腾讯音乐：全民K歌推荐系统架构及粗排设计

原创 全民K歌推荐团队 DataFunTalk 昨天

收录于话题

#原创精选 76 #腾讯音乐 1 #推荐算法 13

分享嘉宾：kevinshuang、fivenwu

编辑整理：张振、于洋

出品平台：DataFunTalk

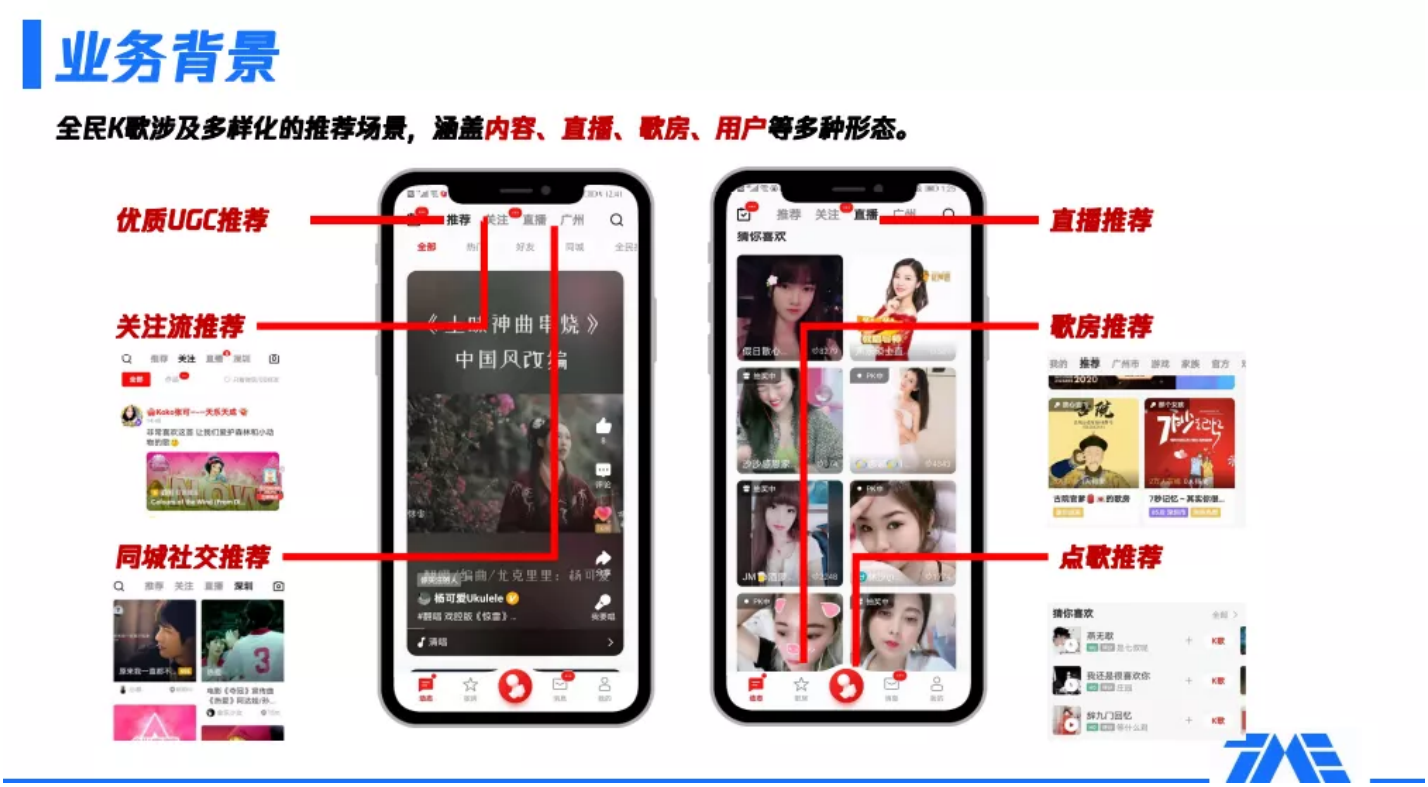
导读：腾讯音乐娱乐集团（TME）目前有四大移动音乐产品：QQ音乐、酷狗音乐、酷我音乐和全民K歌，总月活超8亿。其中，全民K歌与其他三款产品有明显的差异，具体表现如下：以唱为核心，在唱歌的功能上又衍生出了一些音乐娱乐的功能及玩法，目前有超过1.5亿的月活。推荐在全民K歌各个场景中起着重要作用，极大地影响着平台的内容分发状况及生产者与消费者的关系。本文将主要介绍全民K歌的推荐系统架构及粗排设计，具体从以下几方面展开：

