好的连续性和可识别性：

$$L_{\text{latent}} = D_{\text{KL}}(q_\phi(\mathbf{z}_0 \mid y, \mathbf{x}) \| \mathcal{N}(0, I)) \tag{23.202}$$

FlowTime 算法进一步利用用户和视频历史观看时长的分位数表征，对模型生成的 Watch Time 分布进行分布对齐（Distribution Alignment）。具体而言，从基于流的个性化先验中采样多个潜变量并生成对应的 Watch Time 样本，然后将生成样本的顺序统计量（Order Statistics）与用户、视频历史 Watch Time 的分位数进行对齐：

$$L_{\text{W-Dist}} = \frac{1}{L} \sum_{l=1}^{L} \left[\lambda \left| \hat{y}^{(l)} - q_l^u \right| + (1-\lambda) \left| \hat{y}^{(l)} - q_l^v \right| \right] \tag{23.203}$$

其中，λ 用于控制用户侧和视频侧历史分布的影响。该损失使模型生成的 Watch Time 不仅要接近当前样本的真实 Label，同时还要与用户和视频自身的历史观看时长分布保持一致。最终，FlowTime 算法将上述目标组合为：

$$L = \lambda_1 L_{\text{base}} + \lambda_2 L_{\text{latent}} + L_{\text{W-Dist}} \tag{23.204}$$

因此，FlowTime 实际上同时优化了三个层面的目标：观测空间中的预测准确性、隐空间中的分布稳定性以及 Watch Time 分布层面的行为模式一致性。

在模型推理阶段，FlowTime 模型不再需要真实 Watch Time 作为编码器的输入。首先根据当前用户和视频的历史观看分布构建 h_{prior}，然后从基础分布中采样潜变量，并经过基于流的个性化先验来得到最终潜变量：

$$\mathbf{z}_0 \sim p_0(\mathbf{z}), \qquad \mathbf{z}_K = f_K(\cdots f_2(f_1(\mathbf{z}_0; h_{\text{prior}}); h_{\text{prior}}) \cdots ; h_{\text{prior}}) \tag{23.205}$$

随后，将 $\mathbf{z}_K$ 与当前用户、视频特征 $\mathbf{x}$ 输入生成式解码器，得到最终的 Watch Time 预测值：

$$\hat{y} = \mu_\theta(\mathbf{z}_K, \mathbf{x}) \tag{23.206}$$

如果进行多次采样，还可以得到一组 Watch Time 预测结果，从而进一步计算预测分布、分位数以及不确定性等统计信息。

总的来说，FlowTime 算法的核心可以概括为“**VAE 连续生成 + Flow 个性化先验 + 历史分布对齐**”。与前面的 GR 算法相比，GR 算法首先将连续 Watch Time 离散为 Token，再通过 Transformer 进行自回归生成，而 FlowTime 算法则直接在连续潜变量空间中进行生成，避免了词表构建和标签编码带来的量化误差。与传统回归模型相比，FlowTime 模型也不再强制所有可能的观看行为收敛到一个条件均值，而是通过个性化的潜变量分布保留不同用户和视频对应的多模态观看模式。因此，从方法演进角度来看，FlowTime 算法代表了观看时长预估从“**连续值点估计**”到“**离散序列生成**”，再进一步走向“**连续概率分布生成**”的一种探索。

23.6 工业界时长预估实践

在工业界推荐系统中，时长预估一直是一个非常重要且经典的建模任务。尤其是在短视频、直播和信息流推荐场景中，用户的观看时长通常是最直接、最密集的行为反馈之一，因此推荐系统往往会围绕观看时长构建大量的预估目标，并进一步将其与点击、点赞、评论、关注、转发等其他用户行为进行多目标融合。相比 CTR 等相对明确的二分类目标，时长预估的难点在于其本身是一个连续值，并且同时受到视频 Duration、用户兴趣、内容质量、观看场景以及播放机制等多种因素影响。因此，在工业实践中，真正困难的往往并不是选择一个更加复杂的网络结构，而是如何定义一个合理的时长 Label，使其能够尽可能准确地表达用户真实的观看兴趣。

