14 | 融会贯通：SparrowRecSys中的电影相似推荐功能是如何实现的？

你好，我是王喆。

课程进行到这里，推荐系统架构的大部分知识点，包括特征工程、Embedding 模型，到推荐服务的搭建，线上推荐过程的实现，我们都已经学习并且实践过了。如果你坚持跟着我一起学下来的话，可以说已经是“武功小成”了。

为了帮你巩固所学，今天，我就带你从头到尾地实现一个完整的推荐功能，**相似电影推荐**，来帮助你打通推荐系统的“任督二脉”。

“清点技能库”，看看我们已有的知识储备有哪些

在开始实现相似电影推荐功能之前，我想先带着你一起清点一下自己的技能库。我喜欢把推荐的过程比喻成做菜的过程，接下来，我就按照做菜的 4 个关键步骤，带你回顾一下前面学过的重点知识。

第一步，准备食材。 准备食材的过程就是我们准备推荐所需特征的过程。在特征工程篇中，我们不仅学会了怎么挑选“食材”，怎么处理“食材”，而且还实践了“备菜”的高级技能 Embedding 技术。具体来说就是，我们能够利用物品序列数据，通过 Item2vec 方法训练出 Embedding，也能够使用 Deep Walk 和 Node2vec 把图结构数据生成 Graph Embedding。

总的来说，因为 Embedding 技术的本质就是利用了物品之间的相关性，所以 Embedding 是做好“相似推荐”这盘菜的关键。

第二步，食材下锅。 备好了菜，在正式开炒之前，我们肯定要把食材下锅。在推荐系统中“食材下锅”的过程有两个，一是把线上推荐所用的特征存储到数据库中，在之前的课程中我们已经实践过使用 Redis 作为特征数据库的方法，另一个是把模型部署到模型服务模块，我们也已讲过了预训练 Embedding，Embedding 加轻量级线上模型，TensorFlow Serving 等多种模型服务方式，这节课我们将采用预训练 Embedding 的方式进行模型服务。

第三步，做菜技术。 “做菜的技术”说的是推荐服务器线上推荐的整个流程是否合理。那回到推荐系统中就是指，召回层要快速准确，模型排序部分要精确。这些具体的实现都影响着最终的推荐效果。

对于召回层来说，我们已经学过单策略召回、多路召回和基于 Embedding 的召回。对于排序来说，我们会主要利用 Embedding 相似度来排序，后续我们还会学习基于多种推荐模型的排序。

最后是菜品上桌的过程 ，也就是把推荐的结果呈现给用户的过程。这节课，我会带你一起实现这个过程。提前“剧透”一下，在 SparrowRecsys 中，我们会先利用 JavaScript 异步请求推荐服务 API 获取推荐结果，再利用 JavaScript+HTML 把结果展现给用户”。因为，这一部分内容不是推荐系统的重点，所以我们这里只要做到界面清爽、逻辑清晰就可以了。

相信到这里，各位“大厨”已经准备好了所要用到的技能，下面就让我们一起来实现 **SparrowRecSys 中的相似电影推荐功能吧！**

如何实现相似电影推荐功能？

在正式开始相似电影推荐功能之前，我们先来看看我总结的 SparrowRecsys 相似电影推荐功能的详细技术架构图。细心的你可能

已经发现了，这个架构图就是 SparrowRecsys 架构的精简版。因为我们还没有学习深度学习推荐模型和模型评估的相关知识，所以把重点聚焦在已经学过的知识上就可以了。

