02 | Netflix推荐系统：企业级的推荐系统架构是怎样的?

2023-04-12 黄鸿波 来自北京

《手把手带你搭建推荐系统》

你好，我是黄鸿波。

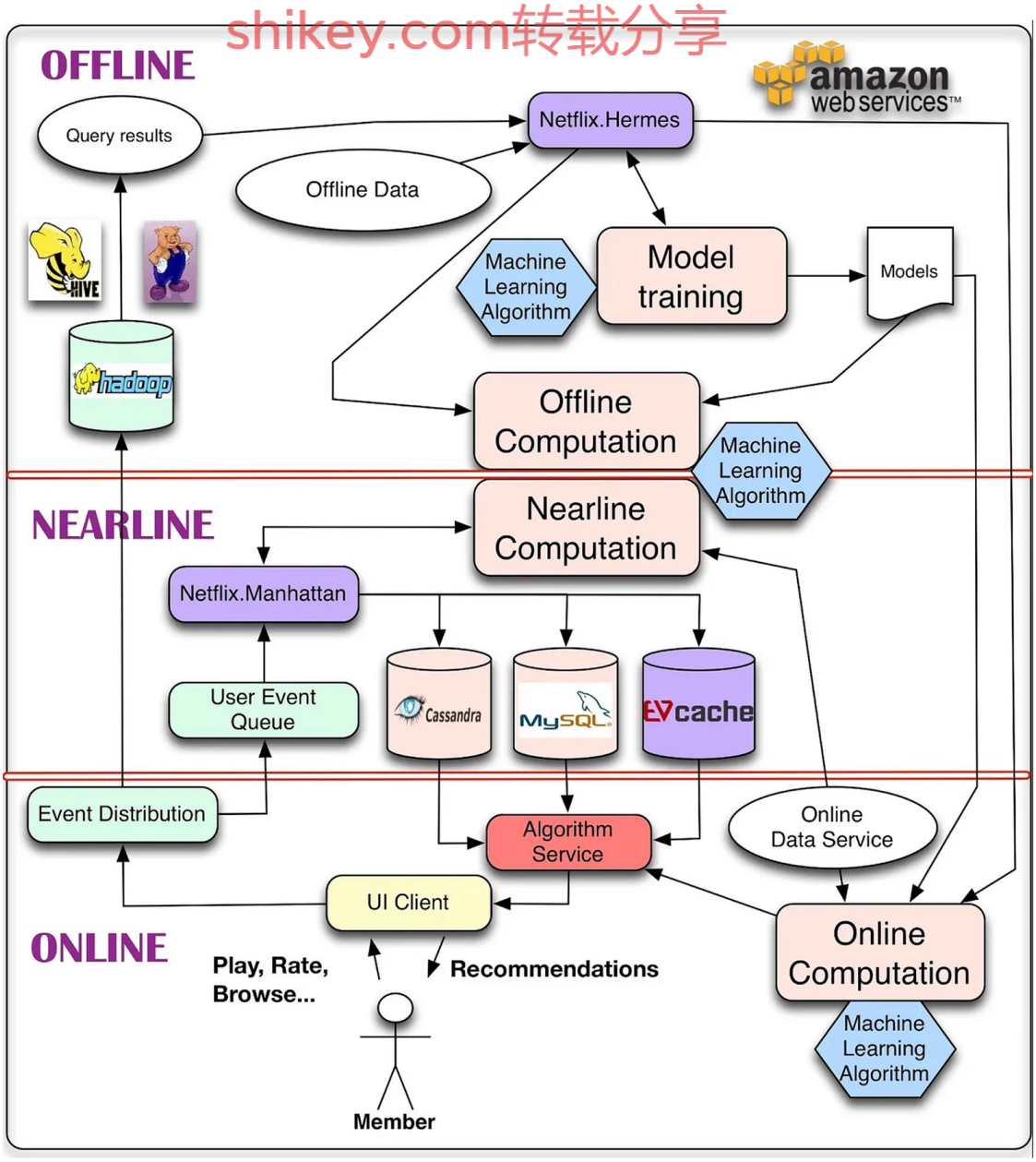
在上节课中，我们已经了解了推荐系统的运作方式，知道了它能够为企业带来什么。本节课我们就用一个实际的案例，讲解一下推荐系统具体是怎么工作的。

