第1章 推荐系统简介

本书是一本讲推荐算法、推荐模型的书。但是为了内容的完整性，也为了使初次接触推荐系统的同学能够快速入门，本章先向读者介绍一下推荐系统（Recommender System）。

首先，我们必须认识到，尽管推荐模型是现代推荐系统的核心，但是它也只是推荐系统的一个模块而已，并非推荐系统的全部。为了让模型能够正常运行、发挥作用，推荐系统中需要众多模块协同工作：

- 需要日志系统收集用户反馈，为推荐系统提供原始数据
- 需要大数据系统、流式计算系统，从原始数据中提取信息，将它们加工成模型需要的形式
- 需要在线学习系统及时更新模型
- 需要监控系统让我们观察模型运行得是否正常，并且能够自动报警
- 需要A/B实验系统评估模型效果，并且能够动态配置、改变推荐系统的运行方式
- ..... 同理，参与推荐系统的也并非只有算法工程师，还需要前端、后端、产品、运营、用户增长等多个团队的通力配合。

总之，一个完整的推荐系统所涉及的知识、技术，要远超推荐模型/算法，也复杂得多。有鉴于本书的读者应该以算法同行居多，作者在本书的开篇就强调以上观点，目的就在于提醒读者，在工作中不要只把目光停留在模型、算法这"一亩三分地"，一定要具备全局视角，从整个推荐系统的角度来思考问题。

举个例子，网上经常看到这样的评论，"为什么我使用xx模型没啥效果"。其实，这种"橘生淮南则为橘，生于淮北则为枳"的现象，并不罕见。越是大厂的模型，其结构越复杂，也就需要更多的数据，用更快频率更新迭代。假如某个小团队生搬硬套大厂的模型，而与之配套的基础设施却非常拉跨，比如用户反馈上报得不及时、模型还只能做到天级别的更新。模型的优势、威力自然发挥不出来，"水土不服"自然也就不奇怪了。这就好比，苏联解体后，乌克兰卖了航母，拆了轰炸机，销毁了核武器，难道是自废武功？非也，因为乌克兰人也清楚，这些大杀器只有大国才玩得起，留在自己这里只能当"白象"。

了解了推荐系统之于推荐模型的重要性之后，本章将分为如下几个部分进行介绍：

- 1.1节介绍推荐系统的缘起与意义，使读者更好地把握互联网技术的发展脉络。
- 1.2节通过一个极简版的推荐系统，描述推荐系统的运行机理，为初学者揭开它的神秘面纱。
- 1.3节从功能和数据两个方面介绍推荐系统的架构。

- 推荐、广告、搜索是互联网的三驾马车，1.4节讲述它们之间的区别与联系。

1.1 推荐系统的意义

现代推荐系统影响着我们日常生活的方方面面：

- 想购物，淘宝、京东的推荐系统向我们推荐商品
- 想了解时事资讯，今日头条的推荐系统向我们推荐新闻
- 想放松一下，抖音、快手的推荐系统向我们推荐视频，网易云音乐的推荐系统向我们推荐歌曲
- ……

推荐系统为什么如此流行？如此重要？回顾互联网的发展历史将有助于我们找到这一问题的答案。互联网早期，网络上的内容非常有限，没啥可推荐的，用户也没有这方面的需求。后来，随着Web 2.0的兴起，个人在网上创作、发布内容变得越来越容易，网络上的内容越来越丰富，开始出现"信息过载"。用户必须借助搜索引擎，过滤掉无用信息，快速找到自己需要的内容。

但搜索引擎需要用户主动输入自己的意图，这一点在很多时候是压根做不到的。

- 有时候，用户并不知道自己需要什么，就好比在乔布斯发明iPhone之前，人们压根就没有对智能手机的需求一样。很多用户就是抱着"逛一逛"、"试一试"的态度使用网站或APP，没有明确的目的与意愿，自然也就没啥可搜的。
- 有时候，有些需求、意愿，是用户自己都意识不到的。比如新用户初次使用APP时，APP会弹出一个问卷让用户选择自己感兴趣的话题。一个钢铁直男选的都是历史、军事、体育、数码这些硬核话题，但是后来，他自己可能都未曾意识到他对娱乐八卦那超乎寻常的热情。

如果网站或APP，因为用户提不出需求，就"无所事事"，显然这是对宝贵流量的巨大浪费，不利于建立用户粘性。正确的作法是，网站或APP要猜用户喜欢什么，然后将自己拥有的、用户可能喜欢的内容主动展示给用户，从而留住用户花费更多的时间与金钱。就这样，推荐系统应运而生。

总之，推荐系统的作用就是在信息过载的情况下，主动在人与信息之间建立高效的连接。有时候，现代推荐系统已经将这种"高效"发挥到了让人啧啧称奇的地步，比你的身边人更懂你，甚至比你自己更懂你。一个让人哭笑不得的案例是，在美国，曾经有一位父亲打电话给Amazon的客服，愤怒指责Amazon向自己还上高中的女儿邮寄了孕妇、婴幼儿产品的折扣券。过了一段时间，Amazon的客户关系经理做电话回访，准备向这位父亲进一步道歉，没想到这位父亲却不好意思地承认了，他的女儿确实怀孕了。

1.2 推荐系统是如何运行的?