- 业务背景

- 推荐系统架构及挑战
- 粗排模块算法设计
- 多样性调节算法设计

01

业务背景



全民K歌涉及多样化的推荐场景，涵盖内容、直播、歌房、用户等多种形态。

具体的推荐功能如上图所示，主要包括以下几类：

① 基于内容的推荐，包括优质UGC推荐、关注流推荐、同城社交推荐等功能模块。

- 优质UGC推荐，将平台原生原创的优质音视频内容进行推荐
- 关注流推荐，对关注的内容进行混排

- 同城社交推荐，基于同城的社交进行推荐

② 除了内容推荐外，我们也会负责一些其他类型的推荐，包括直播推荐、点歌推荐、歌房推荐和点评推荐，都是在K歌生态下独有的推荐。

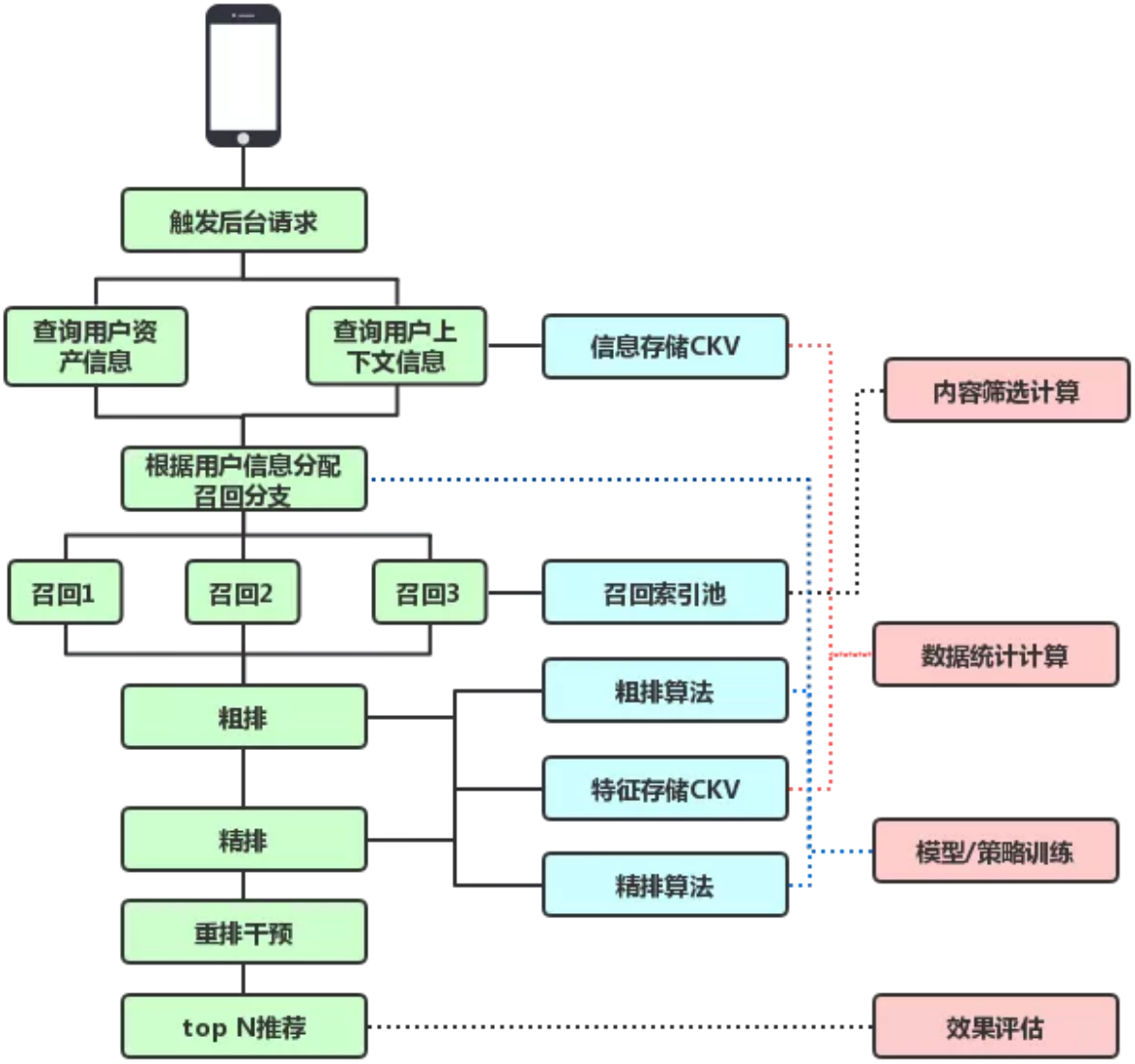
02

推荐系统架构及挑战

我们的推荐系统主要分为四个部分，包括召回层、粗排层、精排层及重排层。

1. 召回层

召回层的作用主要是从海量的item中筛选小量级的用户可能感兴趣的内容。在这个模块中最重要、大家了解最多的肯定就是召回模块本身。



一般来说，我们线上的召回方法会分为索引类的召回、泛社交的召回以及模型的召回。

- 索引类的召回：主要根据画像的结果做精准的ID类召回，比如对用户感兴趣的歌曲进行召回，以及用户感兴趣的创作者进行召回。

- 泛社交的召回：主要基于用户在站内丰富的社交关系，比如用户可能会关注一些其他的用户，也可能加入一些家族，我们会根据这些社团的发现结果做泛化召回。

- 模型的召回：基于模型的召回的方法比较多，在后面会展开介绍。

除了召回模块之外，我们在召回层认为另外一个比较重要的是内容的比例筛选，因为我们整体的内容发表量比较大，每天可能有500万的作品在平台内发表，如何从中筛选出合适的内容，需要基于音视频的理解，以及基于我们自己的流量策略来综合发现。

2. 粗排层

粗排层到精排层相当于是一个承上启下的作用，它会把我们召回到的一些万量级的作品进一步筛选到千量级，既要考虑打分的性能问题，又要考虑排序粗排精准度的问题。一般来说，粗排模型会用一些模型蒸馏的方法，包括特征蒸馏或者基于模型本身的结构蒸馏，以及把这些不同的蒸馏方法组合起来等等。在粗排层，除了做粗排的打分外，我们还会重点做生态的控制，特别在一些内容推荐场景或者直播推荐场景中，我们会注重这里面的内容生态，包括时效性，内容的调性，多样性等等。

3. 精排层

粗排层之后就来到了精排层，精排层主要根据千量级的作品进一步的进行精排打分来筛选到百量级的作品。在精排层，我们主要注意以下几方面：

- 一个就是精排模型本身，我们早期也采用了类似LR的线性模型和LightGBM这样的树模型，之后随着深度模型的技术发展，我们所有的场景都切换到了深度模型，并且随着深度网络的设计复杂度，以及样本规模的逐步增加，让早期基于TF训练的一些深度模型引擎，在训练的速度以及对模型大小规模的限制，已经对我们产生了影响。所以我们现在已经过渡到基于参数服务器框架下，训练深度模型。

- 在精排前，除了模型怎么训练构造外，另外两个比较重要的是特征和样本的构造，以及在这个场景下的多目标设计。特别是多目标的问题，可能还涉及到具体的网络结构如何做，以及最后的结果如何融合。