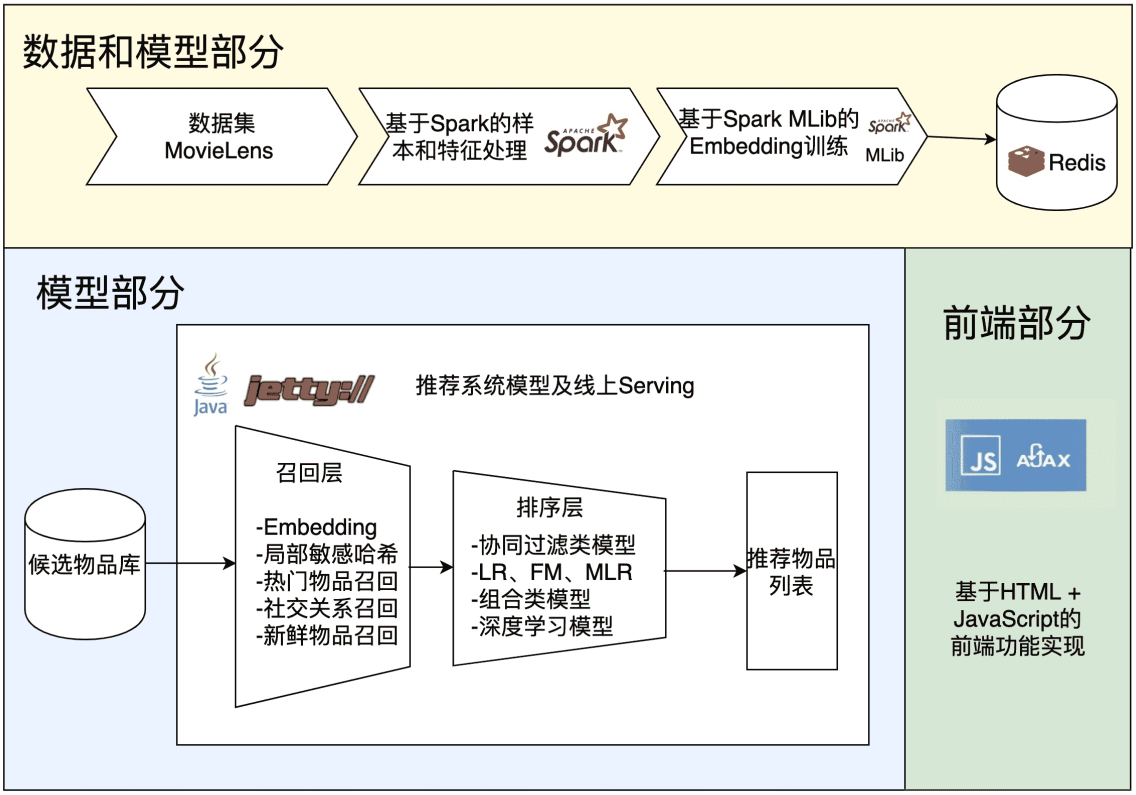


图1 Sparrow Recsys 相似电影推荐功能的技术架构图

接下来，我就结合这个技术架构图，带你一步步地实现其中的每一个模块。并且，我还会给你讲解一些项目中没有实现的其他业界主流方法，如果你还学有余力，希望你能抓住这个机会，来扩展一下自己的知识面。

1. 数据和模型部分

数据和模型部分的实现，其实和我们第 8 节课讲的 Embedding 的实战思路是一样的，我们可以选用 Item2vec、Deep Walk 等不同的 Embedding 方法，来生成物品 Embedding 向量。考虑到大数据条件下，数据处理与训练的一致性，在 SparrowRecsys 中，我们会采用 Spark 进行数据处理，同时选择 Spark MLlib 进行 Embedding 的训练。这部分内容的代码，你可以参考项目中的 _com.wzhe.sparrowrecsys.offline.spark.embedding.__Embedding_ 对象，它定义了所有项目中用到的 Embedding 方法。

对于一些比较复杂的 Embedding 方案，比如特征种类很多，网络结构也更多样化的 Embedding 模型，业界也多采用 Spark 进行原始数据处理，生成训练样本后交由 TensorFlow、PyTorch 训练的方案。

但是不论训练平台是怎样的，Embedding 方法的产出都是一致的，就是物品 ID 对应的 Embedding 向量。那为了方便线上服务使用，我们还需要在生成 Embedding 后，把它们存入某个高可用的数据库。SparrowRecsys 选择了最主流的内存数据库 Redis 作为实现方案，这一部分的具体实现，你可以参照 com.wzhe.sparrowrecsys.offline.spark.embedding.Embedding 对象中 trainItem2vec 函数的 Redis 存储操作。当然，业界也会使用 Cassandra+ 缓存，RocksDB 等不同的存储方案来实现 Embedding 向量的高效读取，但我们现阶段只要学会 Redis 存储和读取操作就够用了。

到这里，Redis 成为了连接线下和线上的关键节点，那我们的线上服务部分又是怎么利用 Redis 中的 Embedding 数据进行相似电影推荐的呢？

2. 线上服务部分

线上服务部分是直接接收并处理用户推荐请求的部分，从架构图的最左边到最右边，我们可以看到三个主要步骤：候选物品库的建立、召回层的实现、排序层的实现。我们逐个来讲一讲。