为了搞清楚这个问题，我们假想一个最最简化的推荐系统。

和所有推荐系统一样，这个极简版的推荐系统中，也有两类角色：用户和物料。

- 用户（User）就是推荐系统要服务的对象。但是接下来，我们也会看到，用户也是推荐系统的重要贡献者。用户的一举一动，为推荐系统的"茁壮成长"提供了源源不断的养料。用户通过"大拇指投票"（划来划去）帮推荐系统分辨出内容、信息的真假优劣。
- 物料（Item），用于统称要推荐出去的信息、内容。不同场景下，物料会有不同的内涵：电商推荐中，物料就是商品；内容推荐中，物料就是一篇文章、一首歌、一个视频；社交推荐中，物料就是另一个人。

建立推荐系统的第一步是给物料打标签。比如对一个视频《*豆瓣评分9.3，最恐怖的喜剧电影--楚门的世界*》，我们给它打上"*电影*、*喜剧*、*真人秀*、*金·凯瑞*"这样的标签。打标工作既可以由人工完成（比如上传视频时由作者提供），也可以由内容理解算法根据封面、标题等自动完成。

第二步，建立倒排索引，将所有物料组织起来，如所示。倒排索引类似一个HashMap，键是标签，值是一个列表，包含着被打上这个标签的所有视频。

第三步，推荐系统接到一个用户的推荐请求。推荐系统根据从请求中提取出来的用户ID，从数据库中检索出该用户的兴趣爱好。在本例中，我们用一种非常简单的方式表示用户的兴趣爱好。假设该用户过去看过10个视频，其中7个带有"喜剧"的标签，3个带有"足球"的标签，则提取出来的该用户的兴趣爱好就是{"喜剧"：0.7，"足球"：0.3}，其中键表示用户感兴趣的标签，后面的数字表示用户对这个标签的喜好程度。

第四步，拿用户感兴趣的每个标签去倒排索引中检索。假设"喜剧"这个标签对应的倒排链中包含A、B两个视频，"足球"标签对应的倒排链中包含C、D、E两个视频。汇总起来，推荐系统为该用户找到了5个他可能感兴趣的视频。这个过程叫作"召回"（Retrieval），查寻倒排索引只是其中一种实现方式。

第五步，推荐系统会猜测用户对这5个视频的喜爱程度，再按照喜爱程度降序排列，将用户最可能喜欢的视频排在最显眼的第一位，让用户一眼就能看到。这个过程叫作"排序"（Ranking）。至于如何猜测用户的喜爱程度，在这个极简版的推荐系统中，我们不使用模型，而是拍脑袋想出一条简单规则，如公式(1-1)所示。

$$Score(u,g,v) = Like(u,g) \times Q(v) \tag{1-1}$$

- u表示发出请求的用户
- g表示用户u喜欢的一个标签
- v表示，在第四步中，根据标签g从倒排索引中提取出来的一个视频

- Like(u,g)表示用户u对标签g的喜爱程度。比如在上面的例子中，Like$(u$,"喜剧")=0.7
- Q(v)表示视频v的质量。可以由v的各种后验消费指标来表示，比如在本例中就用点击率（CTR）来表示视频的质量。
- Score(u, g, v)表示推荐系统猜测的用户u对视频v的喜爱程度。公式(1-1)虽然简单，但是还是具备一定的个性化能力，并非只是按照物料的质量排序。给5个视频都打完分后，推荐系统将它们按照打分降序排列，假设排序结果是[B,C, A, D, E]。

第六步，假设APP前端界面一次只能显示4个视频。推荐系统对排序结果进行截断，只保留前4个视频，返回给用户。

第七步，用户按照[B, C, A,D]的顺序看到了4个视频，点击并观看了视频B。用户行为，即"*用户u点击视频B，A/C/D曝光未点击*"，被记录进日志，发送给推荐系统。

第八步，推荐系统接收到了用户反馈，即"视频B还对我品味，不喜欢视频A、C和D"，并据此更新自己的"知识储备"。

- 更新用户的兴趣爱好。之前用户u点击了10个视频，其中7个是关于"喜剧"的，3个是关于"足球"的。现在推荐系统得知，用户u点击了带"喜剧"标签的视频B，他的统计数据变成了，共点击11个视频，其中8个关于"喜剧"，3个关于"足球"。下次再从数据库中提取出来的用户u的兴趣就变成了{"喜剧"：$\frac{8}{11}$，"足球"：$\frac{3}{11}$}，"喜剧"标签权重上升，"足球"标签权重下降。下次再给该用户推荐时，推荐系统会多推一些"喜剧"内容，少推一些"足球"的内容。
- 更新视频质量。视频B多了一次被点击的正向记录，相当于多了一个人背书，Q(B)因此而升高，反之A/C/D的质量分数会有所下降。根据排序公式(1-1)，未来在给别的用户推荐时，视频B更有可能得到高分，被推荐出去。