4. 重排层

从精排层排序出百量级的作品后，就会进入到重排层。重排层会基于业务规则进行进一步的干预，比如同一首歌曲的视频不能连续出现，同一个创作者的视频不能连续出现等等。除了基于规则性的限制外，我们

也会考虑使用模型化的方法做多样性的打散，并且在第4部分，我们也会具体介绍DPP算法的原理和思想。除了模型方法之外，我们也在考虑通过list wise的方法做内容的重排。

03

粗排模块算法设计

1. 粗排模块定位和方案路线

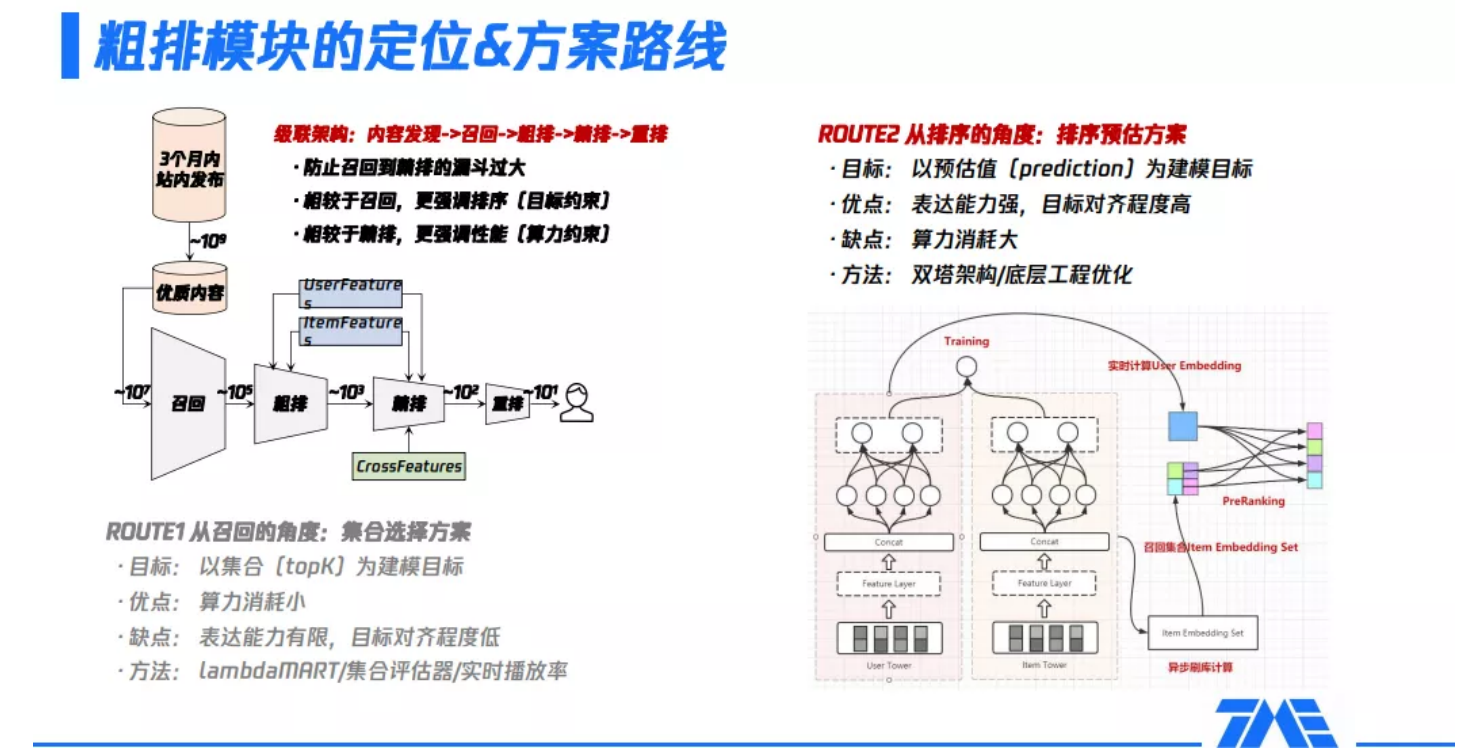
粗排模型和精排模型不同，它可能既需要解决模型预测的准确性，又需要解决模型预测的效率。接下来，为大家介绍我们整个推荐系统如何在线上真实运转。

粗排模块主要包含两部分：

第一部分是粗排的排序部分，这一部分主要就是为了解决一个大规模item的预排序问题。

第二个部分是多样性控制部分，作为一个UGC平台，我们需要考量内容在整个内容生态中的分发状况，以及均衡的考量生产者跟消费者之间的关系。我们是通过多样性调节算法来间接实现这个目的的。

让我们再回顾下前面所提到的整个召回系统架构，我们可以看到它其实是一个典型的节点架构，从内容发现到召回，到粗排到精排，然后重排，最后把合适的内容推荐给用户。由于我们是一个比较大的UGC生产平台，从一个UGC平台的角度来考虑，我们倾向于分发较为新的作品，因为新的作品的分发会为那些活跃的创作者带来一定的流量激励，使得生产端跟消费端产生联动，促进了一种正向的循环。



我们以三个月内的站内发布作品为例，三个月内的站内发布作品大概会到十的九次方量级，然后我们通过一些内容挖掘跟优质内容发现的方式，从中找到更为优质的一部分内容，这部分内容仍然有十的七次方量级，相当于从百万量级的作品库做兴趣的召回。这是一个非常大的候选集场景，从优质内容的发现到召回，是整个UGC平台和推荐系统的连接，背后承载着我们对整个UGC的分发、内容生态的理解。这一部分我们会在第三个环节去做进一步详细的介绍。召回可以看到有十的七次方的候选集召回，通常完成召回过程之后仍然有十的五次方候选集，而精排通常只能去处理一个千级别的规模，所以我们看到这中间存在一个比较大的漏斗，很多时候会成为我们在效果或者一些收益上的瓶颈。这时，粗排模块的重要性就体现出来了。粗排模块的一个重要的作用就是防止召回到精排之间的漏斗过大，导致过多的信息损失。

