推荐系统架构

推荐和搜索系统核心的的任务是从海量物品中找到用户感兴趣的内容。在这个背景下，推荐系统包含的模块非常多，每个模块将会有很多专业研究的工程和研究工程师，作为刚入门的应届生或者实习生很难对每个模块都有很深的理解，实际上也大可不必，我们完全可以从学习好一个模块技术后，以点带面学习整个系统，虽然正式工作中我们放入门每个人将只会负责的也是整个系统的一部分。但是掌握推荐系统最重要的还是梳理清楚整个推荐系统的架构，知道每一个部分需要完成哪些任务，是如何做的，主要的技术栈是什么，有哪些局限和可以研究的问题，能够对我们学习推荐系统有一个提纲挈领的作用。

所以这篇文章将会从**系统架构**和**算法架构**两个角度出发解析推荐系统通用架构。系统架构设计思想是大数据背景下如何有效利用海量和实时数据，将推荐系统按照对数据利用情况和系统响应要求出发，将整个架构分为**离线层、近线层、在线层**三个模块。然后分析这三个模块分别承担推荐系统什么任务，有什么制约要求。这种角度不和召回、排序这种通俗我们理解算法架构，因为更多的是考虑推荐算法在工程技术实现上的问题，系统架构是如何权衡利弊，如何利用各种技术工具帮助我们达到想要的目的的，方便我们理解为什么推荐系统要这样设计。

而算法架构是从我们比较熟悉的**召回、粗排、排序、重排**等算法环节角度出发的，重要的是要去理解每个环节需要完成的任务，每个环节的评价体系，以及为什么要那么设计。还有一个重要问题是每个环节涉及到的技术栈和主流算法，这部分非常重要而且篇幅较大，所以我们放在下一个章节讲述。

架构设计是一个非常大的话题，设计的核心在于平衡和妥协。在推荐系统不同时期、不同的环境、不同的数据，架构都会面临不一样的问题。网飞官方博客有一段总结：

> We want the ability to use sophisticated machine learning algorithms that can grow to arbitrary complexity and can deal with huge amounts of data. We also want an architecture that allows for flexible and agile innovation where new approaches can be developed and plugged-in easily. Plus, we want our recommendation results to be fresh and respond quickly to new data and user actions. Finding the sweet spot between these desires is not trivial: it requires a thoughtful analysis of requirements, careful selection of technologies, and a strategic decomposition of recommendation algorithms to achieve the best outcomes for our members. **“我们需要具备使用复杂机器学习算法的能力，这些算法要可以适应高度复杂性，可以处理大量数据。我们还要能够提供灵活、敏捷创新的架构，新的方法可以很容易在其基础上开发和插入。而且，我们需要我们的推荐结果足够新，能快速响应新的数据和用户行为。找到这些要求之间恰当的平衡并不容易，需要深思熟虑的需求分析，细心的技术选择，战略性的推荐算法分解，最终才能为客户达成最佳的结果。”**

所以在思考推荐系统架构考虑的第一个问题是确定边界：知道推荐系统要负责哪部分问题，这就是边界内的部分。在这个基础上，架构要分为哪几个部分，每一部分需要完成的子功能是什么，每一部分依赖外界的什么。了解推荐系统架构也和上文讲到的思路一样，我们需要知道的是推荐系统要负责的是怎么问题，每一个子模块分别承担了哪些功能，它们的主流技术栈是什么。从这个角度来阅读本文，将会有助于读者抓住重点。

系统架构

推荐系统架构，首先从数据驱动角度，对于数据，最简单的方法是存下来，留作后续离线处理，**离线层**就是我们用来管理离线作业的部分架构。**在线层**能更快地响应最近的事件和用户交互，但必须实时完成。这会限制使用算法的复杂性和处理的数据量。离线计算对于数据数量和算法复杂度限制更少，因为它以批量方式完成，没有很强的时间要求。不过，由于没有及时加入最新的数据，所以很容易过时。