首先是候选物品库的建立。SparrowRecsys 中候选物品库的建立采用了非常简单的方式，就是直接把 MovieLens 数据集中的物品数据载入到内存中。但对于业界比较复杂的推荐业务来说，候选集的选取往往是有很多条件的， 比如物品可不可用，有没有过期，有没有其他过滤条件等等，所以，工业级推荐系统往往会通过比较复杂的 SQL 查询，或者 API 查询来获取候选集。

第二步是召回层的实现。我们在11 课曾经详细学习了召回层的技术，这里终于可以学以致用了。因为物品的 Embedding 向量已经在离线生成，所以我们可以自然而然的使用 Embedding 召回的方法来完成召回层的实现。同时，SparrowRecsys 也实现了基于物品 metadata（元信息）的多路召回方法，具体的实现你可以参照 com.wzhe.sparrowrecsys.online.recprocess.SimilarMovieProcess 类中的 multipleRetrievalCandidates 函数和 retrievalCandidatesByEmbedding 函数。

第三步是排序层的实现。根据 Embedding 相似度来进行“相似物品推荐”，是深度学习推荐系统最主流的解决方案，所以在 SparrowRecsys 中，我们当然也是先根据召回层过滤出候选集，再从 Redis 中取出相应的 Embedding 向量，然后计算目标物品和候选物品之间的相似度，最后进行排序就可以了。

这里“相似度”的定义是多样的，可以是余弦相似度，也可以是内积相似度，还可以根据你训练 Embedding 时定义的不同相似度指标来确定。因为在 Word2vec 中，相似度的定义是内积相似度，所以, 这里我们也采用内积作为相似度的计算方法。同样，具体的实现，你可以参照 com.wzhe.sparrowrecsys.online.recprocess.SimilarMovieProcess 类中的 ranker 函数。

经历了这三个主要的线上服务步骤，SparrowRecsys 就可以向用户返回推荐列表了。所以接下来，我们要解决的问题就是，怎么把这些结果通过前端页面展示给用户。

3. 前端部分

SparrowRecsys 的前端部分采用了最简单的 HTML+AJAX 请求的方式。AJAX 的全称是 Asynchronous JavaScript and XML，异步 JavaScript 和 XML 请求。它指的是不刷新整体页面，用 JavaScript 异步请求服务器端，更新页面中部分元素的技术。当前流行的 JavaScript 前端框架 React、Vue 等等也大多是基于 AJAX 来进行数据交互的。

但前端毕竟不是我们课程的重点，你知道我在上面提到的基本原理就可以了。如果你已经在本地的 6010 端口运行起了 SparrowRecsys，那直接点击这个链接：http://localhost:6010/movie.html?movieId=589 ，就可以看到电影《终结者 2》的详情页面和相似电影推荐结果了（如图 2）。