大家可能比较了解的是，召回跟精排召回更像是一个集合，选择的过程更侧重于选择效率最高的方法，至于topk中item之间的先后顺序，其实不是最关心的因素；而精排环节相对来说它更强调次序，也就是说item A跟item B的先后顺序是非常敏感和关键的。

粗排夹在召回跟精排之间，它的定位是什么？相比于召回，粗排更强调排序性，也就是更强调topk内部的排序关系；相对于精排，粗排更强调性能，因为精排通常有非常复杂的网络结构，非常大的参数量，这也意味着它在实际应用的过程中比较难去处理一个较大规模量级的候选集的打分，这时粗排就必须要解决这

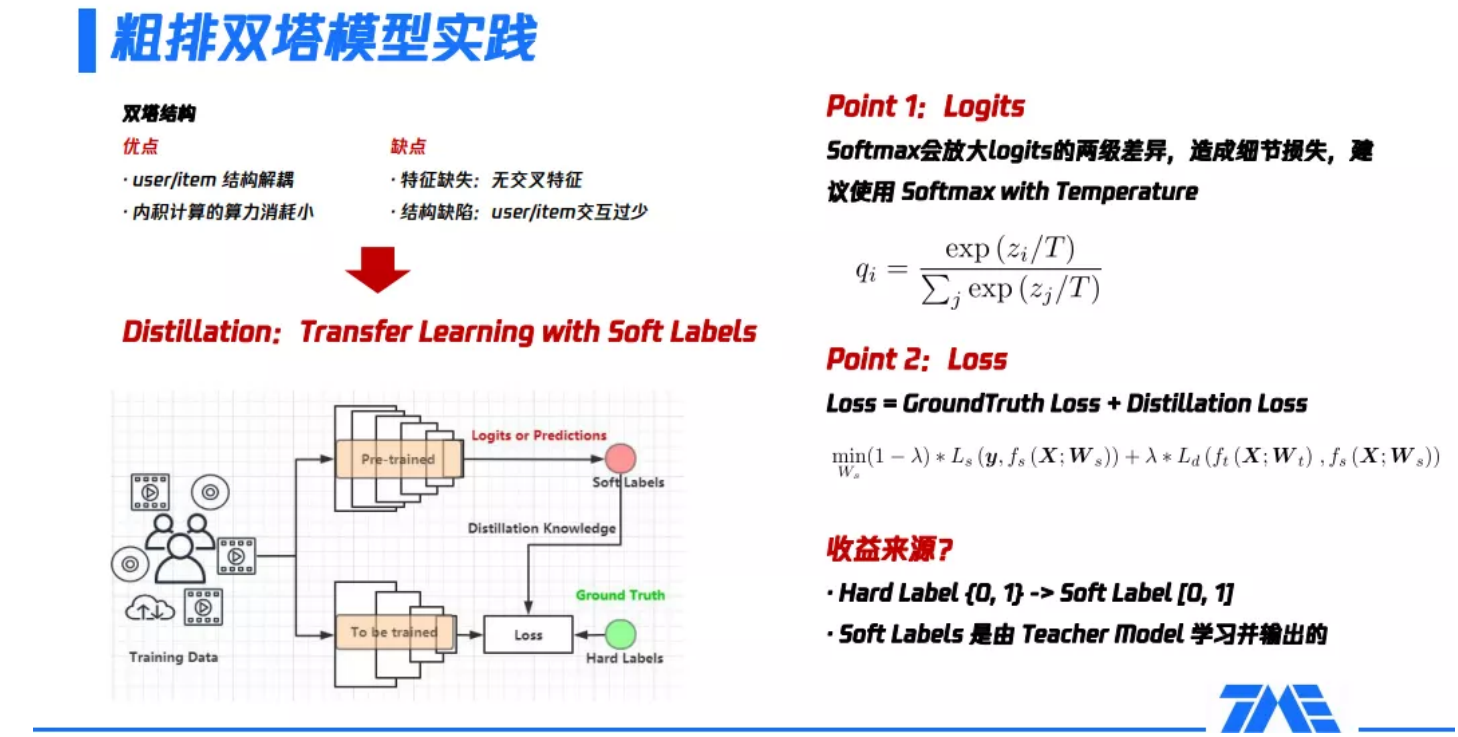
个问题，所以它在性能上会相较于精排有更多的要求。现在主流的粗排方案有两条路线，这两条路线是基于两个出发点来思考的：

第一条路线是把粗排当成是召回的一种延伸，它的技术选型会像是一种集合，选择的方案和目标是选出效率最高的子集。常用的方式，简单的可以通过实时的serving或者一些实时的指标对召回排序的结果做一个选择。总体来说，这条技术路线最大的优点是算力消耗非常小、性能非常好，但是它的缺点是本身的表达能力是有限的，可以明显预估到它的天花板。

第二条路线是从精排的角度，把粗排当成是精排的迁移或压缩，也就是说这是一条排序的路线，它的建模目标是预估值。这种方法的好处是它的表达能力很强，因为通常会用到一些比较复杂的网络结构，而且它跟精排的联动性是更好的，可以让粗排跟精排的目标保持某种程度上的一致性。同时，它的缺点也凸显出来了，就是我们用到了复杂的方法，算力的消耗一定也会相应的提升。因此，需要着重解决的是如何在有限的算力下尽可能地突破表达能力上限。在这种路线下，我们通常会在架构选择上选择双塔结构模型，右下图所示。

接下来，介绍下粗排双塔模型实战相关细节。

2. 粗排双塔模型实践



我们通过把user和item的feature进行结构解耦与分开建模，然后完成整个架构的设计，在模型训练完毕之后，我们会通过user serving实时的产出user embedding，再通过索引服务把该用户所有的候选集合的ID给取出来，最后在user embedding跟item embedding之间做内积的运算，得到一个粗排的预估值，作为整个粗排阶段的排序依据。