我们选取 Netflix 开放出来的推荐系统架构作为这节课的案例，一起来看看什么是 Netflix 系统、Netflix 系统的整体架构是怎样的。另外我还会为你讲解在线层、近似在线层和离线层之间的关系。

相信你学习完今天的内容，会对推荐系统的整体架构有更加深入的认识。

什么是 Netflix 系统?

Netflix 是一家在全球多个国家提供网络视频点播的平台，用户可以付费在 Netflix 上观看自己喜欢的节目，这个平台的网络电影营收曾排在全美国第一名。Netflix 这么高的收入，其中很大一部分原因要归功于它的推荐系统。下面这张图就是 Netflix 的推荐系统架构。



从上面这张图中我们可以发现，Netflix 的推荐系统主要分为了三个大的部分，从下到上分别为在线层（Online）、近似在线层（Nearline）和离线层（Offline）。下面我们来详细地解读这三大部分所做的工作。

离线层、在线层和近似在线层

我们先来简单地了解一下这三层的概念。

一般来讲，离线层包含的是需要离线工作的内容。比如说对数据库的内容的查询、特征工程、数据处理以及数据存储相关的内容，这些都是在离线层做的。也就是说离线层指的是不需要实时给到用户反馈的内容。在推荐系统中，最基础的召回一般就是在离线层里做的。

在线层一般包含的是那些需要实时反馈给用户的内容。例如推荐结果通常需要纳入实时变化的数据和特征，并经过快速处理后返回，这些都是在在线层完成的。对于一个用户来说，从访问 App 或者网页到得到显示结果，一般就是在一二百毫秒以内。而在这一二百毫秒的时间内，又包含着网络传输、数据查询、数据组装和界面显示等多个过程，因此，这里留给推荐系统的时间其实非常非常少，一般也就是四五十毫秒左右。

根据推荐系统呈现的内容、推荐系统的方案的不同，在线层需要做的事情也有很大的区别。

比如说，有些推荐系统的在线层只会对近似在线层或离线层的内容进行重新排列，那么这类在线层所需要消耗的时间可能就只有几毫秒。

而有些推荐系统的在线层需要实时获取用户当前的状态信息，然后将它实时地处理成模型所需要的特征，并放到相应的模型或者算法中去推理，最后再将得到的结果组装起来推荐给用户。这个时候，我们对模型和算法的实时性要求就变得非常高了。因此，这类的在线层就必须选择一些推理速度非常快的模型，避免用户长时间等待，提高用户体验。

而近似在线层是介于在线层和离线层之间的中间层，它对实时性的要求也介于两者之间。根据三层结构的名称我们可以清楚地知道，在线层的响应一般来讲都是非常实时的，它对算法和模型的速度要求非常高。而离线层的数据一般可以线下计算，对速度要求不是很高。在一般的企业级推荐系统中，离线层的内容由一个定时器定时完成，我们可以设置每周、每天或者几个小时执行一次。

而近似在线层对实时性的要求其实更贴近于在线层。一般对于一个中等规模的企业级推荐系统来说，我们会每隔 10 分钟或者 20 分钟来更新一次最近这段时间内的数据，比如用户的浏览

历史、购买记录等。得到这些数据后，我们会再将其处理成模型所需要的特征信息，从而得到模型计算的结果。

在这里需要说明的一点，近似在线层有时和在线层的功能比较类似，就是从召回集中获取数据，然后经过计算，对数据进行排序，从而得到一个粗的排序集合。只不过，相比于在线层，我们会用到更多的特征，也可能会对画像数据库或历史行为数据库进行一些数据查询和特征筛选工作，因此，我们也可以将这个过程称之为粗排序，或者精召回。