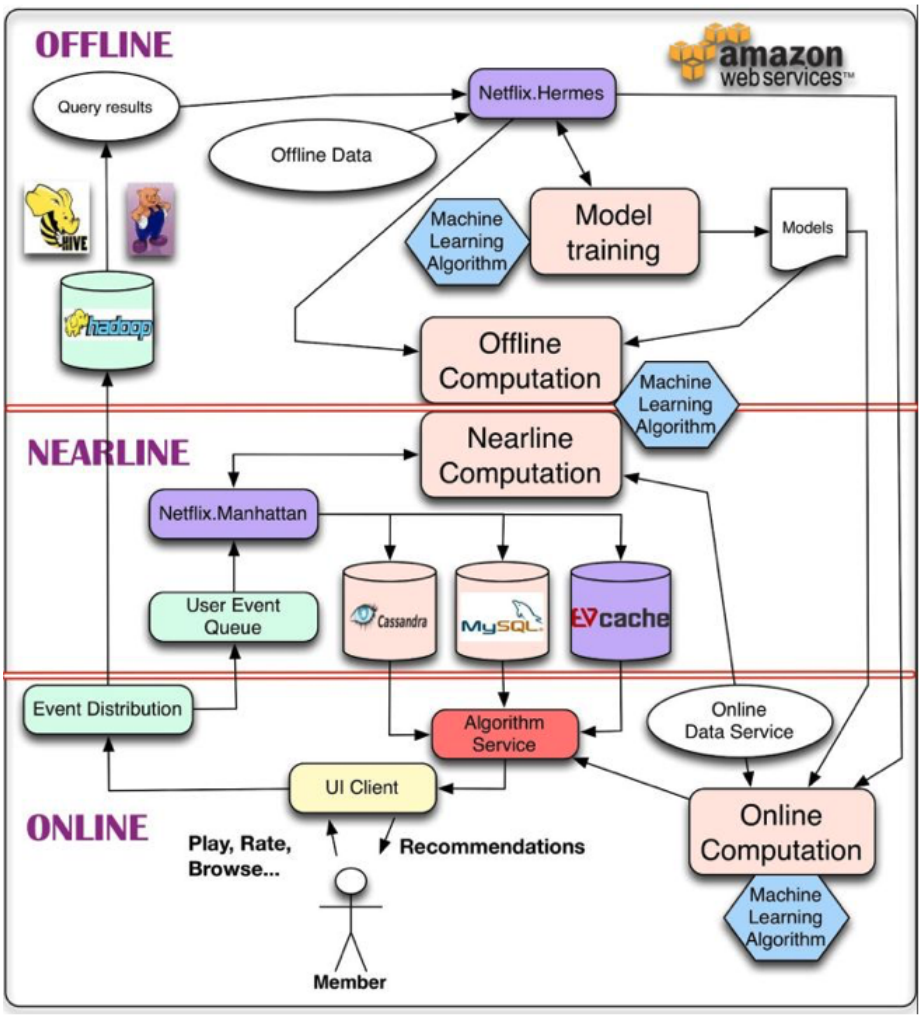
个性化架构的关键问题，就是如何以无缝方式结合、管理在线和离线计算过程。**近线层**介于两种方法之间，可以执行类似于在线计算的方法，但又不必以实时方式完成。这种设计思想最经典的就是Netflix在2013年提出的架构，整个架构分为