这么做的优势是user/item 结构是解耦的，内积计算的算力消耗小。

同时，它的缺点也非常的明显：

- 第一个是它在特征表达上是缺失的，因为user跟item解耦之后，很难使用一些交叉特征，一旦用了交叉特征，有多少item就得进行多少次预估，这违背了我们使用双塔模型的初衷。

- 第二个是它在结构上也是有缺陷的，我们回忆一下上面这幅框架图，可以看到user跟item的交互非常少，这会限制它的表达能力的上限。

如果我们选择了这种技术方案，我们可以保障它的性能，但是我们需要进一步的考虑如何避免这种简易的结构所带来的效果上的损失。在这种情况下，我们通常会使用一些模型蒸馏的方式。

模型蒸馏有两个关键词，第一个是它本质上是一种迁移学习，也就是transfer learning，第二个是transfer的方式是通过所谓的label，区别于我们平时理解的样本label离散值，变为了0到1之间的一个连续值。左下角是一个常见的蒸馏模型架构图，这个里面会有涉及到两个模型，第一个模型叫做教师模型，叫做teacher，第二个模型是学生模型，叫做student。这两个模型的总体思路是teacher模型是一个非常大、非常复杂、学习到的东西非常多的模型，teacher模型会把学习到的知识传导给student模型，受限于某些原因，该模型没有办法做得很复杂，或者它的规模必须限制在一定范围内的子模型。

具体流程如下：

- 准备训练样本，对teacher模型预训练，即得到了teacher模型；
- 把teacher模型最后一层或倒数第二层的输出结果，作为传递给student模型的信息，这部分通常是logits或softmax的形式，也叫做soft labels；
- 把soft labels传导到student模型，作为模型loss的一部分，因为student模型除了要拟合teacher模型传递的信息，也要去拟合样本真实的分布。

针对上述流程里涉及到了几个概念，有如下的进一步解释：

- 关于logits：它是teacher模型层层映射后的一个抽象度很高且信息浓度很大的结果，是整个知识传递的媒介。在logits后，通常会接一个softmax，但是softmax会放大logits的两极差异，造成细节损失，而这些细节恰恰是帮助student模型更好学习的关键。为了克服以上问题，我们选用了改进的softmax，即softmax with temperature

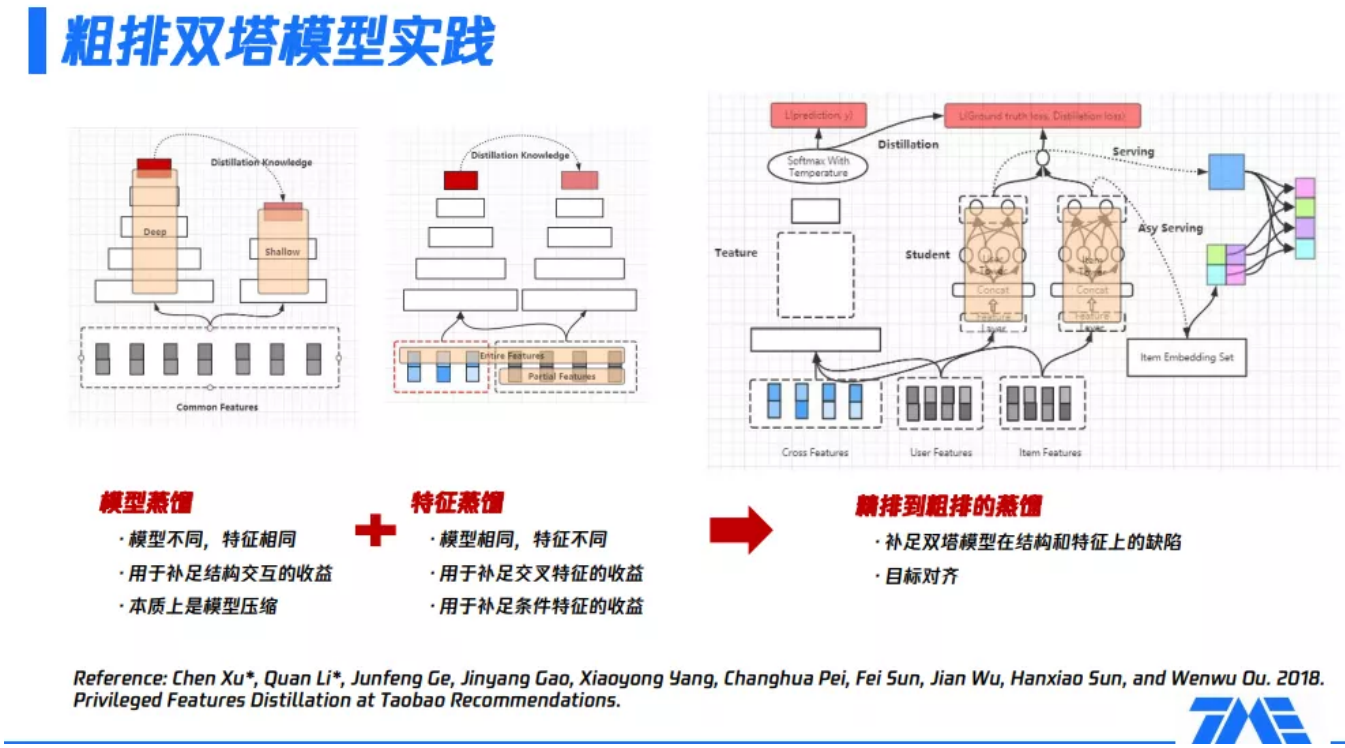
$$q_i = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)}$$

这里，引入一个超参数T控制整个softmax分布的陡峭或平滑程度，间接控制蒸馏的强度。通常，T要设置成多少没有一个定论，与实际场景相关，依赖相应场景下的一些实验设计。

- 关于loss：预训练好teacher模型后，转而关注student模型，它的最小化loss由两部分构成，一部分是尽可能拟合teacher模型的输出结果，另一部分是尽可能拟合真实样本。由于student模型的表达能力有限，直接用student模型拟合真实样本可能学不好或者学偏；teacher模型传递过来的信息对student模型做了约束和纠正，指导student模型的优化方向，进而加强了student模型的学习能力。