至此一次推荐就算完成了。

真实的推荐系统肯定要比以上的这个极简版的要复杂得多得多，比如：

- 真实推荐系统在排序时，不可能像公式(1-1)那样只考虑两个因素，而是要综合考虑来自用户、物料、上下文的上百个因素。包含上百个输入变量的公式，就不太可能靠人拍脑袋想出来了，此时就需要借助机器学习技术自动拟合出用户反馈（比如点击）与这上百个因素之间的关系。推荐模型从此登场，并发展壮大至今，让无数算法打工人"痛并快乐着"。
- 真实推荐系统所包含的步骤环节，也绝不只有"召回"和"排序"两个。"召回"之后，还要经过"去重"、"粗排"。"排序"之后的物料列表，也不会直接呈现给用户，还要经过"重排"再调整一次顺序。
- 细心的读者会发现上面介绍的极简版推荐系统存在一个严重缺陷。对于初次使用APP的新用户，推荐系统不了解其兴趣爱好，即Like(u,g)未知。对于初次收录的新物料，推荐系统无法判断其质量，即Q(v)未知。无论哪种情况，公式(1-1)都无法计算，这就是困扰推荐系统至今的"冷启动"问题。

- 极简版推荐系统只根据公式(1-1)计算出的单一分数排序，这在实际推荐系统中也是极其罕见的。互联网行业向来奉行的是"既要也要"，除了最常见的点击率，内容推荐还要考虑时长、转发率、关注率等指标，电商推荐更关心的是购买率。
- ……

真实推荐系统中的种种技术细节，将在接下来的各个章节中向读者详细介绍。

尽管简化、忽略了很多细节，但是"见微知著"，我们仍然能够从以上描述的极简版中了解到推荐系统的基本运行原理：

- 用户在APP内的行为越多，推荐系统对用户兴趣爱好的了解也就愈加准确（参考极简版的第8步），推荐结果也就愈加符合用户品味，用户与APP的互动也就越多，从而形成一个正向循环。所以，普通用户在使用APP时，不妨主动训练推荐系统。比如，对让你满意的推荐系统，不吝啬你的点赞、关注、转发；对令你不满意的推荐系统，也毫不犹豫地点"X"。这些强烈的信号都有助于推荐系统更加"懂你"，从而更有效、更贴心地为你提供信息。
- 推荐系统是一个体现"人人为我，我为人人"的地方。你通过划动拇指为每个物料投票，作为"义务"标注员，帮推荐系统认清各个物料的优劣。优质的内容会通过推荐系统这个"影响放大器"扩散给更多用户，为更多与你有相似兴趣爱好的人提供价值。

1.3 推荐系统架构

推荐系统是一个典型的大数据、高实时的复杂系统。推荐时，它要从用户几个月甚至更长的历史行为中提炼出他的兴趣爱好，从百万量级的候选集中筛选出匹配的物料，而这一切都要求在毫秒量级下完成，从而大大增加了实现难度。因此，我们要将推荐系统拆分成若干模块，每个模块互有取舍，也各有长短，多个模块彼此配合，取长补短，共同完成艰巨的推荐任务。要拆分成哪些模块，模块间如何配合，这就是系统架构所要研究的问题，也是本节要介绍的内容。

根据划分的角度不同，推荐系统架构又可细分为"功能架构"与"数据架构"。

1.3.1 功能架构

推荐系统的目标是在庞大的候选集中，为用户找出与他的兴趣相符的物料。为实现这一目标，理论上我们可以像如下这样操作：

- 离线训练出一个精度很高的模型F，输入一个用户u和一个物料t，输出二者之间的匹配度s，即$s = F(u, t)$。
- 在线服务时，对每个用户请求，我们将该用户与每个候选物料都过一遍模型F，得到用户与每个候选物料的匹配度，再将最匹配的若干头部物料返回即可。以上想法也只能停留在理论上，现实中是万万行不通的。原因在于大型APP有几百万、上千万的候选物料，一次请求就要进行几百万级的模型运算，时间开销太大，根本无法满足线上服务的实时性要求。

为了应对海量的候选集合，现代大型推荐系统都采用由"召回→粗排→精排→重排"四个环节组成的分级推荐模式，如图 1-1所示。在推荐链路中越靠前的环节，面对的候选集合越大，因此要采用技术较简单、精度稍逊、速度较快的算法，牺牲部分精度换速度；反之，链路靠后的环节，面对的候选集合较小，有条件采用技术较复杂、精度高、速度较慢的算法，牺牲部分速度换精度。