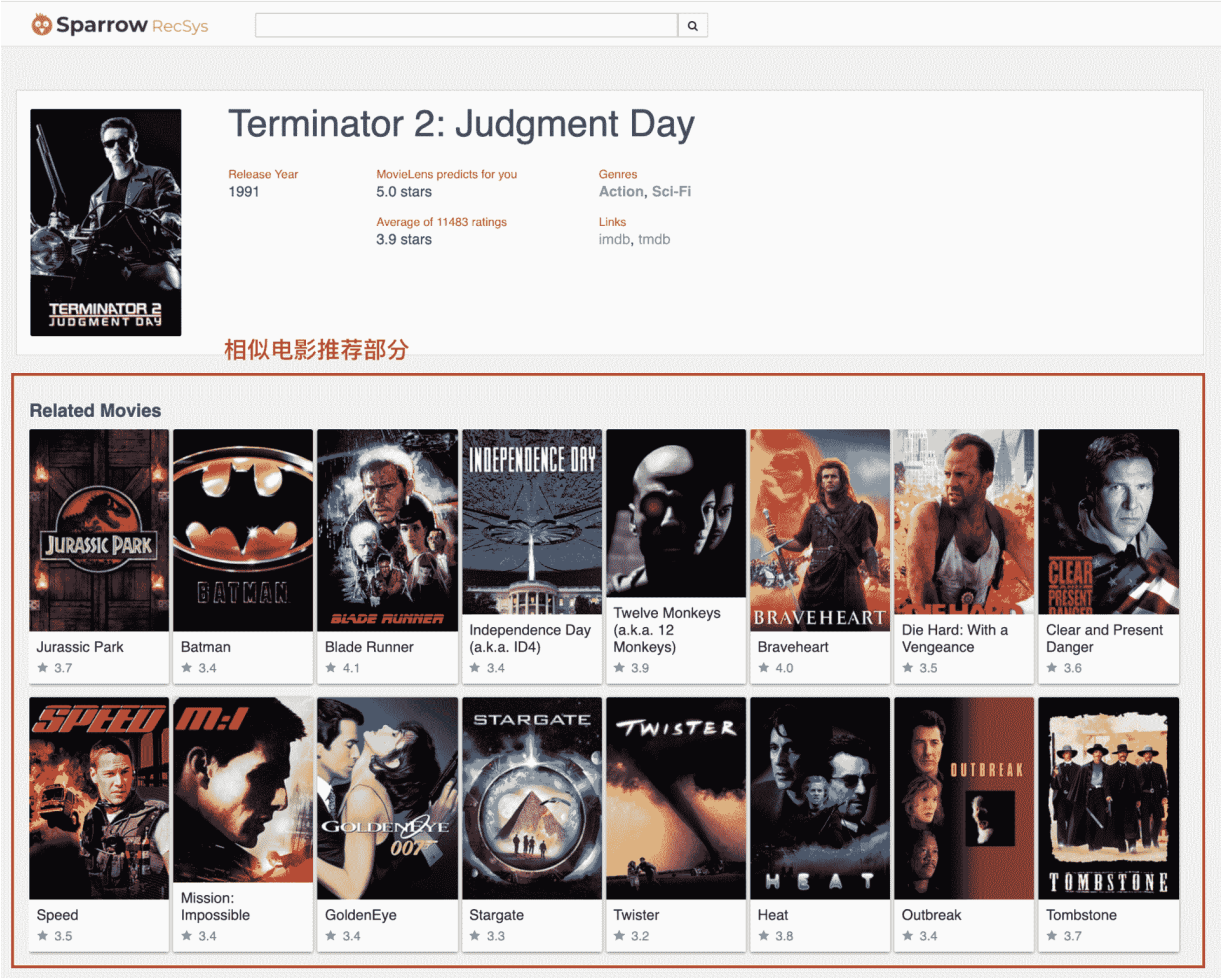


图2 终结者2的相似电影推荐结果

相似电影推荐的结果和初步分析

到这里，我相信你已经串联起来了 SparrowRecsys 相似电影推荐的所有实现，看到了推荐结果。那么问题来了，推荐结果的好坏到底是如何判断的呢？关于这个问题，我们也会在后面的“模型评估篇”中进行系统性的学习。不过，这里我也想先跟你聊聊这个话题，让你对它有一个大体认识，这对你建立后续的模型评估体系是非常有帮助的。

首先提醒你的是，SparrowRecsys 开源项目中自带的 MovieLens 数据集是经过我采样后的缩小集，所以基于这个数据集训练出的模型的

准确性和稳定性是比较低的。如果你有兴趣的话可以去MovieLens 官网选择 **MovieLens 20M Dataset** 下载并重新训练，相信会得到更准确的推荐结果。

其次，针对相似物品推荐这个推荐场景，我们其实很难找到一个统一的衡量标准。比如，你能说出《功夫熊猫》这部电影是跟《玩具总动员》更相近，还是跟《飞屋环游记》更相近吗？好在，工程师们还是总结出了一些有效的评估方法。这里，我挑出了 3 个最常用的来给你讲讲。

方法一：人肉测试（SpotCheck）。 在一种 Embedding 结果新鲜出炉的时候，你作为创造它们的工程师，应该第一时间做一个抽样测试，看一看基于 Embedding 的相似推荐结果是不是符合你自己的常识。比如说，我在 Embedding 训练完之后，随便在 SparrowRecsys 中翻了翻，看到了两个页面，一个是儿童电影《Free Willy》（《人鱼童话》）的相似电影推荐页面（图 3 左），另一个是著名动画电影《Toy Story》（《玩具总动员》）的相似电影推荐页面（图 3 右）。