整个模型蒸馏的收益表现如下：

引入soft labels的概念，不再是原本的非零即一状况，需要考虑正样本有多正和负样本有多负。这看起来类似把分类问题转换成回归问题，但实质并不是。如果构建样本时，用回归的方式构建，通常会基于作品的完播率或规则组合等，人工敲定样本的回归值，这种方式过于主观，没有一个非常合理或可矫正的指标进行后续比对；而模型蒸馏的soft labels是基于teacher 模型，即由teacher 模型对真实样本进行充分训练后给出，包含更多的隐含信息且不受人为主观因素影响。



具体的，来看看我们如何做粗排模块的蒸馏，主要是分为两个大的方向：

- 模型蒸馏，主要适用于模型不同但特征相同的情况，比如，包含多个场景的框架里，某些场景对性能有要求；由于有一些额外的外部限制，使某场景无法用很复杂参数量非常大的模型，可以用一个子模

型使用全部特征。此时，模型蒸馏主要是弥补子模型缺少复杂结构或交互结构导致的部分收益损失，本质上是一种模型压缩方案。

- 特征蒸馏，主要适用于模型相同但特征不同的情况，比如，在某些场景，无法使用全量特征（如交叉特征）或部分特征必须是剪裁过的，即特征没被完全利用，存在一部分损失。这时，通过一个更大的teacher模型学习全量特征，再把学到的知识迁移到子模型，即弥补了上述的部分特征损失。

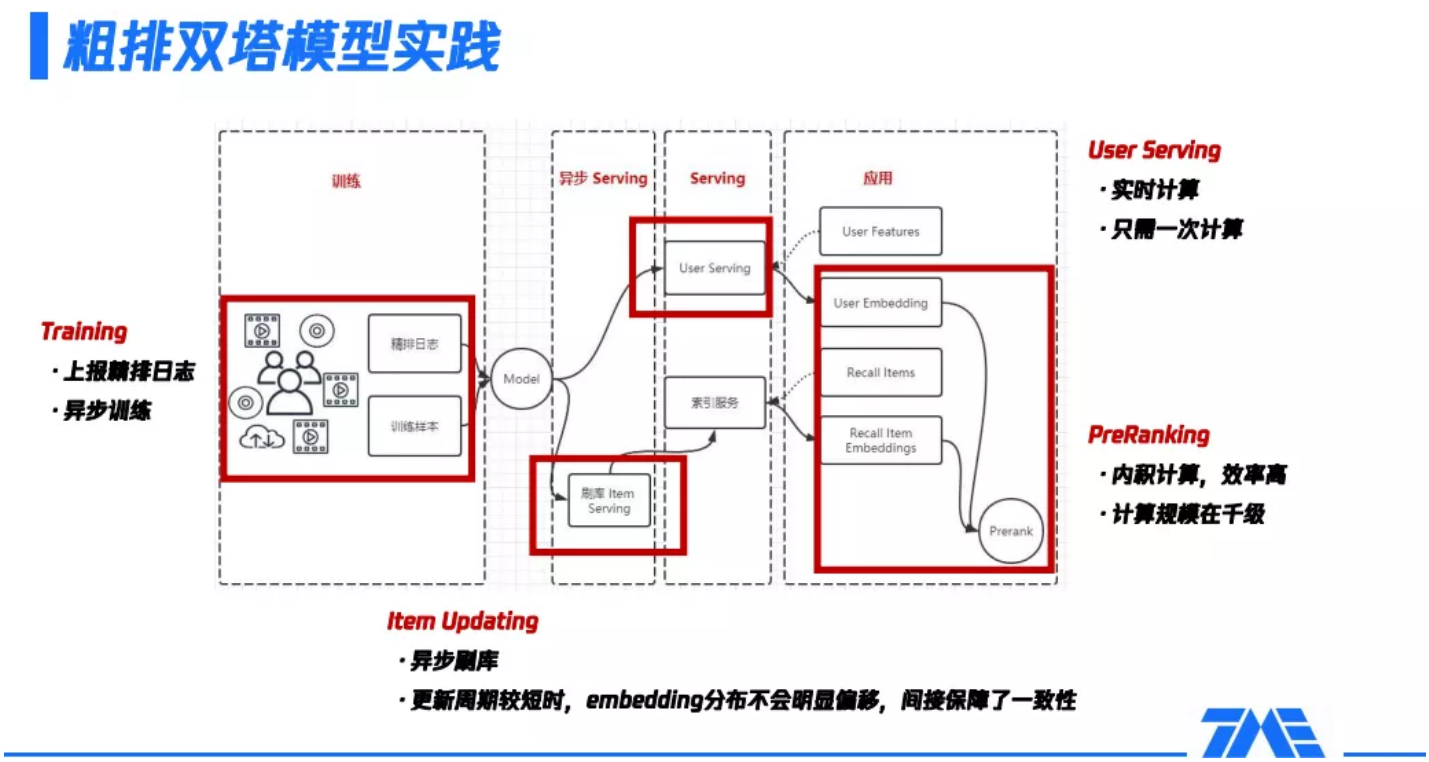
基于以上描述可知，粗排模块可以通过蒸馏的方式，补足双塔模型在结构和特征上的缺陷。接下来的问题就是如何找到一个合适的teacher模型？精排模型通常是一个被充分训练的、参数量很大、表达能力很强的模型，如果通过蒸馏精排模型获取粗排模型，那么目标的一致性和学习能力的上限都符合我们预期的。具体操作方式如上图右侧所示：

- 左侧的teacher模型是精排模型，该模型使用全量的特征，包括三大类，即user侧特征、item侧特征及它们的交叉特征；曲线框里是一些复杂的拓扑结构；最后的softmax with temperature就是整个精排模型的输出内容，后续会给到粗排模型进行蒸馏。

- 中间的粗排模型，user tower只用user features，item tower只用item features，即整体上看，模型未使用交叉特征。在user tower和item tower交互后，加上精排模型传递过来的logits信息，共同的构成了粗排模型的优化目标。整个粗排模型的优化目标，同时对真实样本和teacher信息进行了拟合。