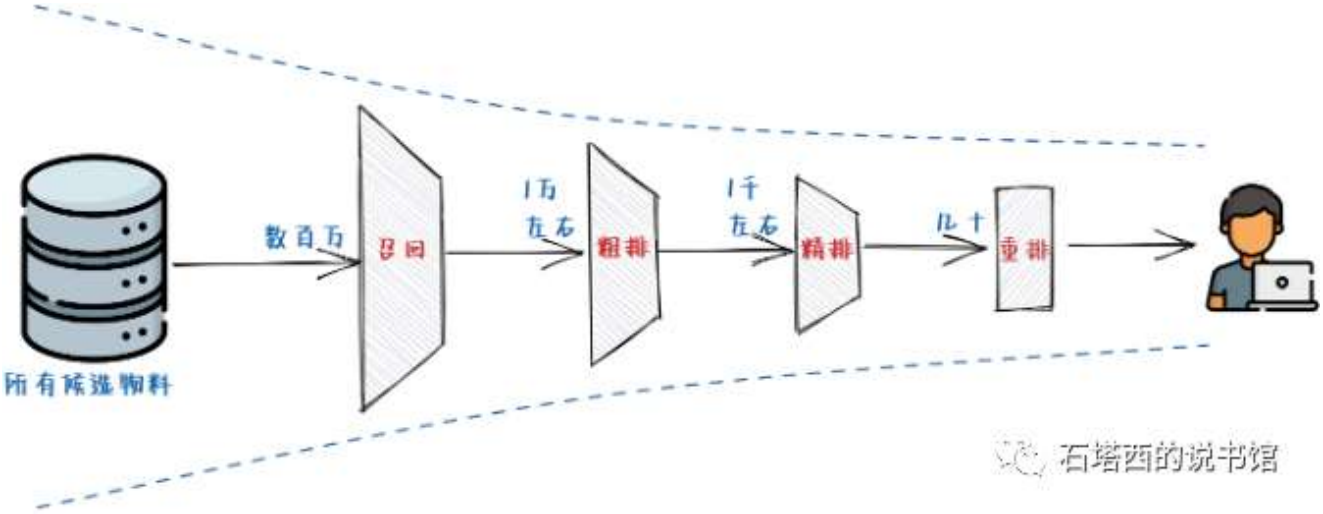

图 1-1 推荐系统的四个环节

召回

召回是推荐的第一个步骤，它面对的候选集一般要达到上百万的规模。所以召回模块的第一要务就是"快"，为此它可以牺牲一部分精度，只要能找到与用户兴趣比较匹配的物料就好，而非最匹配的，因为后者是下游排序模块的目标。

召回模块主要依赖"离线计算+在线缓存"模式来实现对上百万候选集的快速筛选。上百万的候选物料在离线就处理好，处理结果被存入数据库并建立好索引，比如极简版推荐系统中提到的"倒排索引"。线上召回时，只花费一个在索引中的检索时间，时间开销非常小。

离线处理物料时肯定不知道将来要访问的用户是谁，所以召回模型在结构与特征上，都不能出现用户信息与物料信息的交叉。这个特点限制了召回模型的表达能力，也就限制了制约了召回模型的预测精度。

为了弥补精度上的不足，召回模块一般采用多路召回的方式，以数量弥补质量。每路召回只关注用户信息或物料信息的一个侧面，比如有的只负责召回当下最火爆的内容，有的只根据用户喜爱的"标签"进行召回，

有的只负责召回与用户点击过的内容相似的其他内容，……。虽然单独一路召回的视角是片面的，但是多路召回的结果汇总起来，取长补短，查漏补缺，就能覆盖用户兴趣的方方面面。

需要特别强调的是，可能有的读者会觉得召回与下面要介绍的排序，只是速度上的差异，认为将排序模型简化一下（比如删除用户与物料的交叉结构），提升一下速度，就能胜任召回的工作。这种想法是大错特错了，召回与排序（特别是精排），在设计目的、应用场景上，存在着天壤之别：

- 召回是从一大堆物料中排除与用户兴趣八杆子打不着的，留下还比较合用户品味的。举个例子，召回好比经历过社会历练，无论哪种"不靠谱"，他都见识过。
- 精排是从一小拨儿还不错的物料中，精挑细选，优中选优，挑出对用户来说最好的物料。举个例子，精排好比还在校园中的乖学生，见过最不靠谱的人不过是借橡皮不还的同桌。

不同的应用目的与场景，就决定了召回与排序，在样本策略、优化目标、评估方式等方面，都存在显著差异。读者们切记，召回绝非小号排序，照搬精排思路去做召回，是一定会翻车的。具体技术细节将在第5章为读者详细介绍。