1. 离线层：不用实时数据，不提供实时响应；
2. 近线层：使用实时数据，不保证实时响应；
3. 在线层：使用实时数据，保证实时在线服务；

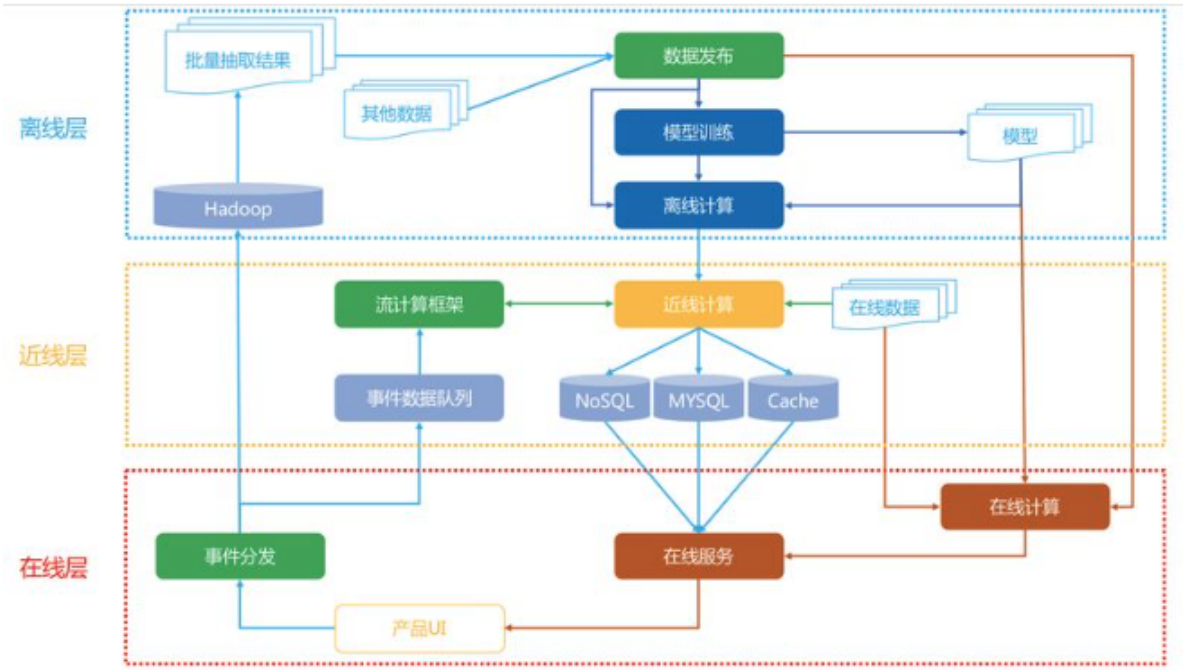
设计思想

网飞的这个架构提出的非常早，其中的技术也许不是现在常用的技术了，但是架构模型仍然被很多公司采用。

这个架构为什么要这么设计，本质上是因为推荐系统是由大量数据驱动的，大数据框架最经典的就是lambda架构和kappa架构。而推荐系统在不同环节所使用的数据、处理数据的量级、需要的读取速度都是不同的，目前的技术还是很难实现一套端到端的及时响应系统，所以这种架构的设计本质上还是一种权衡后的产物，所以有了下图这种模型：



上面是网飞的原图，我搬运了更加容易理解的线条梳理后的结构：



整个数据部分其实是一整个链路，主要是三块，分别是客户端及服务器实时数据处理、流处理平台准实时数据处理和大数据平台离线数据处理这三个部分。

看到这里，一个很直观的问题就是，为什么数据处理需要这么多步骤？这些步骤都是干嘛的，存在的意义是什么？

我们一个一个来说，首先是客户端和服务端的实时数据处理。这个很好理解，这个步骤的工作就是记录。将用户在平台上真实的行为记录下来，比如用户看到了哪些内容，和哪些内容发生了交互，和哪些没有发生了交互。如果再精细一点，还会记录用户停留的时间，用户使用的设备等等。除此之外还会记录行为发生的时间，行为发生的session等其他上下文信息。

这一部分主要是后端和客户端完成，行业术语叫做埋点。所谓的埋点其实就是记录点，因为数据这种东西需要工程师去主动记录，不记录就没有数据，记录了才有数据。既然我们要做推荐系统，要分析用户行为，还要训练模型，显然需要数据。需要数据，就需要记录。

