33｜技术权衡：解决方案这么多，哪个最合适？

你好，我是王喆。

在实际的工作中，我们经常会面临一些技术上的抉择。比如说，在设计推荐系统的时候，我们是应该用模型 A 还是用模型 B，是用 TensorFlow 还是用 PyTorch，是用 Redis 还是用 EVCache 呢？从理论上来说，其实选择哪个方案都可以，但在工程落地中，不同的方案往往对系统整体的开销，整个团队的工作量，甚至最终的推荐效果都有着非常大的影响。

我想这也是很多算法工程师的困惑：在工程落地环节，解决方案这么多，我们到底该选哪个？

今天，我们就一起来探讨一下技术权衡的问题，看看能不能在理论知识和工程落地之间找到一条最优的路径。

工程师职责的本质

“工程”和“理论”之间的权衡是所有工程师都要考虑的问题，对这个问题的思考方式决定了你具备的是“工程思维”还是“研究思维”，抑或是“学生思维”。推荐系统是一个工程性极强，以技术落地为首要目标的领域，因此，“工程思维”对推荐系统工程师是最重要的。

事实上，无论是算法工程师，还是研发工程师，甚至是设计电动汽车、神舟飞船、长征火箭的工程师，他们的职责都相同，那就是**在现有实际条件的制约下，以工程完成和技术落地为目标，寻找并实现最**

优的解决方案。这里面有一个词最关键，那就是“制约”。我们该怎么理解这个制约呢？

比如说，在机器学习的理论层面，一切事情都是理想化的，就像模型结构是精确定义的，训练过程是严格推导的，但在实际条件下，一个模型的训练过程会受算力条件、数据质量、工程时间限制等多重条件的制约。这也是我们不得不面临诸多选择和权衡的原因。

再比如说，我有些同学在航天领域工作，最近趁着嫦娥五号任务圆满完成的热点，我咨询了他们一个问题，说“咱们国家什么时候能实现载人登月啊？”他们这帮航天工程师的回答也很严谨，说“制约咱们国家载人登月的主要是火箭的运力问题，现在的长征五号火箭，近地轨道运载能力是 25 吨，美国阿波罗登月计划使用的土星五号火箭的轨道运载能力是 140 吨，整整差了 5 倍多。”

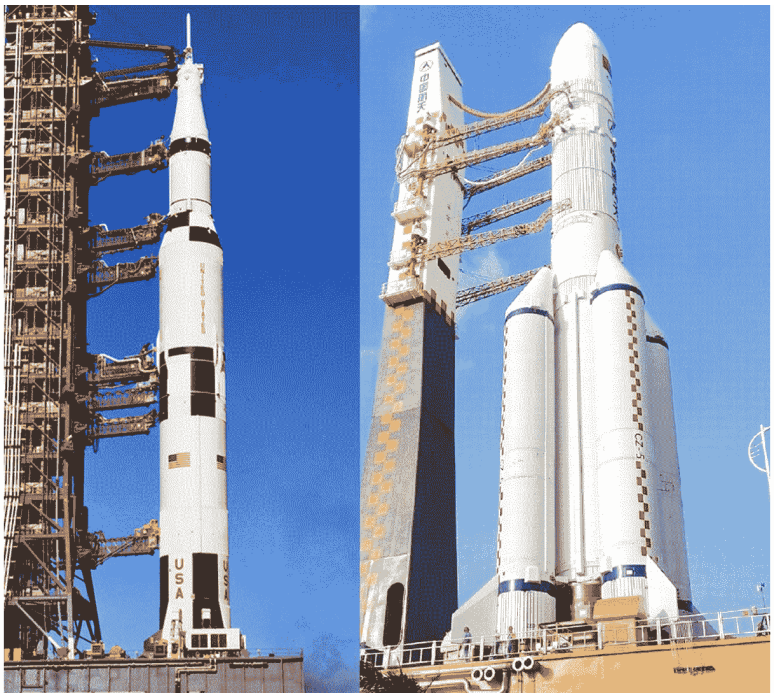

图1 土星五号和长征5号

你看，所有的工程师都在受客观条件的制约，那怎么办呢？他们又讲了，要么就是一步步来，先攻克运载火箭技术，再像阿波罗计划一样，一发火箭把登月舱、返回舱、航天员等等一起送到月球；要么就是用现有的长征五号，发射四到五枚火箭，在太空中完成登月舱、返回舱、载人飞船的组装。