- 右侧展示的是训练完粗排模型后，在线上产出user serving，通过user serving产出user embedding，再与item embedding做内积运算，完成整个排序的过程。上述流程即实现了粗排和精排的联动过程，并且训练完粗排模型就可以进行一个非常高效的serving，进行粗排排序。业界也有相关的一些工作，上图最下方给出了一篇阿里在这方面的论述。

3. 线上的粗排双塔模型



实践中，粗排模型训练时，依赖的精排输出结果来自上报精排日志。如果在离线重新对样本进行预估再输出，其实是对线下资源的浪费，所以通常在线上精排预估时，就把一些结果作为日志进行上报。粗排模型异步训练，产出模型提供给Serving。

Serving环节主要包括异步Serving和User Serving两部分，具体功能如下：

- 异步Serving，主要通过刷库Item Serving捕获所有的item embedding，以异步的方式反复刷库。大家经常会问的一个问题：如果模型更新了，item embedding有的是新的，有的是旧的，版本不一致怎么办？如果更新周期比较短，item embedding的分布不会发生较明显的偏移，即相邻版本比较接近，这间接保障了一致性，不需要版本是强一致性的，只要更新周期比较短就可以。

- User Serving，当用户请求推荐引擎后，获取该用户的user features，然后请求User Serving产出user embedding；同时，会拿到该用户的所有召回item集合，基于刷库结果产出的索引服务，取到所有的item embedding。由此可见，User Serving的最大优势是可以做实时计算且只需计算一次，效率非常高。并且，双塔粗排Serving阶段是做内积运算，这种高效的计算方式也使它更适合大规模的预排序场景。

4. 粗排的线上收益

整个粗排模块上线后，我们关注的两类指标：互动类指标和播放类指标，都有非常明显的正向提升，具体的指标提升幅度可以参考上图的在线实验结果。

一般，粗排模块的表现和实际场景有较大的相关性：在候选集非常大的推荐场景下，粗排到精排间的漏斗有好几个量级，这时粗排模块会带来非常显著的收益；在候选集比较小的推荐场景下，从召回到精排间的漏斗不是很大，这时粗排的收益可能就比较有限。

进而，考虑做进一步优化。

5. 还有哪些需要考虑？

首先，上述粗排蒸馏过程本质上是pointwise，但通过上报精排日志可以拿到精排模型给出的完整再推荐列表，用pairwise学习精排排出来的序，可以进一步逼近精排结果，能更多的提取精排传递出的信息。

其次，前面虽然不断地用teacher模型和student模型描述整个过程，但实际效果上student模型不一定低于teacher模型。换一个角度，student模型其实是基于teacher模型做进一步训练，所以student模型的表征能力有可能超过teacher模型。事实上，如何让student模型通过反复的蒸馏，效果超过teacher模型，在模型蒸馏领域也有许多相关方法。

最后，粗排到底是召回的延伸，还是精排的压缩跟前置？虽然召回和精排都是一个检索的过程，但二者实际侧重点还有一些不同，比如，召回在多样性上有更多的考量，精排更强调排序次序的精准性，而粗排处于这两个环节之间，如何利用粗排模块更好地平衡召回和精排？通常而言，会考虑设计多样性调节算法解决这一问题。

04

多样性调节算法设计

1. 推荐多样性的意义

多样性的概念在推荐系统里常被提到，在不同视角下，推荐多样性对应着不同的问题。

- 在系统角度下，多样性是一种popularity bias，即流行度的偏置。流量在UGC作品上的分布，体现了系统层面的多样性：一个多样性弱的系统，更像是中心化分发的，只分发非常头部、非常类似的一部分作品；而一个多样性强的系统，则是一个去中心化分发的，会更多地兼顾中长尾内容流量的供给。实际上，在推荐系统中普遍会遇到如下问题：如果没有对推荐系统做额外的干预和纠偏，不可避免地会使推荐系统往多样性弱的方向发展。从数据层面解释，有丰富数据的那部分内容会在推荐过程被反复加强，使整个推荐循环链路越缩越小，马太效应越来越严重。对于一个ugc平台，需要考量生产者或创作者的利益，而这种聚集在部分创作者身上的马太效应是我们不愿意看到的。

- 在用户角度下，多样性就是Explore&Exploit问题，对用户做兴趣的探索与聚焦。如果多样性弱，推荐的item同质化严重，都很像，那么推荐系统可能没办法发现用户的真实兴趣，因为系统可能都没给用户推荐过这类item；如果多样性强，那么用户的推荐流里的内容会很不一样，坏处可能是用户在持续消费过程的兴趣聚焦程度不同，比如用户看了五个item，明明对其中某一两个item更感兴趣，但和这一两个item相似的item的后续推送密度却跟不上，这对用户体验是有损的。如果不做额外的优化，

用户角度的多样性会越来越小，因为通常选用的pointwise模型会导致同质化的现象，比如说用户喜欢的item是乐器类的，则pointwise在每一个单点上的预估都觉得乐器是最好的，最后可能连续给用户推了5个乐器，在单点上收益最高不代表用户对整个推荐结果（ 5~10个item ）的满意度是最高的，所以这里也需要做多样性控制，提升用户的满意度。

2. 多样性控制的方案路线

在具体实现上，多样性有三个主流的技术方案：规则打散、embedding打散和DPP，下面会详细介绍：

- 基于规则打散，比如从item里抽象出发布作者、标签、伴奏等特征，基于这些特征去做一个session内的频控。这种方法的好处是易实现，缺点也非常明显，该方案本质上是进行枚举加排列组合，扩展性非常差，永远没办法枚举所有可能的情况，枚举出来的情况不一定能真实表征两个item间的相似或差异程度有多大。

- 基于embedding打散，一个连续值的方案。好处是它可以基于embedding对候选集做离散性的评估，相当于此时可以用向量化的方式表达item。缺点在于它虽然可以衡量离散性或多样性，但难以衡