第二个步骤是流处理平台准实时数据处理，这一步是干嘛的呢，其实也是记录数据，不过是记录一些准实时的数据。很多同学又会迷糊了，实时我理解，就是当下立即的意思，准实时是什么意思呢？准实时的意思也是实时，只不过没有那么即时，比如可能存在几分钟的误差。这样存在误差的即时数据，行业术语叫做准实时。那什么样的准实时数据需要记录呢？在推荐领域基本上只有一个类别，就是用户行为数据。也就是用户在观看这个内容之前还看过哪些内容，和哪些内容发生过交互。理想情况这部分数据也需要做成实时，但由于这部分数据量比较大，并且逻辑也相对复杂，所以很难做到非常实时，一般都是通过消息队列加在线缓存的方式做成准实时。

最后我们看第三个步骤，叫做离线数据处理，离线也就是线下处理，基本上就没有时限的要求了。

一般来说，离线处理才是数据处理的大头。所有“脏活累活”复杂的操作都是在离线完成的，比如说一些join操作。后端只是记录了用户交互的商品id，我们需要商品的详细信息怎么办？需要去和商品表关联查表。显然数据关联是一个非常耗时的操作，所以只能放到离线来做。

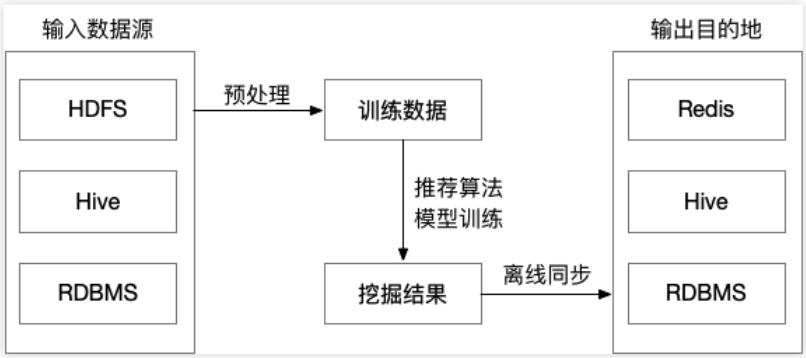
接下来详细介绍一下这三个模块。

离线层

离线层是计算量最大的一个部分，它的特点是不依赖实时数据，也不需要实时提供服务。需要实现的主要功能模块是：

1. 数据处理、数据存储；
2. 特征工程、离线特征计算；
3. 离线模型的训练；

这里我们可以看出离线层的任务是最接近学校中我们处理数据、训练模型这种任务的，不同可能就是需要面临更大规模的数据。离线任务一般会按照天或者更久运行，比如每天晚上定期更新这一天的数据，然后重新训练模型，第二天上线新模型。



离线层优势和不足

离线层面临的数据量级是最大的，面临主要的问题是海量数据存储、大规模特征工程、多机分布式机器学习模型训练。目前主流的做法是HDFS，收集到我们所有的业务数据，通过HIVE等工具，从全量数据中抽取出我们需要的数据，进行相应的加工，离线阶段主流使用的分布式框架一般是Spark。所以离线层有如下的优势：