精排

让我们暂时先跳过粗排，先看一下精排。精排的任务是从上游层层筛选出来的千余号还比较符合用户兴趣的物料中，精挑细选出百余个最合用户品味的物料。

精排可谓是4个环节中的VIP，各互联网大厂投入的资源最多，每年研究的论文也是层出不穷。究其原因：

- 到达精排的候选集已经大大缩小了，速度已经不再是严重制约因素。这就使得精排有条件采用一些新颖、复杂的模型结构，来进一步提升预测精度，换言之，精排有"卷一卷"的空间。
- 精排在推荐链路中比较靠后，对业务目标的影响更直接有力，便于算法团队发力。

精排的设计重点是提升预测精度。所以不同于召回、粗排不允许用户信息与物料信息交叉，精排模型的发力重点就是让物料信息与用户信息更加充分地交叉，为此业界在精排引入了更多的交叉特征，使用了更复杂的交叉结构。技术细节将在第4章详细展开。

粗排

前面我们提到了，召回的精度不足，所以用数量弥补质量，倾向于召回更多物料，送往下游。而精排为了提升预测精度，不断加大模型复杂度，而牺牲了模型的吞吐能力。因为这一增一减，如果让召回直接对接精排，笨重的精排面对召回送来的越来越多的候选物料，肯定又会吃不消了。为了解决这一矛盾，重排应运而生，它接在召回后面，一般将召回的10000个结果再过滤掉9成，只保留最具潜力的1000个，再交给精排重点考察。

由此可见，粗排夹在召回与精排之间，又是一个速度与精度折衷妥协的产物，地位有些尴尬。在一些小型推荐系统中，召回的结果也不太大，干脆就放弃粗排，召回结果都喂入精排。

粗排在技术上也夹在召回与精排之间，不上不下的：

- 一方面，由于候选集规模比召回小得多，相比召回，粗排模型可以接入更多特征，使用更复杂的结构。
- 另一方面，由于候选集比精排还大得多，粗排模型比精排又简单太多。比如主流粗排模型仍然依赖"离线计算+在线缓存"模式来处理候选物料，所以仍然不能使用用户信息与物料信息交叉的特征与结构。

重排

精排时，相似内容（比如相同话题、相同标签）会被粗排模型打上相近的分数，从而在结果集中排在相近的位置。如果将这样的排序结果直接呈现给用户，用户连看几条相似内容，很容易审美疲劳，从而伤害用户体验。所以，精排结果还需要经过重排。与前面三个环节不同，重排的主要目的不是为了过滤筛选，而是为了调整精排结果的顺序，将相似内容打散，保证用户在一屏之内看到的推荐结果，丰富而多样。

虽然重排与粗排都属于"配角"，但是与粗排在小型推荐系统中可有可无的地位不同，无论系统规模，重排都是不可或缺的。只不过在小型系统中，重排比较简单，用几条启发性规则就能实现。而到了互联网大厂的推荐系统中，重排也用上了比较复杂的模型。

1.3.2 数据架构

除了上面介绍的功能架构，推荐系统中的模块还可以按照数据生产、计算、存储的不同方式进行划分，也就是推荐系统的数据架构。

举个例子，为了让模型了解每个视频的受欢迎程度，我们计算每个视频的CTR当特征。看起来这个任务稀松平常，不就是数出每个视频的曝光数、点击数，再相除不就行了？其实实践起来远没有那么简单，有三方面的难点。

- 首先，为了保证统计结果的有效性，我们需要将统计窗口拉得长一些，比如统计过去一周每个视频的曝光数、点击数。但是大型互联网系统每天产生的日志数以T计，要回溯的历史越长，涉及的计算量也就越大。
- 其次，时间上还非常紧张。线上预测时，从接到用户请求到返回推荐结果的总耗时要严格限制在百十毫秒，而且其中的大头还被好几次模型调用占据了，留给所有特征的准备时间不会超过10毫秒。
- 最后，以上所说的回溯历史指的是回溯已经存储在Hadoop分布式文件系统（Hadoop Distributed File System, HDFS）上的那部分日志，也就是所谓的"冷数据"（Cold Data）。但受HDFS只支持批量读写的性质所限，还有许多用户行为未得及组成用户日志，或者未来得及落盘在HDFS上，即所谓的"热数

据"（Hot Data）。比如我们在下午3点想获得最新的指标数据，但是HDFS上的日志才保存到下午1点，计算出来也是2小时前的过时数据，这2小时的缺口如何填补？

为了应对互联网大数据系统的复杂性，Lambda架构[1]应运而生。目前各互联网大厂的推荐系统，基本上都是按照Lambda架构或其变形搭建出来的。忽略一些称谓上的差异，Lambda的技术精髓可以由以下4句话来概括，如图1-2所示：