从实际工程经验来看，时长 Label 的定义会直接影响模型的训练目标、样本分布、Loss 设计以及最终的线上排序效果。一个看似简单的 Label 修改，例如调整一个播放时长阈值，或者重新划分 Duration 区间，都可能导致模型学习到完全不同的用户偏好。因此，时长预估通常不是一个“模型训练完成就结束”的任务，而是一个需要持续进行数据分析、Label Review、模型迭代和线上实验的长期优化过程。

23.6.1 基于阈值的时长 Label 定义

工业界最常见的一类时长 Label 是基于业务经验构造的阈值化 Label。其基本思想是将连续的 playing_time 根据 Duration 或者绝对播放时长划分为若干区间，并定义用户是否完成某种程度的观看。例如，可以定义 EVR、LVR、SVR、FPR 等不同的播放行为 Label，不同 Label 分别刻画用户是否进行了有效观看、较长观看、短时观看或者完整播放等行为。

下面给出一些典型的时长 Label 定义示例，如表23.5所示。例如，EVR 通过同时考虑视频完整播放和绝对观看时长，试图避免单纯使用播放完成率导致短视频天然占优的问题；LVR 则通过 Duration 相关的分段阈值定义较长观看行为，使不同 Duration 的视频能够使用不同的观看标准；SVR 主要用于描述较短的播放行为，而 FPR 则更加接近传统意义上的播放完成行为。

表 23.5: 典型的阈值化时长 Label 定义示例。

Label	定义
EVR	$playing\_time \geq 12000$
LVR	$duration \in [0, 3s]$: $playing\_time > 10s$ $duration \in [3s, 40s]$: $playing\_time > 0.8 \cdot duration$ $duration > 40s$: $playing\_time > 50s$
SVR	$0 < playing\_time \leq 3s$
FPR	$duration > 3s \wedge playing\_time \geq duration$

这类 Label 的优点是非常直观，并且具有较强的业务可解释性。模型最终预测的并不是一个抽象的连续数值，而是一个具有明确业务含义的用户行为。例如，预测 EVR 可以理解为“用户是否进行了有效观看”，预测 FPR 则可以理解为“用户是否完成了视频播放”。因此，这类 Label 非常适合直接参与 CTR、CVR 以及其他行为目标的多目标融合。

但阈值化 Label 也存在一个非常明显的问题，即 Label 本身具有较强的人工规则属性。不同阈值对应着不同的用户行为定义，而最优阈值通常并不是固定不变的。随着用户行为、内容生态以及推荐策略发生变化，原有阈值可能逐渐失效。例如，当短视频整体时长发生变化后，原来的 3 秒、7 秒、18 秒或者 40 秒阈值可能已经无法准确区分不同程度的用户兴趣。因此，在工业实践中，基于阈值的时长 Label 通常需要定期进行 Review，并通过 SQL 和离线数据分析重新观察不同 Duration 下的播放行为分布。当数据分布发生明显变化时，重新定义 Label 本身就可能带来一定的线上收益。

23.6.2 渐进式分桶与分布型时长 Label

除了直接定义二值 Label 之外，工业界也经常将连续播放时长进行分桶，将其转化为多个有序的时长区间，从而让模型学习更加细粒度的观看行为。例如，可以将播放时长按照不同区间划分为多个桶，并通过分类或者

序数建模的方式预测用户最终落在哪个时长区间。这类方法可以理解为对传统二分类时长 Label 的一种细化。

其中比较典型的是 WTD（Watch Time Distribution）类 Label，即对播放时长进行渐进式分桶建模。相比简单的 SVR、LVR 等二值 Label，WTD 可以保留更多的观看时长信息，使模型能够区分“观看了几秒”和“观看了几十秒”之间的差异。在实际推荐系统中，这类 Label 也比较适合与其他行为目标进行组合，例如同时预测点击、点赞以及不同等级的观看时长。

然而，WTD 存在一个非常核心的问题，即原始播放时长受到视频 Duration 的强烈影响。对于两个用户而言，即使他们对两个视频的真实兴趣程度完全不同，只要视频 Duration 不同，其绝对播放时长也可能存在非常明显的差异。例如，一个用户观看一个 60 秒视频 30 秒，另一个用户观看一个 10 秒视频 9 秒，单纯比较播放时长时前者显然更高，但从用户兴趣的角度来看，后者实际上可能具有更高的观看完成程度。如果直接使用 WTD 进行排序，模型很容易学习到“长视频更值得观看”的 Shortcut，从而产生明显的 Duration Bias。