1. 可以处理大量的数据，进行大规模特征工程；
2. 可以进行批量处理和计算；
3. 不用有响应时间要求；

但是同样的，如果我们只使用用户离线数据，最大的不足就是无法反应用户的实时兴趣变化，这就促使了近线层的产生。

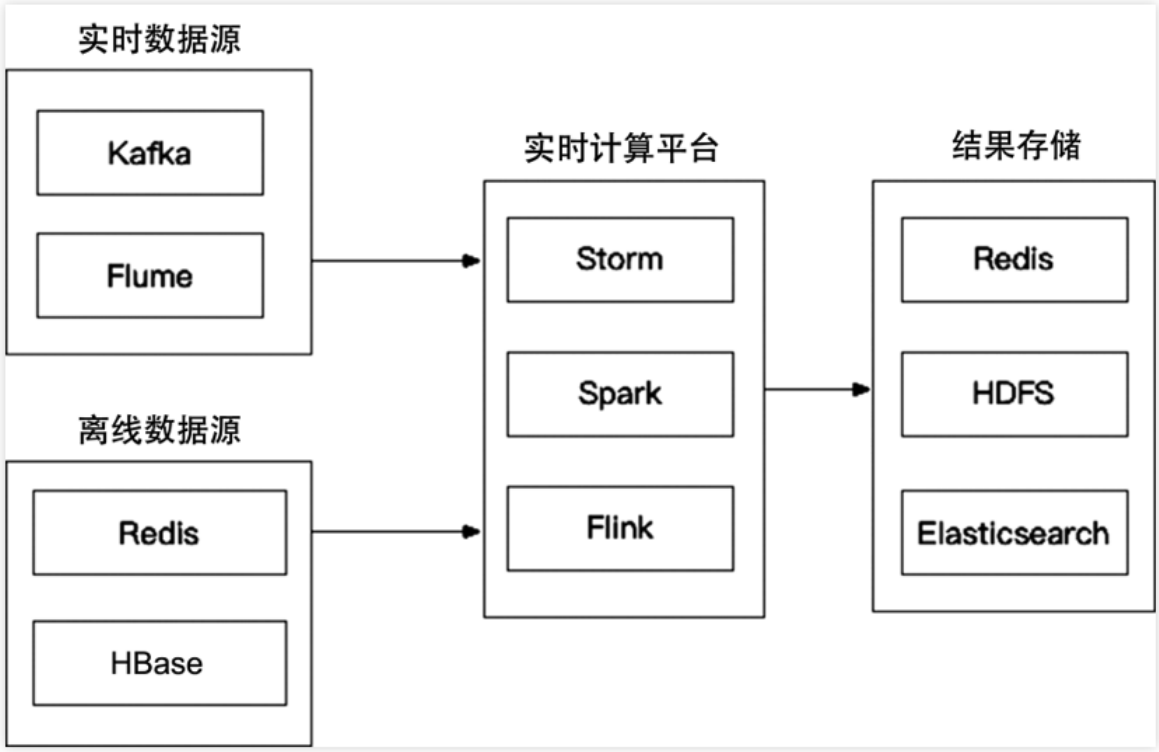
近线层

近线层的主要特点是准实时，它可以获得实时数据，然后快速计算提供服务，但是并不要求它和在线层一样达到几十毫秒这种延时要求。近线层的产生是同时想要弥补离线层和在线层的不足，折中的产物。

它适合处理一些对延时比较敏感的任务，比如：

1. 特征的事实更新计算：例如统计用户对不同type的ctr，推荐系统一个老生常谈的问题就是特征分布不一致怎么办，如果使用离线算好的特征就容易出现这个问题。近线层能够获取实时数据，按照用户的实时兴趣计算就能很好避免这个问题。
2. 实时训练数据的获取：比如在使用DIN、DSIN这行网络会依赖于用户的实时兴趣变化，用户几分钟前的点击就可以通过近线层获取特征输入模型。
3. 模型实时训练：可以通过在线学习的方法更新模型，实时推送到线上；

近线层的发展得益于最近几年大数据技术的发展，很多流处理框架的提出大大促进了近线层的进步。如今Flink、Storm等工具一统天下。



在线层

在线层，就是直接面向用户的的那一层了。最大的特点是对响应延时有要求，因为它是直接面对用户群体的，你可以想象你打开抖音淘宝等界面，几乎都是秒刷出来给你的推荐结果的，不会说还需要让你等待几秒钟时间。所有的用户请求都会发送到在线层，在线层需要快速返回结果，它主要承担的工作有：

1. 模型在线服务；包括了快速召回和排序；
2. 在线特征快速处理拼接：：根据传入的用户ID和场景，快速读取特征和处理；
3. AB实验或者分流：根据不同用户采用不一样的模型，比如冷启动用户和正常服务模型；
4. 运筹优化和业务干预：比如要对特殊商家流量扶持、对某些内容限流；