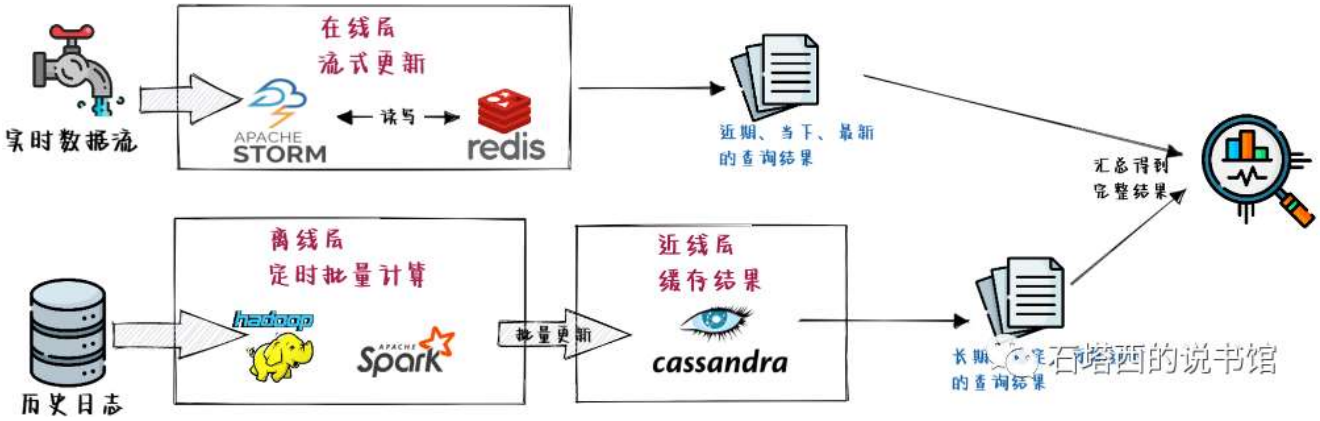

图 1-2 Lambda架构示意图

- 将数据请求拆解为分别针对冷、热数据的两个子请求。
- 针对冷数据的请求，由"离线层"批量完成计算，其结果由"近线层"缓存并提供快速查询。
- 针对热数据的请求，由"在线层"基于流式算法进行处理。
- 汇总从冷、热数据分别获得的子结果，得到最终的计算结果。

回到计算每个视频的CTR这个例子上来，我们可以将这个数据请求拆解为如下两个子任务

- 在冷数据上计算代表一个视频V长期、稳定的受欢迎程度的CTR_V^{cold}
- 在热数据上计算代表一个视频V短期、当下的受欢迎程度的CTR_V^{hot}

离线层

为了计算CTR_V^{cold}，我们启动一个小时级的定时任务，每个小时都向前回溯一周的用户行为日志，统计这个时间窗口内每个视频V的曝光数与点击数，再相除就得到了CTR_V^{cold}。为了加速，我们可以定时跑另一个小时级的批量任务，统计每个小时内每个视频的曝光、点击总数，并保存结果。有了这些小时级中间结果的加持，"统计一周的CTR"只需要汇总168个中间结果就行了，而无须从头分析统计每条用户日志，计算效率大为提高。另外，这些小时级的中间结果也能够被其他上层计算任务所复用，避免重复计算。

以上这些定时扫描日志的批量计算任务，就构成了Lambda架构中的"离线层"。技术上，这些批量计算任务可以凭借Hadoop[2]、Spark[3]、Flink[4]等大数据框架来完成，而多个任务之间的协同可以由Airflow[5]来完成。

近线层

离线层计算完毕之后，所有视频过去一周的CTR还停留在HDFS上。而HDFS是一种擅长批量读写，但随机读写效率极低的存储介质，不利于线上快速读取。为了提高查询速度，我们将离线批量计算的结果导入Cassandra、Redis这样的"键～值"（Key-Value，KV）型数据库。这些起缓存、加速访问作用的KV数据库就构成了Lambda架构中的"近线层"。

回到视频CTR这个例子中，KV数据库存储离线计算好的该视频过去一周的CTR，线上服务以每个视频的ID作为键，在其中快速检索。

在线层

前面已经讲过，离线批量计算只能处理已经落盘在HDFS上的冷数据。但是因为HDFS只支持批量读写，所以用户行为从发生到被记录在HDFS上，之间存在着小时级别的延时。这也就意味着，用户最新的行为都未能体现在离线批量计算的结果中。假设现在是下午4点，我可以根据某视频的ID，从"近线层"提取到该视频过去一周的CTR，但是这个CTR是以下午2点为终止时间向前回溯一周的统计结果，和当前时间有着2小时的缺口，已经过时了。下午3～4点这两小时内，该视频的消费情况，我们还一无所知呢。

"在线层"正是为了弥补上这块短板。这一层凭借Storm[6]、Flink[4]等流式计算框架，对接用户的行为数据流，不等数据落地，就直接对它们进行分析计算，计算结果也缓存在Redis这样支持随机读写的数据库中，方便线上查询。