因此，工业实践中会进一步尝试对 WTD 进行改进，例如通过引入 Duration 相关的信息，对不同 Duration 下的播放时长进行重新建模，以降低原始播放时长中的 Duration Bias。其核心思想并不是简单地预测“用户看了多长时间”，而是希望模型学习“在当前 Duration 条件下，用户看到了多大程度的时长”，从而让不同 Duration 的视频具有更加合理的可比性。

除了直接分桶之外，还可以进一步从播放时长的条件分布中构造分位数 Label。例如，EVR_p60 可以拟合当前不同 Duration 下播放时长的 P60 曲线，新定义的 LVR_p70 和 LVR_p80 则分别拟合不同 Duration 条件下播放时长的 P70 和 P80 曲线。与固定阈值相比，这类 Label 更加依赖当前数据分布，可以随着 Duration 与 playing time 关系的变化动态调整，因此在实际业务中具有较强的适应性。

从这个角度来看，工业界时长 Label 实际上存在一个比较明显的演进过程：从最初的固定阈值二分类 Label，逐渐发展到 Duration 相关的分段 Label，再进一步发展到分桶、分位数以及条件分布建模。其核心目标都是希望在保留用户观看时长信息的同时，降低 Duration Bias 对模型的干扰。

23.6.3 时长 Label 设计中的数据分析

在真正进行时长预估优化时，Label 设计往往需要大量的数据分析工作，而不是简单地凭经验确定几个阈值。一个典型的工业实践流程是先通过 SQL 统计不同 Duration、不同用户群体以及不同内容垂类下的播放时长分布，然后进一步观察 Duration 与播放时长之间的关系。例如，可以按照 Duration 将视频划分为若干区间，并分别统计每个区间的播放时长均值、中位数、P50、P60、P80、P90、P95 等统计量。通过这些统计量，可以直接观察不同 Duration 下用户观看行为的变化趋势。如果不同 Duration 对应的播放时长曲线存在明显的非线性关系，那么简单的固定阈值 Label 通常就很难准确表达用户兴趣。

与此同时，还需要按照用户群体进行进一步切分。例如，新用户与老用户、高活跃用户与低活跃用户、不同年龄段用户以及不同消费能力用户的观看行为可能存在明显差异。对于同一个视频，一个高活跃用户可能快速划走，而一个低活跃用户可能观看更长时间。因此，如果直接在全量样本上定义统一的时长阈值，很容易将不同用户群体的行为差异混合在一起。

内容侧同样如此。不同视频垂类的 Duration 和观看行为通常存在天然差异。例如，知识类视频、娱乐类视频、剧情类视频以及新闻类视频的合理观看时长并不相同。如果所有内容都采用相同的时长阈值，模型可能会将内容类型本身带来的观看行为差异错误地解释为用户兴趣。因此，在工业实践中，Duration、用户群体和内容垂类通常都是时长 Label 分析时非常重要的切分维度。

进一步地，还需要关注数据分布随时间的变化。推荐系统中的用户行为和内容生态并不是静态的，视频整体 Duration 可能发生变化，用户平均观看深度可能发生变化，甚至推荐策略本身也可能改变用户的观看行为。因此，过去通过数据分析得到的最优阈值，并不一定能够长期保持有效。这也是为什么工业界通常需要持续进行 Label Review，通过周期性的离线分析发现 Label 分布漂移，并根据新的数据分布重新调整 Label。

23.6.4 长尾分布与异常时长处理

时长预估还有一个非常典型的问题，就是播放时长通常具有明显的长尾分布。大部分用户可能只观看几秒到几十秒，但少量用户可能产生远高于平均水平的长时长观看行为。如果直接使用原始播放时长进行回归，这些极端样本可能对模型训练产生非常大的梯度影响。

因此，在工业实践中通常需要对时长进行合理的截断、归一化或者分桶处理。例如，可以对特别长的播放时长设置最大值，将超过阈值的样本统一截断到某个上限；也可以采用 log 变换压缩长尾分布，使模型在训练过程中不会过度关注少数极端样本。对于需要进行分桶建模的场景，则通常采用非均匀分桶，使短时长区域拥有更加密集的桶，而长时长区域采用更加稀疏的桶。