量相关性，从而无法实现联合的收益评估。事实上，多样性只是我们的一个目标，相关性也很重要，如果推了很多不同的东西，但和用户不怎么相关，可能会适得其反。

- DPP概率模型，本质上是一种集合选择技术，选择出一个子集使得多样性和相关性联合建模的收益最大。直观理解，先基于多样性和相关性构建一个矩阵，该矩阵行列式的物理含义是矩阵中各向量张成的平行多面体体积的平方，这样就把问题转换成了一种可度量的方式：要想同时最大化多样性和相关性，只需要最大化平行多面体的体积。

通常，从平台生态控制的角度，类似DPP的控制算法需要贯穿整个推荐链路。在粗排和精排之后，都需要DPP环节，后续会介绍DPP算法的具体实现。

3. DPP 技术细节

DPP算法的具体实现比较复杂，这里主要介绍两个关键点：

①基于多样性和相关性构建矩阵L，该矩阵的行列式等价于最终要度量的目标。如何去定义相关性和多样性？

- 多样性(diversity)是指两个item相似不相似，如果很相似就不多样。利用两个item各自的item embedding计算内积，即表示两个item的相似度。

- 相关性(relativity)是一个候选item与当前用户的匹配程度。在不同环节的实现有差异：在粗排层的DPP，一般是基于user embedding和item embedding计算内积。在精排层的DPP，需要精排的CTR或CVR预估结果，因为这些预估结果反映了当前用户对候选item的喜爱程度。

基于上述定义的相关性和多样性就可以构建矩阵L：用户与$item_i$的相关性（偏好程度）、用户与$item_j$的相关性（偏好程度）、$item_i$与$item_j$的多样性三者乘积。通过最大化矩阵L，就可以实现相关性和多样性的联合度量。

②如何优化求矩阵L行列式的复杂度，该行列式的原始计算复杂度是三阶，线上难以支撑这样的运算性能消耗，可以通过贪婪算法把计算复杂度进一步降低至一阶。

- 先进行矩阵分解，基于分解的结果将计算复杂度降低到二阶。

- 用增量的方式更新参数，绕过求解线性方程组的部分，将复杂度进一步降低到一阶。

由此，行列式的求解过程由三阶降低到一阶，满足了线上的性能，上图最下方给出的paper就是相关方向的论述。

4. 线上收益

在线上做多样性的相关实验，我们较关注的系统和数据两部分，都有明显收益：

- 系统收益，作为一个ugc平台，会考量推荐系统分发覆盖了多少创作者、覆盖了多少作品以及内容的时效性，从上图右侧三条曲线可见，在加入多样性的控制后，三个指标都有稳步的提升。

- 数据收益，有关播放时长、关注渗透、点赞渗透相关的指标都有1~2个点的提升，这也体现了多样性是有必要的，因为它相当于对pointwise进行了简单的listwise化，形象化阐述就是，用户浏览多个session时，多样性调节避免了同质化内容扎堆。

5. 关于DPP还能做更多的是什么？

针对上述基于多样性和相关性构建矩阵L的过程，有两方面可以做进一步优化：

- 多样性本质上是定义什么样的item是相似的，用相似矩阵刻画这种相似关系。在我们K歌平台，会做一些内容的混排，比如音频跟视频的混排，如果在原始的item embedding构建出来的item similarity matrix基础上，加入一些预设的先验信息，判断什么样的item是更相似的，比如音频跟音频间更相似，视频跟视频间更相似，那么它最后的度量目标就加入了对作品类型的考量，这就间接实现了内容的混排。

- 相关性本质上是定义什么样的item更匹配当前用户，之前都是从消费者的角度去考量用户更喜欢什么样的item，而我们作为一个ugc平台，很多时候也要考量什么样的item更适合被分发，比如某item是不是更有平台的画风，更符合我们对内容分发的理解，这时，就是从平台系统或者生产者的角度去理解什么样的item更应该被优先推荐。如果已知一部分item更需要被优先推荐，在构建用户跟item间的相关性分时，可以对relativity score加调节权重进行干预，实现流量分配。比如，如果要使乐器类item比其他item有更高的权重，这时，可以调高乐器类item和用户的匹配权重。

综上可知，DPP不只是一个多样性或相关性的度量，它本身是一种调控方式，具体调控的量和业务场景相关，具有非常大的挖掘空间。

今天的分享就到这里，谢谢大家。

在文末分享、点赞、在看，给个3连击呗~

分享嘉宾：

kevinshuang

腾讯音乐 | 全民K歌推荐算法团队负责人

kevinshuang，现负责腾讯音乐-全民K歌业务的推荐团队，团队业务包括音视频推荐、直播推荐、社交推荐、推荐平台建设等。清华大学出版社《推荐系统与深度学习》一书第一作者。

fivenwu

腾讯音乐 | 应用研究

本科毕业于中山大学数学系，研究生毕业于香港科技大学CSE系，当前主要负责全民K歌的内容推荐。

社群推荐：

欢迎加入 DataFunTalk **推荐算法** 交流群，跟同行零距离交流。**识别二维码**，添加小助手微信，**入群。**

关于我们：

DataFunTalk 专注于大数据、人工智能技术应用的分享与交流。发起于2017年，在北京、上海、深圳、杭州等城市举办超过100场线下沙龙、论坛及峰会，已邀请近600位专家和学者参与分享。其公众号 DataFunTalk 累计生产原创文章300+，百万+阅读，9万+精准粉丝。

DataFunTalk

专注于大数据、人工智能技术应用的分享与交流。致力于成就百万数据科学家。定期组织技...

459篇原创内容

公众号

分享、点赞、在看，给个**3连击**呗！

收录于话题 #原创精选·76个

上一篇 · 有赞数据治理之提质降本

喜欢此内容的人还喜欢

重磅：某国产操作系统发布，称完全可替代Windows 7，由华为牵头制作！

码农扫地僧

多目标排序在快手短视频推荐中的实践

DataFunTalk

用 Python 自动玩王者荣耀，简直太秀了！

Python编程