回到视频CTR的这个例子，我们在Redis中以"<xx视频，xx日xx小时>"为键，存储某视频在某个时间段上的曝光数、点击数，并通过流式计算程序加以在线实时更新。由于在线层只为了弥补冷、热数据间的时延缺口，而这个缺口不过个把小时而已，所以Redis中只需要保留最近几个小时的数据，不会占用太多资源。

线上访问的时候，我们已知"近线层"中存储的是从下午2点向前回溯的结果，所以我们从"在线层"的Redis中查询到今天下午3～4点这两个小时内的曝光数与点击数，从而计算出CTR_V^{hot}。

到目前为止，对于"视频V的CTR"的这次查询，我们得到了两个子答案，代表长期受欢迎程度的CTR_V^{cold}和代表当前受欢迎程度的CTR_V^{hot}。我们可以想出一个策略，将两个数字合并成一个数字。而最常见的处理方法则是，把这两个指标都喂入模型，使模型具备不同时间维度的视角。

至此，笔者将Lambda架构要解决什么样的问题、如何解决都简单介绍完毕。毕竟本书的主题并非关于大数据架构设计，介绍过程中难免忽略了很多技术细节，请感兴趣的读者自学之。

1.4 推广搜的区别与联系

推荐、广告、搜索，简写"推广搜"，是互联网业务的三驾马车，其中推荐与搜索负责留住用户，生产流量，而广告负责将流量变现。这三驾马车之间既有区别也有联系，本节将为读者详细梳理它们之间的异同，拓宽、加深读者对互联网业务的理解与认识。

1.4.1 三驾马车的相同点

从本质上说，推广搜都是针对用户需求找到最匹配的信息。只不过用户需求的表达方式，区分了是推荐还是搜索；信息服务的对象，区分了是推搜还是广告。技术细节将在以下章节中详细介绍。

- 推广搜都遵循"先由召回模块粗筛，再由排序模块精挑细选"的功能架构。
- 数据架构上，推广搜都遵循Lambda架构。
- 因为从业务本质到系统架构都是高度相似的，很多算法、模型在"推广搜"三个领域都是通用的。一个领域发表的论文，很容易在其他两个领域复现；由于技术栈也是相同的，一个领域的工程师也很容易转到其他两个领域。
- 推广搜都需要高度的个性化。大家对个性化推荐已经习以为常了，而事实上广告对个性化的需求更高。毕竟推荐结果不符合用户兴趣，只是有损用户体验而已，而广告如果不满足用户需求的话，浪费的可是真金白银。搜索也不能仅仅满足于返回的文档的确包含了搜索关键词。比如不同用户搜索"苹果价格"，显示的结果肯定是不同的，至于显示的是水果价格还是手机价格，就取决于搜索系统对用户画像的掌握与利用。

1.4.2 推荐 vs.搜索

推荐与搜索两个场景最大的差异，在于用户表达其意图的方式不同。搜索中，用户通过输入查询语句显式表达其意图，如(1-2)所示。

$$F_{search}(t|q,u) \tag{1-2}$$

- u表示当前用户，q表示用户输入的查询语句，t表示某一个候选物料
- F_{search}表示搜索模型，衡量物料t对用户u输入的查询q的匹配程度
- 注意用户信息u也是公式的输入条件。不同用户输入相同的查询语句q，得到的结果也是不一样的，这正是个性化搜索的体现。

推荐中，用户无须显式表达其意图。推荐系统通过自己的长期观察，猜测用户意图，完成推荐，如公式(1-3)所示。

$$F_{recommend}(t|u) \tag{1-3}$$

- $F_{recommend}$表示推荐模型，衡量物料t对用户u的匹配程度
- 其他符号的意义参考公式(1-2)。

对比公式(1-2)与公式(1-3)，查询语句q是搜索中表达用户意图最重要的信息来源，在搜索模型中享受VIP待遇。比如，推荐中最重要的特征来源于用户信息与物料信息的交叉，而搜索中最重要的特征要让位于查询语句与物料信息的交叉。查询语句q在搜索模型中也处于关键位置，比如在Attention结构中，q能够用来衡量用户过去行为历史的重要性。

推荐与搜索第二个差异在于，搜索对结果的准确性要求得更加严格。这种准确性一方面体现在对多个查询条件的处理上。比如，某用户搜索"二战德国坦克"，搜索结果如公式(1-4)所示。

$$D_{search} = D(二战) \bigcap D(德国) \bigcap D(坦克) \tag{1-4}$$

- $D(二战)$代表带有"二战"标签的文章的集合，$D(德国)$与$D(坦克)$含义类似。
- D_{search}代表搜索结果，是三个标签集合的交集，即由同时包含所有标签的文章组成。