这种“短区间密、长区间疏”的 Duration 分桶设计在工业级推荐系统中非常常见。原因在于用户从 0 秒观看到 1 秒、2 秒、3 秒之间的行为差异可能非常明显，而从 100 秒观看到 101 秒通常没有那么大的业务意义。因此，时长 Label 的分辨率并不需要在整个数值空间内保持一致，而应该根据用户行为分布以及业务价值进行调整。

对于更加长尾的时长目标，还可以采用分位点分桶。例如，在短时长区域使用较密集的 P50、P60、P70 等分位点，而在长时长区域逐渐降低分辨率，同时对极端长时长进行截断。这样既能够保留大多数用户的细粒度观看信息，又能够避免少量异常样本对模型造成过大的影响。

23.6.5 多维时长信号的联合建模

在实际推荐系统中，用户的时长行为并不只有单次视频播放时长。除了最直接的播放时长之外，视频之间的间隔时长、用户在视频上的回滑时长、侧边栏停留时长、评论区停留时长以及用户进入视频作者个人页后的停留时长等，都可以作为非常重要的用户兴趣信号。这些行为虽然都可以归类为“时长”，但其业务含义存在明显差异。例如，视频播放时长更加直接地反映用户对当前内容的观看程度，而评论区停留时长可能更多反映用户对内容讨论的兴趣，作者主页停留时长则可能反映用户对创作者本身的兴趣。因此，在实际建模中通常不会简单地将这些时长全部相加，而是分别定义不同的时长 Label 和预测目标，再通过多任务学习或者多目标融合的方式进行联合建模。

对于这些时长目标，同样需要进行合理的分桶和归一化。例如，可以针对不同的行为类型分别设置合理的最大时长，对特别长的样本进行截断，并采用不同的分位点进行分桶。模型训练阶段则可以根据目标的分布特点选择不同的 Loss，包括 MSE、Huber Loss、WLR 以及其他针对长尾分布设计的损失函数。

其中，WLR 等加权回归方法的核心思想是根据不同 Label 区域的重要程度调整样本权重，使模型不会因为大量短时长样本的存在而忽略长时长区域的预测精度。在多目标时长建模中，还需要进一步考虑不同目标之间的量纲和数值范围差异，因此通常需要先对各个时长相关 XTR 进行归一化，再进行多目标融合，否则数值较大的时长目标可能在联合优化过程中占据过大的权重。

23.6.6 从单点时长预测到分布建模

随着工业推荐系统的发展，时长预估也正在从传统的单点回归逐渐向更加细粒度的分布建模方向发展。传统 MSE 回归最终学习的是条件期望，即模型试图回答“用户平均会观看多长时间”。但对于真实的推荐场景而言，同一个用户在观看同一个视频时，其行为本身可能具有较大的不确定性。例如，一个视频可能存在两种明显的观看模式：一部分用户快速划走，而另一部分用户会完整观看。此时直接使用 MSE 训练模型，最终得到的预测值可能落在两个模式之间，而这个平均值本身在真实数据中可能几乎不存在。因此，相比单点预测，更合理的方向是直接建模 $P(playing\_time|user, video)$，或者预测不同分位点下的观看时长。

这也是近年来时长建模逐渐从“Label Engineering”向“Distribution Modeling”演进的重要原因。分位数预测、离散分布预测、混合密度模型以及 Flow-based 生成式时长建模等方法，都试图从不同角度刻画完整的观看时长分布，而不仅仅是输出一个期望值。对于推荐系统而言，获得完整的时长分布后，还可以根据具体业务目标选择不同的统计量进行排序，例如使用期望观看时长、P80 观看时长或者某个风险敏感的统计量，从而使模型预测结果与最终业务目标之间建立更加灵活的联系。

当然，分布建模并不意味着传统的 Label 工程已经失去价值。在实际工业系统中，固定阈值 Label、分桶 Label、分位数 Label 与连续分布建模往往是共存的。不同 Label 具有不同的业务含义，也对应不同的模型优化目标。例如，EVR 更适合描述有效观看行为，FPR 更适合描述播放完成行为，而连续时长分布则更加适合描述用户观看深度。最终如何组合这些目标，仍然需要结合具体业务进行实验验证。

