

设计一个推荐系统

MLE 算法指北

2025 年 3 月 2 日

1 System Scope

推荐系统的目标是为用户提供个性化内容，提高用户体验和业务指标（如点击率、转化率等）。本设计适用于：

- 电商（如 Amazon、Taobao）推荐商品
- 视频平台（如 YouTube、Netflix）推荐内容
- 社交媒体（如 TikTok、Facebook）推荐帖子

2 Challenges

推荐系统面临以下主要挑战：

- **冷启动问题 (Cold Start)**：新用户/新物品的数据不足
- **数据稀疏性 (Data Sparsity)**：用户行为数据较少，矩阵填充困难
- **实时性要求 (Low Latency)**：需要在毫秒级响应用户请求
- **规模化问题 (Scalability)**：应对亿级用户和海量物品
- **多样性和新颖性 (Diversity & Novelty)**：避免推荐单一热门内容

3 System Architecture

推荐系统通常采用 **离线 + 在线** 结合架构，主要包含：

- 数据收集与存储与特征工程（离线 & 在线计算）
- 候选集生成（定向与召回）
- 精排模型（粗精排 Ranking）
- 实时服务 (Serving)
- 监控与优化

4 推荐系统的数据存储与 Embedding 计算

在推荐系统中，不同类型的数据（交互数据、Item 元数据、Context 数据）需要不同的存储和查询方式，以满足实时性和扩展性要求。此外，这些数据通常需要经过 Embedding 计算，以便供召回和排序模型使用。

4.1 数据分类及特点

推荐系统中的主要数据类型如下：

- **交互数据 (Interaction Data)**：记录用户的点击、浏览、购买、收藏等行为，数据量大，需要支持高效的历史查询和实时更新。
- **Item 元数据 (Item Metadata)**：包括商品 ID、类别、品牌、价格、发布时间等信息，通常更新频率较低，但需要支持快速查询。
- **Context 数据 (Context Data)**：用户当前的设备、时间、地点、天气等上下文信息，通常需要实时查询，以提升推荐的相关性。

不同数据的访问模式、存储方式如下表：

数据类型	规模	存储方式	查询方式
交互数据	海量 (TB-PB 级)	HDFS / Kafka / Redis	ClickHouse / Redis
Item 元数据	大规模 (GB-TB 级)	MySQL / Redis / Elasticsearch	低延迟 KV 查询
Context 数据	小规模 (MB 级)	Redis / Kafka	低延迟缓存查询

表 1: 推荐系统中的数据存储方式

4.2 交互数据的存储与查询

交互数据用于建模用户行为，存储方式包括：

- **离线存储：**使用 HDFS / Hive / Delta Lake 存储完整的用户行为数据，供离线训练模型使用。
- **实时存储：**
 - Kafka / Pulsar：用于流式日志存储，支持实时计算。
 - ClickHouse / Druid (OLAP)：用于高效查询用户过去 N 天的行为数据。
 - Redis / Cassandra：存储用户最近的点击记录，支持毫秒级查询。

查询方式：

- 离线模型训练时，使用 Spark / Flink 读取 HDFS 数据，计算用户特征。
- 在线推荐时：
 - 先查询 Redis，获取用户最近 N 次点击记录。
 - 若 Redis 未命中，则查询 ClickHouse 获取长期行为数据。

4.3 Item 元数据的存储与查询

Item 元数据用于描述物品信息，存储方式包括：

- **MySQL / PostgreSQL：**存储完整的商品信息，支持 CRUD 操作。
- **Elasticsearch：**支持全文搜索，例如根据商品名称进行模糊匹配。
- **Redis / HBase / DynamoDB：**缓存热门商品的信息，提高查询效率。

查询方式：

- 先查 Redis 获取缓存的热门商品信息，若未命中，则查询 MySQL。
- 复杂查询（如模糊匹配）使用 Elasticsearch。

4.4 Context 数据的存储与查询

Context 数据用于捕捉用户的实时环境信息，存储方式包括：

- **Redis / Memcached：**存储用户最近的上下文信息（如 GPS 位置）。
- **Kafka / Flink State Store：**存储实时事件数据（如天气、热点事件）。
- **ClickHouse / Druid：**用于存储统计分析结果，如当前时间段的热门商品。