第一个计划的优势是一次性搞定，但周期长，需要等待火箭技术的攻克。第二个计划的优势是现在就可以着手开始做，但因为要发射四到五枚火箭，整个任务的容错率低，失败的风险大。

所以你看，航天工作者们发射火箭也成天抱怨运力不够，就像我们训练模型总说算力不够一样。那怎么办，日子就不过了吗？当然不是，我们工程师就是要在这样的制约条件下，又快又好地解决问题、完成任务，这就是你的职责所在。

那推荐系统中“现有实际条件的制约”都有什么呢？我们经常会遇到这 3 种：

1. 软硬件环境的制约
2. 研发周期的制约
3. 实际业务逻辑和应用场景的制约

正是因为有这些制约的存在，一名工程师不可能像学术界的研究者一样不断尝试新的技术，做更多探索性的创新，也正是因为工程师永远以“技术落地”为目标，而不是炫耀自己的新模型、新技术是否走在业界前沿，所以在前沿理论和工程现实之间做权衡是一名工程师应该具有的基本素质。

那这个技术权衡具体该怎么做呢？下面，我就借用三个实际的案例帮助你体会一下。

Redis 容量和模型上线方式之间的权衡

对线上推荐系统来说，为了进行在线服务，需要将特征和模型参数存储到线上数据库或者线上服务器内存中。为了保证这两部分数据的实时性，很多公司采用内存数据库的方式实现，就像我们在第 10 讲中讲的一样，Redis 这类内存数据库自然是主流的选择。

但 Redis 需要占用大量内存资源，而内存资源相比存储资源和计算资源又比较稀缺和昂贵的资源。因此，无论是 AWS（Amazon Web Services，亚马逊网络服务平台）、阿里云，还是自建数据中心，实现 Redis 模块的成本都比较高，自然 Redis 的容量就成了制约推荐系统模型上线方式的关键因素。

在这个制约因素下，我们要考虑两方面的事情：

1. 模型的参数规模要尽量小。特别是对深度学习推荐系统而言，模型的参数量级较传统模型有了几个量级的提升，所以我们更应该着重考虑模型的参数规模；
2. 因为要考虑线上预估延迟和特征存储空间有限的情况，所以线上预估所需的特征数量不能无限制地增加，要根据重要性做一定的删减。

因此，在实际上线推荐系统的时候，我们必然需要进行一定的取舍：**舍弃一些次要的要素，关注主要的矛盾**。具体怎么做呢？这里，我结合自己的经验，把整个取舍的思考过程做了梳理，一共可以分成四步，你可以作为参考。

首先，对于千万量级甚至更高量级的特征向量维度（理论上模型参数的数量级也在千万量级）来说，因为线上服务很难支持这种级别的数

据量，所以我们在上线模型时关注模型的稀疏性和复杂度，通过舍弃一定的模型预测准确度来换取上线的速度和资源消耗。

明确了工程权衡的目标，我们就要思考怎么提高模型的稀疏性，降低模型的复杂度。提高稀疏性的常见方法是，加入 L1 正则化项，或者采用 FTRL 这类稀疏性强的训练方法让大部分参数归零，从而减少模型体积。