当用户点击了D_{search}中的一篇并阅读完毕，此时会触发相关推荐，给用户推荐与刚才那篇文章类似的文章。推荐结果如(1-5)所示。

$$D_{recommend} = D(二战) \bigcup D(德国) \bigcup D(坦克) \tag{1-5}$$

- $D_{recommend}$代表推荐结果，是由三个标签的文章集合的并集组成，目的是为了能够更大范围地涵盖用户兴趣。
- 其他符号含义参考公式(1-4)。

搜索结果对准确性要求更高，还体现在对扩展性的容忍程度上。

- 如果用户画像中包含"二战、德国、坦克"这三个标签，推荐系统对这些标签扩展一下，给该用户推荐一篇讲"二战苏联飞机"的文章，也是可以接受的，不会认为是一个Bug，反而有利于对用户的兴趣探索。
- 但是如果用户搜索"二战德国坦克"，用户的需求已经非常明确了，再展示"二战苏联飞机"的文章，就是不可接受的，说明搜索系统出了Bug。

1.4.3 推搜 vs.广告

首先推搜是为了留用用户来制造流量，所以要服务的目标比较简单，就是为了给用户提供最佳使用体验。而广告是为了将流量变现，所以要兼顾用户、广告主、平台三方面的利益，参与方更多、更复杂，优化起来难度更高。

推搜的目标基本上都是能够即时完成的，比如点击、完成播放等。而广告关注的目标是更深层次的"转化"。比如给用户展示一个电商APP的广告，只有用户点击广告、下载安装APP、注册、成功下单后，才算一次完完整整的"转化"。广告的转化链条越长，延时反馈的问题越严重（比如用户今天下载APP，明天才下单），最后成功转化的正样本越稀疏，建模难度越高。

推广搜都要预测点击率（CTR）、转化率（CVR）之类的指标，但对精度的要求不同。推荐与搜索对预测出来的CTR/CVR只要求"相对准确性"，即它们的预测精度能够将用户最喜欢的物料排在最前面，这就足够了。举例来说，如果用户喜爱A物料超过B物料，如果推荐模型给B物料的打分是0.8，那么推荐模型给A物料的打分是0.81还是0.9都不影响产生将A排在B前面的正确排序。

而广告则不同，由于预测出来的CTR/CVR要参与对广告费用的计算，误差一点都将带来真金白银的损失，因此广告对预测精度要求"绝对准确性"。在模型的预测结果出来之后，广告还需要对其修正、校准。

不过，毕竟制作、投放广告还存在一定技术、财力上的门槛，因此相比于推搜几十万、上百万的候选集合，广告的候选集就会小很多，从而能够简化过程，降低一些系统复杂度。比如有的广告系统就忽略了粗排环节，将召回的广告直接喂入精排。

1.5 本章小结

本章是本书的开篇第一章，在正式介绍讲解各种推荐算法、模型之前，介绍一下推荐系统。读者们应该明确，推荐算法、模型虽然发挥着举足轻重的使用，但是远非推荐系统的全部。各位算法同行应该从全局、系统的角度来看待问题、解决问题，而不要只执迷于算法、模型，"只见树木，不见森林"。

1.1节介绍了现代推荐系统的缘起与意义，我们知道推荐就是在信息过载的情况下，主动在人与信息之间建立高效的连接。

1.2节通过一个极其简化的案例，向读者介绍了推荐系统的运行机制。从这个例子中我们可以看到，一个成功的推荐系统就是要在用户与系统之间建立正向反馈。用户的行为越多，推荐系统对该用户的兴趣爱好了解得越详细，推荐结果也就更加精准，从而吸引用户留下更多行为。而且，用户行为越多，推荐系统发现的高价值的信息、内容也就越多，从而能够服务更多用户。

推荐是一项数据量庞大、计算密集、实时性要求高的任务。为此，我们需要将整个推荐系统拆分与若干模块，每个模块在速度、精度上互有取舍，需要相互配合，完成推荐任务。至于如何拆解模块、模块之间如何配合，这就是1.3节介绍的推荐系统的架构问题。

- 从功能角度划分，推荐系统可以划分为"召回→粗排→精排→重排"四个环节。前面的环节以精度换速度，后面的环节以速度换精度。
- 从数据视角来看，大多数现代推荐系统都遵循Lambda架构，即"离线层批量计算，近线层缓存结果，在线层流式更新"。

推荐、广告、搜索是互联网的三驾马车，既高度相似，在技术细节上又有显著差异。1.4节为读者梳理了这三者之间区别与联系，给对搜索、广告感兴趣的同学继续自学打下基础。

1.6 本章参考文献

[1] NATHAN MARZ. Lambda architecture[CP/OL]. https://en.wikipedia.org/wiki/Lambda_architecture.

[2] Apache Hadoop[CP/OL]. https://hadoop.apache.org/.

[3] Apache Spark[CP/OL]. https://spark.apache.org/.

[4] Apache Flink[CP/OL]. https://flink.apache.org/.

[5] Apache Airflow[CP/OL]. https://airflow.apache.org/.

[6] Apache Storm[CP/OL]. https://storm.apache.org/.