典型的在线服务是用过RESTful/RPC等提供服务，一般是公司后台服务部门调用我们的这个服务，返回给前端。具体部署应用比较多的方式就是使用Docker在K8S部署。而在线服务的数据源就是我们在离线层计算好的每个用户和商品特征，我们事先存放在数据库中，在线层只需要实时拼接，不进行复杂的特征运算，然后输入近线层或者离线层已经训练好的模型，根据推理结果进行排序，最后返回给后台服务器，后台服务器根据我们对每一个用户的打分，再返回给用户。

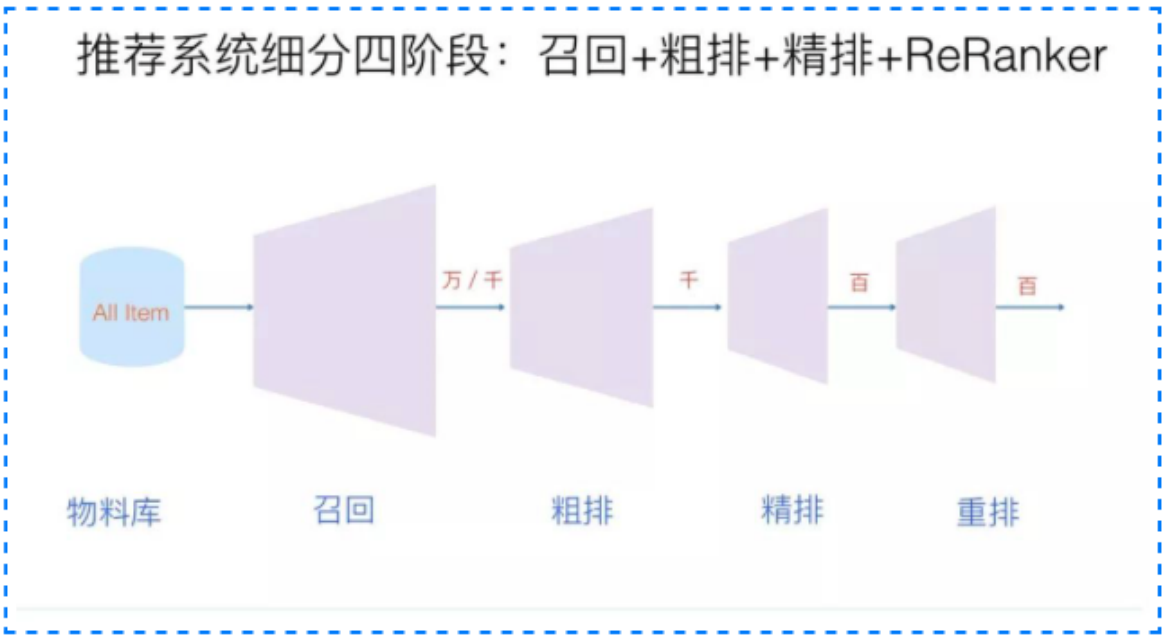
在线层最大的问题就是对实时性要求特别高，一般来说是几十毫秒，这就限制了我们能做的工作，很多任务往往无法及时完成，需要近线层协助我们做。

算法架构

我们在入门学习推荐系统的时候，更加关注的是哪个模型AUC更高、topK效果好，哪个模型更加牛逼的问题，从基本的协同过滤到点击率预估算法，从深度学习到强化学习，学术界都始终走在最前列。一个推荐算法从出现到在业界得到广泛应用是一个长期的过程，因为在实际的生产系统中，首先需要保证的是稳定、实时地向用户提供推荐服务，在这个前提下才能追求推荐系统的效果。

算法架构的设计思想就是在实际的工业场景中，不管是用户维度、物品维度还是用户和物品的交互维度，数据都是极其丰富的，学术界对算法的使用方法不能照搬到工业界。当一个用户访问推荐模块时，系统不可能针对该用户对所有的物品进行排序，那么推荐系统是怎么解决的呢？对应的商品众多，如何决定将哪些商品展示给用户？对于排序好的商品，如何合理地展示给用户？