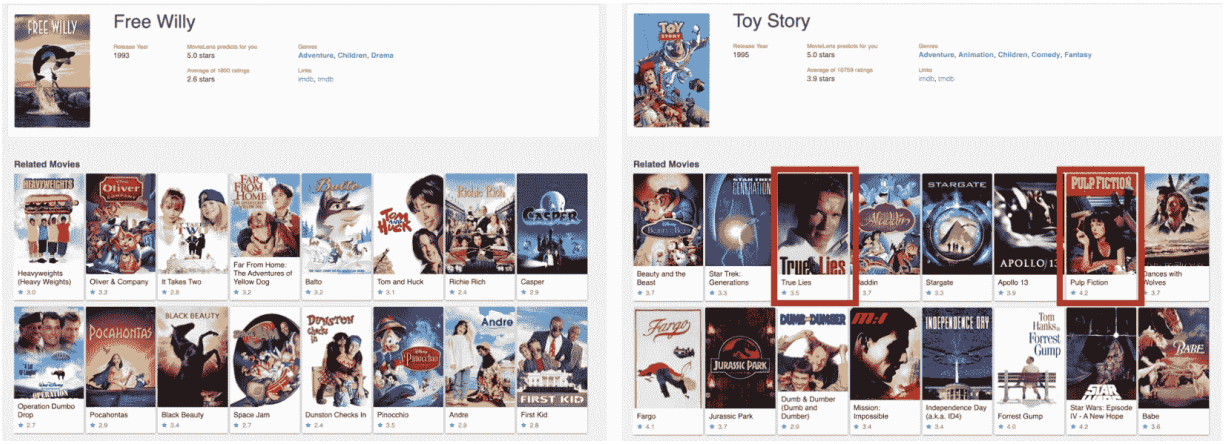


图3 随机测试

直观上来看，《Free Willy》的推荐结果就非常不错，因为你可以看到相似电影中都是适合儿童看的，甚至这些电影都和动物相关。但是《玩具总动员》就不一样了，它的相似电影里不仅有动画片，还有《真实的谎言》（《True Lies》）、《阿甘正传》这类明显偏成人的电影。这明显不是一个非常好的推荐结果。

为什么会出现这样的结果呢？我们来做一个推测。事实上，《玩具总动员》本身是一部非常流行的电影，跟它近似的也都是类似《真实的谎言》、《阿甘正传》这类很热门的电影。这就说明了一个问题，热门电影其实很容易跟其他大部分电影产生相似性，因为它们会出现在大多数用户的评分序列中。

针对这个问题，其实仅利用基于用户行为序列的 Embedding 方法是很难解决的。这需要我们引入更多内容型特征进行有针对性的改进，比如电影类型、海报风格，或者在训练中有意减少热门电影的样本权重，增大冷门电影的样本权重等等。总的来说，遇到推荐结果不合理的情况，我们需要做更多的调查研究，发掘这些结果出现的真实原因，才能找到改进方向。

方法二：指定 Ground truth（可以理解为标准答案）。 虽然我们说，相似影片的 Ground truth 因人而异。但如果只是为了进行初步评估，我们也可以指定一些比较权威的验证集。比如，对于相似影片来说，我们可以利用 IMDB 的 more like this 的结果去做验证我们的相似电影结果。当然要补充说明的是，要注意有些 Ground truth 数据集的可用范围，不能随意在商业用途中使用未经许可的数据集。

方法三：利用商业指标进行评估。 既然相似影片比较难以直接衡量，那我们不如换一个角度，来思考一下做相似影片这个功能的目的是什么。对于一个商业网站来说，无非是提高点击率，播放量等等。因