查询方式：

- 直接从 Redis 查询用户 session 相关的 context 信息。
- 订阅 Kafka 流，获取最新的实时事件。

5 Candidate Generation (定向与召回)

定向和召回层的目标是从海量物品中快速筛选出较小集合。

5.1 定向检索 (Targeting Retrieval)

定向检索 (Targeting Retrieval) 是指基于用户画像、业务需求等条件，在召回阶段筛选出符合展示条件的物品，从而提高推荐的精准度和转化率。定向检索一般包括以下三个主要逻辑：

- **定向检索逻辑 (Targeting Logic)：**根据用户特征匹配合适的物品集合。
- **频控逻辑 (Frequency Control)：**限制特定物品或类别的展示频率，防止用户产生疲劳。
- **业务筛选逻辑 (Business Filtering Logic)：**剔除不适合当前用户或业务策略的物品，例如已购买商品、低库存商品等。

5.1.1 定向检索逻辑 (Targeting Logic)

定向检索的核心思想是基于用户特征选择适合的物品，通常采用如下方式实现：

- 基于用户属性的定向：匹配用户的性别、年龄、地域、兴趣标签等，例如：

$$\text{Targeted Items}(u) = \{i \mid \mathbf{A}_i \cap \mathbf{A}_u \neq \emptyset\} \quad (1)$$

其中， \mathbf{A}_i 表示物品 i 的属性集合， \mathbf{A}_u 表示用户 u 的属性集合。

- 基于用户行为的定向：利用用户的历史点击、收藏、购买等行为数据，选择相关性较高的物品，例如：

$$S(u, i) = \sum_{j \in \text{history}(u)} \text{sim}(i, j) \quad (2)$$

其中， $\text{sim}(i, j)$ 代表物品 i 和 j 之间的相似度。最后对所有定向候选交集即可。

5.1.2 频控逻辑 (Frequency Control)

频控逻辑的目的是防止某些物品过度曝光，避免用户产生疲劳。常见的频控策略包括：

- 物品曝光频次控制：

$$N(i, u) < N_{\max} \quad (3)$$

其中， $N(i, u)$ 表示物品 i 在用户 u 处的展示次数， N_{\max} 是最大允许展示次数。

- 类别曝光频次控制：

$$\sum_{i \in C} N(i, u) < N_{C_{\max}} \quad (4)$$

其中， C 表示某个类别的商品集合， $N_{C_{\max}}$ 是该类别的最大允许展示次数。

- 时间窗口控制：- 限制用户在短时间内看到同样的物品，例如：

$$T_{\text{last}}(i, u) - T_{\text{now}} > T_{\text{threshold}} \quad (5)$$

其中， T_{last} 代表用户上次看到物品 i 的时间， T_{now} 是当前时间， $T_{\text{threshold}}$ 是最小间隔时间。

- 动态冷却策略：- 如果用户多次对某个物品无反应，则降低该物品的展示概率：

$$P_{\text{next}}(i, u) = P_{\text{init}}(i, u) \times e^{-\lambda N(i, u)} \quad (6)$$

其中， λ 是衰减系数， $P_{\text{init}}(i, u)$ 是初始推荐概率。

5.1.3 业务筛选逻辑 (Business Filtering Logic)

业务筛选逻辑主要用于剔除 ** 不适合展示的物品 **，保证推荐内容符合业务需求。常见的筛选逻辑包括：

- 已购买商品过滤：- 如果用户已购买某商品，则避免重复推荐：

$$\text{Filtered Items} = \{i \mid i \in \text{Purchased}(u)\} \quad (7)$$

- 低库存商品过滤：- 对于库存低于一定阈值的商品，不予推荐：

$$S(i) > S_{\min} \quad (8)$$

其中， $S(i)$ 表示物品 i 的当前库存量。

- 价格/折扣策略：- 过滤掉不符合用户消费能力的商品：

$$P_{\min}(u) < P(i) < P_{\max}(u) \quad (9)$$

其中， $P(i)$ 是物品 i 的价格， $P_{\min}(u)$ 和 $P_{\max}(u)$ 代表用户 u 的消费能力范围。

- 品牌/类别排除：- 根据用户的负反馈，排除特定品牌或类别的商品：

$$i \notin \text{Blocked Categories}(u) \cup \text{Blocked Brands}(u) \quad (10)$$