所以一个通用的算法架构，设计思想就是对数据层层建模，层层筛选，帮助用户从海量数据中找出其真正感兴趣的部分。



- 召回

召回层的主要目标时从推荐池中选取几千上万的item，送给后续的排序模块。由于召回面对的候选集十分大，且一般需要在线输出，故召回模块必须轻量快速低延迟。由于后续还有排序模块作为保障，召回不需要十分准确，但不可遗漏（特别是搜索系统中的召回模块）。

如果没有召回层，每个User都能和每一个Item去在线排序阶段预测目标概率，理论上来说是效果最好，但是是不现实的，线上不延迟允许，所以召回和粗排阶段就要做一些候选集筛选的工作，保证在有限的能够给到排序层去做精排的候选集的时间里，效果最大化。另一个方面就是通过这种模型级联的方式，可以减少用排序阶段拟合多目标的压力，比如召回阶段我们现在主要是在保证Item质量的基础上注重覆盖率多样性，粗排阶段主要用简单的模型来解决不同路的召回和当前用户的相关性问题，最后截断到1k个以内的候选集，这个候选集符合一定的个性化相关性、视频质量和多样性的保证，然后做ranking去做复杂模型的predict。

目前基本上采用多路召回解决范式，分为非个性化召回和个性化召回。个性化召回又有content-based、behavior-based、feature-based等多种方式。

召回主要考虑的内容有：

1. **考虑用户层面**：用户兴趣的多元化，用户需求与场景的多元化：例如：新闻需求，重大要闻，相关内容沉浸阅读等等
2. **考虑系统层面**：增强系统的鲁棒性；部分召回失效，其余召回队列兜底不会导致整个召回层失效；排序层失效，召回队列兜底不会导致整个推荐系统失效
3. **系统多样性内容分发**：图文、视频、小视频；精准、试探、时效一定比例；召回目标的多元化，例如：相关性，沉浸时长，时效性，特色内容等等
4. **可解释性推荐一部分召回是有明确推荐理由的**：很好的解决产品性数据的引入；

- 粗排

粗排的原因是有时候召回的结果还是太多，精排层速度还是跟不上，所以加入粗排。粗排可以理解为精排前的一轮过滤机制，减轻精排模块的压力。粗排介于召回和精排之间，要同时兼顾精准性和低延迟。目前粗排一般也都模型化了，其训练样本类似于精排，选取曝光点击为正样本，曝光未点击为负样本。但由于粗排一般面向上万的候选集，而精排只有几百上千，其解空间大很多。

粗排阶段的架构设计主要是考虑三个方面，一个是根据精排模型中的重要特征，来做候选集的截断，另一部分是有一些召回设计，比如热度或者语义相关的这些结果，仅考虑了item侧的特征，可以用粗排模型来排序跟当前User之间的相关性，据此来做截断，这样是比单独的按照item侧的倒排分数截断得到更加个性化的结果，最后是算法的选型要在在线服务的性能上有保证，因为这个阶段在pipeline中完成从召回到精排的截断工作，在延迟允许的范围内能处理更多的召回候选集理论上与精排效果正相关。

- 精排

精排层，也是我们学习推荐入门最常常接触的层，我们所熟悉的算法很大一部分都来自精排层。这一层的任务是获取粗排模块的结果，对候选集进行打分和排序。精排需要在最大时延允许的情况下，保证打分的精准性，是整个系统中至关重要的一个模块，也是最复杂，研究最多的一个模块。

精排是推荐系统各层级中最纯粹的一层，他的目标比较单一且集中，一门心思的实现目标的调优即可。最开始的时候精排模型的常见目标是ctr,后续逐渐发展了cvr等多类目标。精排和粗排层的基本目标是一致的，都是对商品集合进行排序，但是和粗排不同的是，精排只需要对少量的商品(即粗排输出的商品集合的topN)进行排序即可。因此，精排中可以使用比粗排更多的特征，更复杂的模型和更精细的策略（用户的特征和行为在该层的大量使用和参与也是基于这个原因）。