23.6.7 时长预估的持续迭代与实践经验

总体来看，工业界的时长预估并不存在一种可以适用于所有场景的“一招鲜”解决方案。时长 Label、模型结构、Loss 函数以及最终排序目标之间存在非常强的耦合关系。一个 Label 在离线 AUC 或者 MSE 指标上表现更好，并不意味着它一定能够带来线上收益；同样，一个看似更加复杂的分布建模方法，也不一定能够在实际推荐系统中超过一个经过长期迭代的简单阈值 Label。

因此，在实际工程优化中，通常需要形成一个持续迭代的闭环：首先通过 SQL 和数据分析理解当前用户的观看行为以及 Duration 与播放时长之间的关系；随后根据业务目标设计合适的时长 Label，并通过离线实验验证 Label 的区分能力；在此基础上选择合适的模型和 Loss 进行训练，再通过线上 A/B 实验观察真实推荐效果。如果线上收益不符合预期，则需要进一步回到 Label 定义和数据分布本身进行分析，而不是简单地认为模型结构不够复杂。

尤其需要注意的是，Duration Bias 几乎贯穿整个时长预估过程。从 Label 定义、样本构造，到模型训练和最终排序，都可能受到视频 Duration 的影响。如果不进行处理，模型很容易学习到“长视频天然拥有更高观看时长”的 Shortcut，最终导致推荐系统过度偏好长视频。因此，工业界的时长优化通常需要同时关注绝对观看时长、相对观看深度以及 Duration-conditioned 的行为分布，并根据具体业务选择合适的目标进行建模。

从整体演进来看，工业界时长建模大致经历了从固定阈值 Label、Duration 相关阈值 Label，到分桶 Label、分位数 Label，再到连续值回归和条件分布建模的逐步演进过程。其本质并不是简单追求更加复杂的模型，而是不断回答一个核心问题：

笔记 什么样的观看行为才真正能够代表用户对内容的兴趣？

因此，在实际推荐系统中，往往不会只用一种时长预估方法，而是会综合采用多种阈值化 Label、绝对时长回归、分位数预测以及分布建模方法，并进一步与 CTR、点赞、评论、关注等行为目标进行多目标融合。与此同时，随着用户行为、内容生态以及推荐策略不断变化，时长 Label 本身也需要持续 Review 和重新定义。可以说，时长预估并不是一个一次性的模型建设任务，而是一个贯穿推荐系统数据分析、Label 设计、模型训练、目标融合和线上优化的长期工程问题。对于推荐算法工程师而言，真正重要的也并非找到一个固定的“最佳时长 Label”，而是建立一套能够持续发现 Duration Bias、识别用户行为变化并快速迭代时长目标的工程化方法论。

23.7 本章小结

本章围绕工业界推荐系统中的时长预估问题，对时长 Label 设计、经典建模方法、离散化建模以及生成式时长建模进行了系统介绍。时长预估看似是一个简单的连续值预测问题，但实际观看时长同时受到视频 Duration、用户兴趣、内容质量以及观看行为等多种因素影响，因此，如何定义合理的时长 Label，以及如何降低 Duration Bias 对用户兴趣建模的干扰，一直是工业界时长建模中的核心问题。

首先，本章从工业界常见的阈值化时长 Label 出发，介绍了 Duration-aware 的时长阈值定义、基于比例的 Duration-aware 时长阈值定义、基于 Duration 分桶的经验阈值以及多个时长阈值的联合建模，并结合具体实现案例分析了不同阈值 Label 在实际推荐系统中的应用。随后介绍了经典的 Weight LR 方法，从加权二分类的角度对连续观看时长进行建模，并进一步与基于 MSE 的直接时长回归进行了对比，分析了二者在 Label 定义、样本权重以及长尾时长建模方面的差异。

在此基础上，本章进一步介绍了离散化时长建模方法，包括 One-Hot Label、高斯 Soft Label、自适应分桶以及基于分位点的 Soft Label 等方法。相比直接对连续时长进行回归，离散化方法能够将复杂的连续值预测转化为

分类或者分布预测问题，并通过 Soft Label 缓解相邻时长区间之间的边界不连续问题。同时，自适应分桶和分位点建模能够更加充分地利用真实时长分布，在长尾的观看时长场景下具有较好的实践价值。