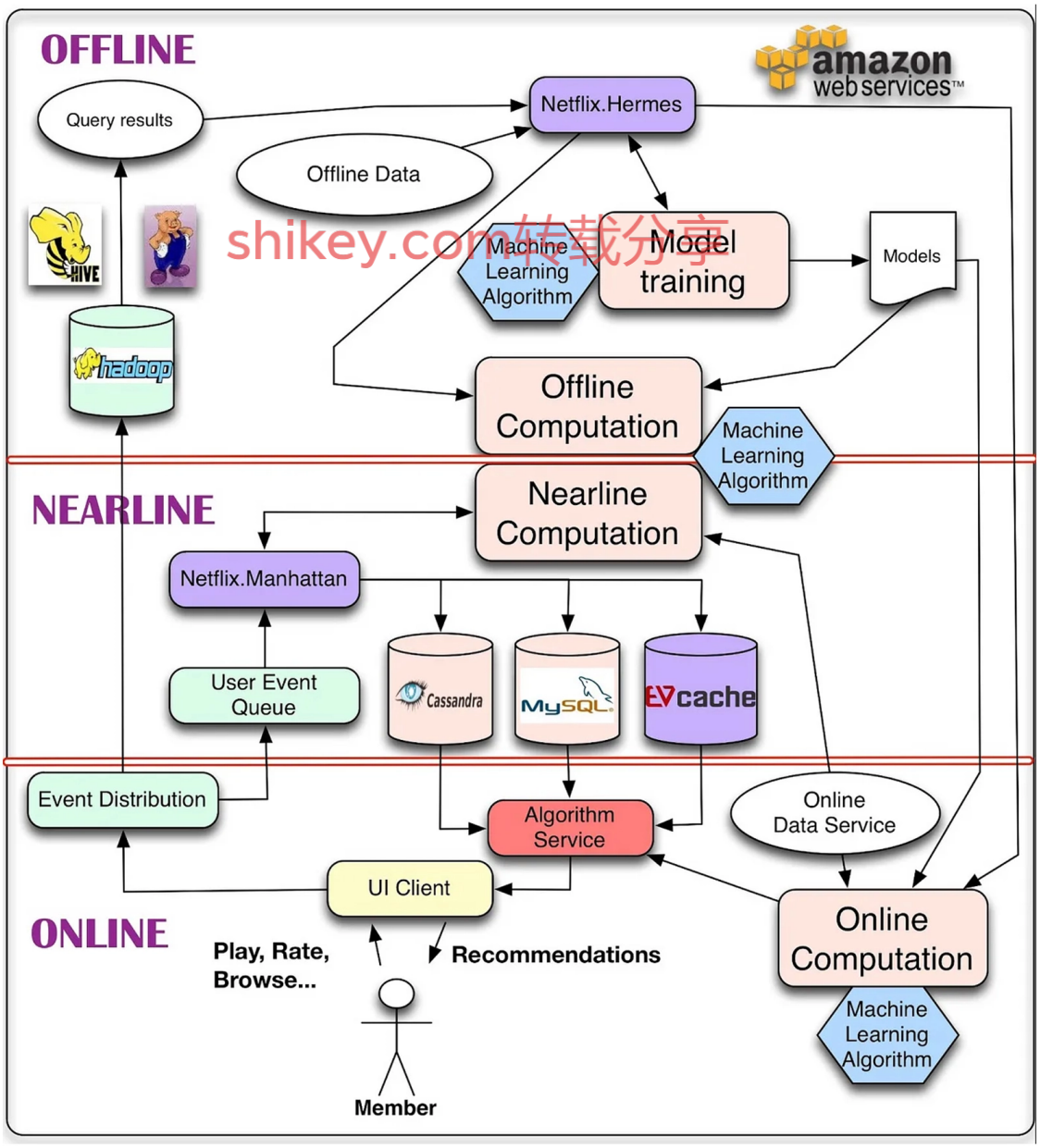
Netflix 推荐系统的工作流程

在理清了 NetFlix 推荐系统的结构后，接下来我们就看看它具体的工作流程是怎样的。这里还是要提醒一下，**虽然三层结构分工不同，但任何一个系统都不会将系统中的某一层单独分离出来，层与层之间都是相互联系并协调工作。**所以，我们接下来的讲解也会把它们看做一个整体。

根据之前的学习我们知道，推荐系统首先要得到召回集，这个召回集是对用户行为的记录，并且召回集一般是离线得到的。

那么在 Netflix 的推荐模型中，是如何得到召回集的呢？

我们还是拿一开始的这个架构图进行讲解。首先在线层针对用户有两个箭头，一个是接收来自推荐系统所推荐的内容，另一个是向离线层提交用户的行为数据，这些用户数据一般会被存放在数据库中。例如，Netflix 的推荐系统就将用户数据存放在了 Hadoop 中。



由于 Hadoop 是一个分布式的文件操作系统，具有高效性、高可靠性和分布式存储等特点，因此很多推荐系统都将其作为首选的离线数据存储方案。当数据存储到 Hadoop 之后，推荐系统中的离线模型就会从 Hadoop 中提取推荐系统所需要的数据信息，然后再通过 Hive 对数据进行一系列的处理和清洗，使其变成可以直接拿来训练的数据。

在 Netflix 中，离线层也是首先将在线层提交的数据存储到了 Hadoop 中，然后通过 Hive 将 Hadoop 中的数据提取出来（图中的 Query results 部分）。

我们可以看到，在 Netflix 的离线层有一个叫做 Netflix.Hermes 的模块，这个模块类似于实时管道数据流，主要负责实现发布 - 订阅机制，这个机制与 Apache Kafka 的作用有异曲同工之处。但是要注意的是，**Netflix.Hermes 模块并不是一个消息 / 事件队列系统，这一点是它和 Apache Kafka 最大的区别。**