5.1.4 实现方式

- 数据库 + 索引查询**：索引查询，快速筛选符合定向条件的物品。
- 预计算缓存**：对于用户画像较稳定的定向规则（如性别、年龄、地域），可以提前计算定向推荐池，存入 Redis，提高查询效率。
- 在线计算 + 过滤**：对于实时性较高的定向规则（如短期兴趣、时间频控），可以在在线服务层（如 Flink + Redis）进行动态计算和过滤。

5.2 召回算法与模型训练

召回（Recall）是推荐系统中的**第一层筛选**，其主要目标是**从海量物品集合中快速筛选出一个较小的候选集**，以供排序（Ranking）模型进一步优化。在实际应用中，可以采用**传统召回方法**（如协同过滤、矩阵分解等）或**深度学习召回模型**（如 YouTube DNN、Transformer-based 召回）。

5.2.1 召回算法（Retrieval Algorithms）

召回算法主要分为 **基于规则的方法** 和 **基于模型的方法**。

(1) **规则召回（Rule-based Retrieval）** 适用于冷启动或无用户行为数据的情况：

- 热门召回（Popular-based）**：推荐全局最热门的物品。
- 新物品召回（Newest-based）**：推荐最新上架的物品。
- 地域召回（Geo-based）**：基于用户的 GPS 位置推荐附近的物品。
- 业务规则召回（Business Rules）**：如促销商品、品牌曝光策略等。

(2) **协同过滤召回（Collaborative Filtering, CF）** 基于用户-物品交互行为数据，计算用户和物品的相似性：

$$\text{sim}(u, v) = \frac{\sum_i r_{ui} r_{vi}}{\sqrt{\sum_i r_{ui}^2} \sqrt{\sum_i r_{vi}^2}} \quad (11)$$

$$\text{sim}(i, j) = \frac{\sum_u (r_{ui} - \bar{r}_u)(r_{uj} - \bar{r}_u)}{\sqrt{\sum_u (r_{ui} - \bar{r}_u)^2} \cdot \sqrt{\sum_u (r_{uj} - \bar{r}_u)^2}} \quad (12)$$

(3) **矩阵分解（Matrix Factorization, MF）**

$$R \approx U \cdot V^T \quad (13)$$

其中：

- U ：用户的 Embedding
- V ：物品的 Embedding

优化目标：

$$L = \sum (r_{ui} - U_u V_i^T)^2 + \lambda(\|U\|^2 + \|V\|^2) \quad (14)$$

(4) **深度学习召回（Deep Learning Retrieval）** 采用**双塔（Two-tower）结构**，计算用户和物品的 Embedding：

$$P(i|u) = \frac{e^{\mathbf{E}_u \cdot \mathbf{E}_i}}{\sum_j e^{\mathbf{E}_u \cdot \mathbf{E}_j}} \quad (15)$$

5.2.2 召回 Embedding 的存储与在线检索

训练好的 User Embedding 和 Item Embedding 需要存储并用于在线召回：

1. 存储方式：

- 离线存储：HDFS / S3。
- 在线存储：Redis / HBase / Faiss。

2. 检索方式：ANN (Approximate Nearest Neighbor)：

- HNSW Hierarchical Navigable Small World 基于图结构的 ANN 方法，适用于高维数据
- IVF (Inverted File Index) 将 Embedding 空间分为多个聚类中心，加速最近邻查找。
- LSH (Locality-Sensitive Hashing) 使用哈希映射将相似向量映射到相同的桶，提高检索速度。
- Faiss Facebook AI Similarity Search Facebook 开源的 ANN 库，支持 HNSW/IVF/LSH。

6 Ranking Model (精排)

精排模型用于对召回结果进一步排序，核心目标是优化业务指标：

6.1 常见模型

- Logistic Regression (线性基准模型)
- GBDT (Gradient Boosting Decision Trees)：如 XGBoost、LightGBM
- Deep Learning Models：
 - Wide & Deep：结合线性和深度特征
 - DIN/DIEN：考虑用户兴趣演化
 - Transformer-based Models：如 BERT4Rec