此，我们完全可以跃过评估相似度这样一个过程，直接去评估它的终极商业指标。

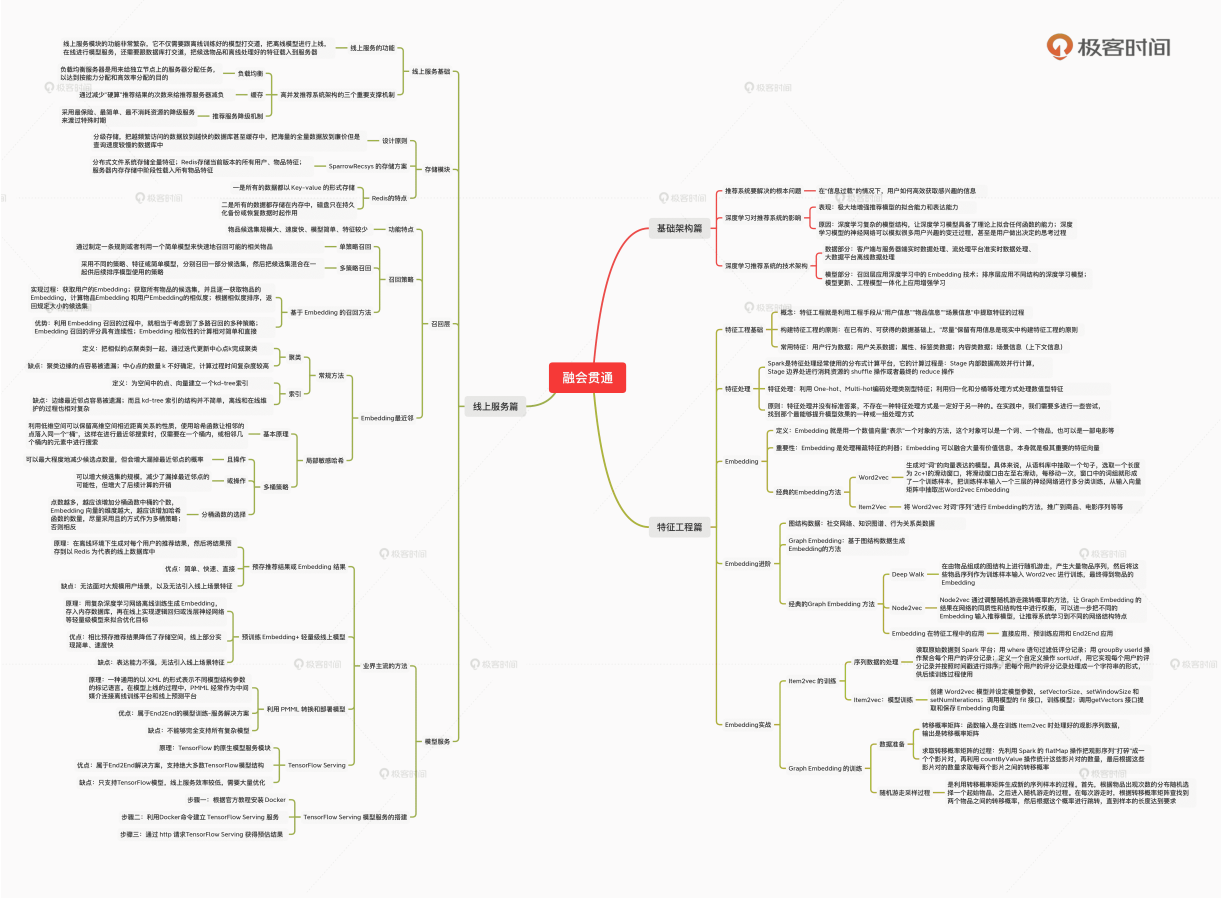
举个例子，我们可以通过上线一个新的相似电影模型，让相似电影这个功能模块的点击率提高，假设提高了 5%，那这就是一个成功的模型改进。至于相似电影到底有没有那么“相似”，我们反而不用那么纠结了。

小结

这节课，我们使用 Embedding 方法准备好了食材，使用 Redis 把食材下锅，做菜的步骤稍微复杂一点，分为建立候选集、实现召回层、实现排序层这 3 个步骤。最后我们用 HTML+Ajax 的方式把相似电影推荐这盘菜呈现出来。

既然有做菜的过程，当然也有品菜的阶段。针对相似物品推荐这一常见的功能，我们可以使用人肉测试、Ground truth 和商业指标评估这三种方法对得到的结果进行评估。也希望你能够在实际的业务场景中活学活用，用评估结果指导模型的下一步改进。

我希望，通过这节课的总结和实战，能让你融会贯通的厘清我们学过的知识。所以我把你需要掌握的重要知识点，总结在了一张图里，你可以利用它复习巩固。



好了，那到这里，我们线上服务篇的内容就全部结束了。通过这一篇的学习，我相信你已经清楚了推荐系统的全部技术架构，以及深度学习核心技术 Embedding 的运用方法。

但我要说的是，盛宴还未开始，下一篇我们将进入深度推荐模型的学习和实践。我曾经说过，深度推荐模型是深度学习推荐系统这个王冠上的明珠，正是它对推荐模型的革命，让深度学习的浪潮席卷推荐系统领域。希望你再接再厉，让我们一起把这颗明珠摘下吧！

课后思考

刚才我说到，《玩具总动员》的相似电影推荐结果并不好，我认为可能是因为热门电影的头部效应造成的。你认同这一观点吗？你觉得还

有其他可能的原因吗？如果让你去做一些 Embedding 方法上的改进，你还有什么好的想法吗？

欢迎把你的成果和优化想法分享到留言区，也欢迎你能把这节课转发出去，让更多人从我们的实践中受益，我们下节课见！