Netflix.Hermes 模块的信息传递主要包括四个部分。

首先，通过 Hive 查询到的数据结果会放入 Netflix.Hermes 模块。

其次，一些其他的离线数据也会通过 Netflix.Hermes 模块进行数据的传递。

另外，Netflix.Hermes 模块会与离线模型训练双向互动。这里之所以是一个双向的过程，是因为我们在训练离线模型的时候，需要从 Netflix.Hermes 模块中获取数据，当模型训练完成之后，推荐系统还需要将模型训练的结果返存到 Netflix.Hermes 模块中。因此，只有这样的双向传递设计才能满足整体的需求。

最后，在模型训练完成之后会产生一个模型文件（如图中的 Models），拿到这个模型文件后我们就可以用它进行推理预测了。

使用这个模型文件的方法一般分为两种：一种是直接使用用户的 ID 进行预测，得到召回层的召回集信息，我们在后面几章会讲到的协同过滤算法就是这类操作的代表；另外一种方法是要在文件内加入其他的特征数据，然后再进行离线计算，xDeepFM 或者 YoutubeDNN 就是这一类模型的代表。

根据上图的数据传输流来看，Netflix 的推荐系统应该属于第二种，即对模型文件和用户特征进行组合，将组合后的数据进行离线计算。

在这个组合的过程中，我们需要两部分的输入：模型本身、特征和标签。特征和标签部分使用 Netflix.Hermes 模块进行数据流传递，通过 Netflix.Hermes 模块可以拿到两部分数据：特征数据和预测的标签。特征数据里的特征包含了用户的特征和商品的特征，特征的好坏对于推荐算法来说有着很大的影响。而预测的标签会与模型文件结合后输入离线计算的算法中，得到召回集数据（这里使用的是机器学习算法）。