6.2 排序损失函数 (Loss Functions for Ranking)

在推荐系统和学习排序 (Learning to Rank, LTR) 任务中，损失函数的选择直接影响模型的优化效果。常见的排序损失函数包括：

- Pointwise 损失函数：针对单个样本计算误差，适用于回归或分类任务。
- Pairwise 损失函数：基于成对样本计算相对排序关系，优化两两比较的正确性。
- Listwise 损失函数：对整个排序列表计算损失，全局优化排名顺序。

6.2.1 Pointwise 损失函数 (逐点损失)

交叉熵损失 (Binary Cross Entropy, BCE)

$$L = - \sum_{i=1}^N (y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)) \quad (16)$$

其中， y_i 表示用户是否点击 (1 表示点击，0 表示未点击)。

优缺点：

- 计算简单，可直接优化评分或点击率预测。
- 不能直接优化排序目标，可能导致排序质量下降。

6.2.2 Pairwise 损失函数 (对比损失)

Pairwise 方法优化 ** 两个物品的相对排序关系 **，目标是让模型正确学习 ** 更相关的物品应该排在无关物品之前 **。

(1) **BPR Loss (Bayesian Personalized Ranking)** BPR (贝叶斯个性化排序) 是一种常用于 ** 推荐系统 ** 的 Pairwise 损失函数:

$$L = - \sum_{(u,i,j) \in D} \log \sigma(\hat{y}_{ui} - \hat{y}_{uj}) \quad (17)$$

其中:

- \hat{y}_{ui} 是用户 u 对物品 i 的预测得分。
- \hat{y}_{uj} 是用户 u 对物品 j 的预测得分 (i 是正样本, j 是负样本)。
- $\sigma(x) = \frac{1}{1+e^{-x}}$ 是 **sigmoid 函数 **，用来控制排序的概率。

(2) **Hinge Loss (合页损失)**

$$L = \sum_{(u,i,j) \in D} \max(0, 1 - (\hat{y}_{ui} - \hat{y}_{uj})) \quad (18)$$

- 强制要求 ** 两个物品的得分差距至少为 1 **，否则损失为 0。

优缺点:

- 直接优化排序目标，适用于 ** 隐式反馈数据 (点击、购买等) **。
- 需要负采样，计算开销较大。

6.2.3 Listwise 损失函数 (列表损失)

Listwise 方法直接优化 ** 整个排序列表 **，是最符合排序任务的优化方式。

(1) **ListNet (Softmax Cross Entropy Loss)**

$$P(y_i) = \frac{e^{y_i}}{\sum_{j=1}^N e^{y_j}}, \quad \hat{P}(\hat{y}_i) = \frac{e^{\hat{y}_i}}{\sum_{j=1}^N e^{\hat{y}_j}} \quad (19)$$

$$L = - \sum_{i=1}^N P(y_i) \log \hat{P}(\hat{y}_i) \quad (20)$$

其中:

- $P(y_i)$ 是 ** 真实排名概率 **， $\hat{P}(\hat{y}_i)$ 是 ** 模型预测的排名概率 **。

(2) **LambdaRank (基于 NDCG 的损失)** LambdaRank 直接优化 **NDCG (Normalized Discounted Cumulative Gain) ** 目标:

$$\Delta NDCG = \frac{\sum_i \frac{2^{y_i} - 1}{\log_2(i+1)}}{\sum_i \frac{2^{y_i^*} - 1}{\log_2(i+1)}} \quad (21)$$

- 目标是最大化 ** 高权重的物品 (点击、购买) 排名靠前 **。

优缺点:

- 直接优化排序指标 (如 NDCG)，最终效果更优。
- 计算复杂，对长列表排序梯度计算较难。

7 Online Serving

详见模型部署那篇笔记

8 Monitoring and Metrics

业务指标：

- CTR (点击率)、CVR (转化率)、GMV (总交易额)
- 召回率、用户覆盖率、多样性

系统指标：

- QPS (每秒请求数)
- 99th 延迟 (p99 latency)
- 负载均衡情况

模型指标：

- AUC ranking 能力
- Log Loss CTR 准确性
- 模型预估均值平稳