精排层模型是推荐系统中涵盖的研究方向最多，有非常多的子领域值得研究探索，这也是推荐系统中技术含量最高的部分，毕竟它是直接面对用户，产生的结果对用户影响最大的一层。目前精排层深度学习已经一统天下了，精排阶段采用的方案相对通用，首先一天的样本量是几十亿的级别，我们要解决的是样本规模的问题，尽量多的喂给模型去记忆，另一个方面时效性上，用户的反馈产生的时候，怎么尽快的把新的反馈给到模型里去，学到最新的知识。

- 重排

常见的有三种优化目标：Point Wise、Pair Wise 和 List Wise。重排序阶段对精排生成的Top-N个物品的序列进行重新排序，生成一个Top-K个物品的序列，作为排序系统最后的结果，直接展现给用户。重排序的原因是因为多个物品之间往往是相互影响的，而精排序是根据PointWise得分，容易造成推荐结果同质化严重，有很多冗余信息。而重排序面对的挑战就是海量状态空间如何求解的问题，一般在精排层我们使用AUC作为指标，但是在重排序更多关注NDCG等指标。

重排序在业务中，获取精排的排序结果，还会根据一些策略、运营规则参与排序，比如强制去重、间隔排序、流量扶持等、运营策略、多样性、context上下文等，重新进行一个微调。重排序更多的是List Wise作为优化目标的，它关注的是列表中商品顺序的问题来优化模型，但是一般List Wise因为状态空间大，存在训练速度慢的问题。

由于精排模型一般比较复杂，基于系统时延考虑，一般采用point-wise方式，并行对每个item进行打分。这就使得打分时缺少了上下文感知能力。用户最终是否会点击购买一个商品，除了和它自身有关外，和它周围其他的item也息息相关。重排一般比较轻量，可以加入上下文感知能力，提升推荐整体算法效率。比如三八节对美妆类目商品提权，类目打散、同图打散、同卖家打散等保证用户体验措施。重排中规则比较多，但目前也有不少基于模型来提升重排效果的方案。

- 混排

多个业务线都想在Feeds流中获取曝光，则需要对它们的结果进行混排。比如推荐流中插入广告、视频流中插入图文和banner等。可以基于规则策略（如广告定坑）和强化学习来实现。

总结

整篇文章从系统架构梳理了Netfliex的经典推荐系统架构，整个架构更多是偏向实时性能和效果之间tradeoff的结果。如果从另外的角度看推荐系统架构，比如从数据流向，或者说从推荐系统各个时序依赖来看，就是我们最熟悉的召回、粗排、精排、重排、混排等模块了。这种角度来看是把推荐系统从前往后串起来，其中每一个模块既有在离线层工作的，也有在在线层工作的。而从数据驱动角度看，更能够看到推荐系统的完整技术栈，推荐系统当前面临的局限和发展方向。

召回、排序这些里面单拿出任何一个模块都非常复杂。这也是为什么大家都说大厂拧螺丝的原因，因为很可能某个人只会负责其中很小的一个模块。许多人说起自己的模块来如数家珍，但对于全局缺乏认识，带来的结果是当你某天跳槽了或者是工作内容变化了，之前从工作当中的学习积累很难沉淀下来，这对于程序员的成长来说是很不利的。

所以希望这篇文章能够帮助大家在负责某一个模块，优化某一个功能的时候，除了能够有算法和数据，还能能够考虑对整个架构带来的影响，如何提升整体的一个性能，慢慢开阔自己的眼界，构建出一个更好的推荐系统。

参考资料

- 《从零开始构建企业级推荐系统》
- Netflix
- 回顾经典，Netflix的推荐系统架构
- 大数据处理中的Lambda架构和Kappa架构
- 张俊林：推荐系统技术演进趋势
- 推荐算法架构1：召回/等
- 微信"看一看"多模型内容策略与召回
- 阿里粗排技术体系
- 推荐系统架构与算法流程详解
- 业内推荐系统架构介绍
- 推荐系统笔记，一张图看懂系统架构