近似在线层的处理过程介于离线层和在线层之间，它主要是利用流式计算得到中间结果，这是为了响应用户事件而进行的中间层计算，在这里执行的是近似在线计算，不需要实时完成。

在近似在线层计算的过程中，需要从在线层和离线层分别获取数据。在线层主要是获取用户实时的行为数据，包括播放量、评分和浏览量等。离线层获取的是用户的画像和上一步召回的结果。

接下来我们要对数据进行计算和整合：将数据输入在线层算法中进行计算，得到给用户的待推荐列表，再根据业务的需求存入相应的数据库中。这就完成了近似在线层的业务逻辑。

等到近似在线层处理完成后，会将结果输入到在线层。而在线层更多地是做了一个排序工作，它把离线特征数据和从近似在线层得到的待推荐列表组合起来，得到用户最感兴趣的列表，然后推荐给用户。这部分一般会涉及点击率预估（CTR,Click-Through Rate）这个概念。所谓点击率预估，就是根据用户的特征来估计用户对于哪个商品的点击率更高，然后取 TopN 作为预估的结果，最终推荐给用户。

一般而言，点击率预估是实时的，这样可以根据用户行为尽快做出响应，提高推荐的有效率。这一步也有下面两种处理思路。

第一种思路，完全实时地推荐商品，根据业务需要一次给用户推荐 N 个商品。例如，一个列表推荐 10 个商品，每次计算都取 Top10。

第二种思路，先通过算法预估出用户可能点击的 TopN，这里的 N 可能是 50，也可能是 100。当用户有刷新操作时，再按顺序取相应的内容推荐给用户。

这两种方案都是实际生产中常用的方案，具体使用哪一种，要根据业务场景、算法的复杂度及服务器的负载能力决定。

可以看出，整个推荐过程都需要层与层之间的协同配合，这样才能得到一个比较好的推荐效果，Netflix 在整体的设计里也对此做了充分的考量。至此，我们对 Netflix 推荐系统的整体架构有了一个相对详细的了解，在后面的章节中，我们会根据实际的业务需要，用一些公开的数据集来实现里面每一个类似的小的部分，最终完成一整套推荐系统。

总结

这节课到这里也就接近尾声了，我来给你总结一下。学完这节课，你应该了解和掌握以下几个方面的内容。

我们需要知道推荐系统在企业中的整体架构是什么样子的，在线层、近似在线层、离线层这几个层之间的关系是怎么样的。

在推荐系统中，我们一般将召回集放在离线层进行处理，这类的处理一般不需要实时进行，可以根据业务场景的需要，选择是每几个小时还是每几天来进行一次离线层的模型训练和推理。

而近似在线层处理的是最近一段时间内的用户数据，比如 5 分钟或者半个小时内的数据。近似在线层会把这些数据处理成相应的特征，再放到模型中进行推理计算，得到推理结果。

在线层一般都是实时处理，它对于模型或者算法的效率要求非常高。一般来讲，它需要在几十毫秒内完成所有的模型计算，并将计算出来的结果返回给上层服务，然后由上层服务进行数据的组装，最后将内容呈现给用户。

学完这节课之后，你还应该知道一个完整的企业级推荐系统，还会涉及各种对于评分的计算、点击率的预估等，这些内容都是一个完整的企业级推荐系统必不可少的。只有将这节课的内容学会、学通，我们才能够设计出一个优秀的企业级推荐系统。

课后题

学完这节课的内容，给你留两道思考题。

1. 你认为在线层、近似在线层和离线层都适合存储什么样的数据，怎么存储比较好？
2. 我们将最后计算出的结果给到后端服务后，后端服务会经过哪些阶段，做什么样的事情，将最终的结果推荐给用户？

欢迎你在评论区留下你的观点，我们一起交流讨论，下节课见。

精选留言 (3)