随后，本章回顾了经典的时长预估方法，包括 D2Q、DVR、D^2Co、OR、CREAD、TPM 和 EGMN 等。其中，D2Q 通过 Duration 分桶与分位数变换降低 Duration Bias，DVR 通过构造 WTG 并结合对抗学习进一步消除 Duration 信息对用户兴趣预测的影响，D^2Co 则同时考虑 Duration Bias 和 Noisy Watching，通过潜在观看时长分布估计与敏感度控制的校正函数，从原始观看时长中分离更加接近用户真实兴趣的监督信号。OR、CREAD、TPM 和 EGMN 等方法则从序数建模、概率建模以及多阶段时长建模等不同角度进一步拓展了时长预估的建模范式。

随着生成式建模逐渐应用于推荐系统，时长预估也开始从传统的点估计和离散化建模向连续分布建模演进。本章进一步介绍了生成式推荐与时长预估结合的代表性方法，包括 GR 和 FlowTime。其中，生成式方法不再局限于预测单一的期望观看时长，而是尝试建模用户观看时长背后的条件分布，从而更好地刻画真实观看行为中的多模态性和不确定性。尤其是 FlowTime 通过连续生成建模和个性化先验，使时长预估从传统的 “预测一个值” 逐渐转向 “学习一个分布”，为后续更加精细的时长建模提供了新的思路。

最后，本章结合工业界实际实践，对时长预估中的 Label 设计、数据分析、长尾处理、Duration Bias 以及多维时长信号进行了总结。在真实业务中，时长预估并不存在一种可以适用于所有场景的 “一招鲜” 方案。时长 Label 的设计需要结合用户行为、内容生态、Duration 分布以及具体业务目标持续进行 Review 和迭代，同时还需要通过 SQL 分析不同用户群体和内容垂类下的观看时长分布，及时发现数据分布变化。对于长尾明显的观看时长，还需要结合截断、归一化、分桶、分位点以及合适的 Loss 进行处理。

总体而言，工业界时长预估正在经历从固定阈值 Label、Duration-aware Label、离散化时长建模，到分位数预测和连续分布建模的逐步演进。不同方法并不是相互替代的关系，而是可以根据具体业务目标进行组合。在实际推荐系统中，通常需要综合采用多种时长 Label 和预测方法，并进一步与点击、点赞、评论、关注、转发等行为目标进行多目标融合，最终将模型预测目标与推荐系统整体优化目标结合起来。对于推荐算法工程师而言，时长预估的核心并不只是选择一个更加复杂的模型，而是持续理解用户观看行为，合理定义能够表达用户真实兴趣的时长目标，并通过 Label、模型、Loss 和线上排序目标的协同迭代，不断提升推荐系统的整体性能和用户体验。

23.8 参考文献

[1] Paul Covington, Jay Adams, and Emre Sargin. “Deep neural networks for youtube recommendations”. In: *Proceedings of the 10th ACM conference on recommender systems*. 2016, pp. 191–198.

[2] Xiao Lin et al. “Tree based progressive regression model for watch-time prediction in short-video recommendation”. In: *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2023, pp. 4497–4506.

[3] Hongxu Ma et al. “FlowTime: Towards Continuous Generative Watch Time Prediction via Flow-based Personalized Priors”. In: *arXiv preprint arXiv:2606.01352* (2026).

[4] Hongxu Ma et al. “Generative regression based watch time prediction for short-video recommendation”. In: *Proceedings of the ACM Web Conference 2026*. 2026, pp. 6183–6193.

[5] Jie Sun et al. “Cread: A classification-restoration framework with error adaptive discretization for watch time prediction in video recommender systems”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 38. 8. 2024, pp. 9027–9034.

[6] Ruohan Zhan et al. “Deconfounding duration bias in watch-time prediction for video recommendation”. In: *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining*. 2022, pp. 4472–4481.

[7] Haiyuan Zhao et al. “Uncovering user interest from biased and noised watch time in video recommendation”. In: *Proceedings of the 17th ACM Conference on Recommender Systems*. 2023, pp. 528–539.

[8] Xu Zhao et al. “Multi-granularity distribution modeling for video watch time prediction via exponential-gaussian mixture network”. In: *Proceedings of the Nineteenth ACM Conference on Recommender Systems*. 2025, pp. 309–318.

[9] Yu Zheng et al. “Dvr: micro-video recommendation optimizing watch-time-gain under duration bias”. In: *Proceedings of the 30th ACM International Conference on Multimedia*. 2022, pp. 334–345.