一五一十

2023-04-27 来自北京

1. 在线层对响应时间要求较高，因此用 redis 这种内存数据结构较好；近似在线层可以用 Mysql 这种存储非海量的数据；离线曾用 hadoop hive 这些存储海量数据。
2. 后端会对召回数据做一些精细的筛选和重排后返回给用户 topN，然后将用户的操作行为再收集给到离线层。
自己理解，如果不对还请老师和同学们批评指正哈。

作者回复: 答：同学你好，你的理解是完全正确的。另外，近似在线层也可以使用Redis这类的数据库，主要是看你的近似在线层对实时性的要求有多高。

peter

2023-04-12 来自北京

请教老师几个问题：
Q1：Netflix推荐系统“开放”具体指什么？是指开源吗？或者是类似工具一类的可以拿来直接用？或者SDK？或者只是提供架构图一类的原理性解释？
Q2：近似在线层，既然是“粗排序”，怎么会是“精召回”呢？应该是“粗召回”才更合理啊。
Q3：在线层依赖近似在线层的话，怎么保证在线层的实时性？
文中提到“等到近似在线层处理完成后，会将结果输入到在线层。而在线层更多地是做了一个排序工作，它把离线特征数据和从近似在线层得到的待推荐列表组合起来”，从这句话看，在线层是依赖近似在线层。如果有依赖，近似在线层是分钟级，而在线层是50毫秒，就无法保证实时性了啊。
Q4：NetFlix的三层架构是通用的吗？
Q5：推荐系统对大数据依赖大吗？大数据框架出现之前有推荐系统吗？

作者回复: 同学你好，我来回答你的这几个问题：
1、课程中所说的开放，指的是他们的论文，而不是开源，就是Netflix已经发表了论文，详细地说明了这一套是怎么做的；
2、我是这么理解的，近似在线层是在召回层之后，实际上是对召回层的数据根据用户的信息再一次做筛选，因此可以理解为是更精细的召回，但是相对于排序来说，它的内容还是太多，所以是粗排序；
3、在线层依赖于近似在线层的结果，但是近似在线层实际上是对用户最近几分钟的历史行为来做更

新，我们主要是取近似在线层更新的结果，所以，近似在线层的计算耗时并不影响在线层取内容；
4、Netflix的三层架构基本上现在的推荐系统都是这么来做的，相对通用；
5、推荐系统对大数据的依赖可大可小，主要看用户和内容的规模，因此，大数据框架出现之前，实际上也是可以利用其它的存储方式来获取数据进行计算，而且在大数据出现之前，一般业务量也不会特别大，所以，我认为，这个是时代发展的结果。

GAC·DU
2023-04-12 来自北京

在线层存储用户实时轨迹数据，近似在线层存储业务逻辑数据，离线层存储用户历史轨迹数据。离线数据存储在Hadoop的分布式文件系统中，其余两者存储在Kafka消息队列中。后端服务系统接收数据后，做进一步的关联查询，把最终结果推给前端服务。不知道思考的对不对？思考的过程中觉得虽然做了分层，对执行性能上做了优化，但是耦合度太高，尤其是近似实时层，承上启下，重要程度不言自明，能否可以做一个降级的方案，做到即使某一层挂了也能做到最小程度的推荐？

作者回复: 同学你好，你的这个理解稍微有一点点偏差。首先在线层没有做存储功能，只是去收集实时数据，然后给到近似在线层和离线层。离线层存储的有2个，一个是历史轨迹，一个是本地信息，比如用户画像、用户数据、内容数据、内容画像等，kafka的消息队列，不做存储，你可以理解为一个管道，kafka只是这根管子，数据是在这里面排队的。
这些数据会经历离线的召回、近似在线层的排序和在线层的重排序，然后给到用户。
如果需要做降级的话，我们一般可以把近似在线层的值同时存入到两种数据库，比如MongoDB和Redis，如果Redis取不到就从MongoDB中取。层与层之间也是这样。