对于降低复杂度，我们可以通过减少神经网络的层数，以及每层内神经元的数量来实现。当然，这个方法只有在不明显降低模型效果的前提下，才是可行的工程策略。

在明确了多种备选方案之后，如果还是无法确定哪种技术效果更佳，我们就需要实现所有备选方案以及方案间的各种组合，进行离线和线上的效果测试。

最后，我们要根据测试数据确定最终的技术途径、技术方案，完成最终上线。

以上就是模型侧的“瘦身”方法，在线特征的“瘦身”方法当然也可以采用同样的思路。首先采用“主成分分析”等方法进行特征筛选，在不显著降低模型效果的前提下减少所用的特征。针对不好取舍的特征，进行离线评估和线上 A/B 测试，最终达到工程上可以接受的水平。

研发周期和平台迁移之间的权衡

除了硬件条件的限制，研发周期的制约同样是不可忽视的因素。这就需要工程师对于项目有整体的把控，以及对研发周期有预估。在产品迭代日益迅速的互联网领域，没人愿意成为拖累整个团队的最慢的一

个环节。接下来，我就以一个平台迁移的例子，来给你讲一讲如何在研发周期的制约下完成技术决策。

比如，公司希望把机器学习平台从 Spark MLlib 整体迁移到 TensorFlow 上，毫无疑问，这是顺应深度学习浪潮的技术决策，是非常正确的决定。但由于 Spark 平台自身的特性，它的编程语言、模型训练方式都和 TensorFlow 有很大的差别，因此整个迁移必然要经历一个较长的研发周期。

我就经历过很多次公司产品和技术平台的大规模升级，很常见的情况是，在保证平台升级正常进行的同时，我们还需要兼顾日常的开发进度，去实现一些新的产品需求。这就是工程师需要做出权衡的时候了，也是最考验工程师架构能力的时候。

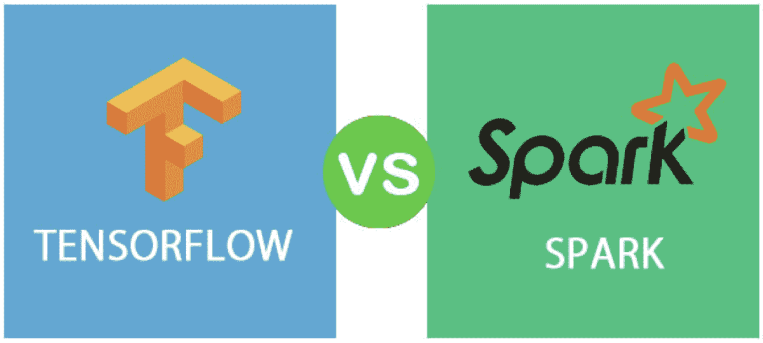

图2 TensorFlow vs Spark

在这样的情况下，一般来说有 2 种可行的技术方案：

1. 集中团队的力量完成 Spark 到 TensorFlow 的迁移，在新平台上进行新模型和新功能的研发。
2. 让团队一部分成员利用成熟稳定的 Spark 平台，快速满足产品需求，为 TensorFlow 的迁移、调试、试运行留足充分的时间。同时，让另一部分成员全力完成 TensorFlow 的相关工作，尽量保证新平台的成熟和可靠。

不过，单纯从技术角度考虑，既然团队已经决定迁移到 TensorFlow 平台了，理论上就没必要再花时间利用 Spark 平台研发新模型，否则到时候还要进行二次开发。但是，再成熟的平台也需要整个团队磨合调试较长时间，绝不可能刚迁移 TensorFlow 就让它支持重要的业务逻辑。而且，技术平台的升级换代，应该作为技术团队的内部项目，最好对其他团队是透明的，不应该成为减缓对业务支持的直接理由。

因此，从工程进度和风险角度考虑，第 2 个技术途径更符合工程实际的需求。

冷启动等业务逻辑对推荐模型的制约

最后一类制约是来自业务逻辑，或者说应用场景的，最常见的如物品冷启动问题，这类业务问题往往制约了模型的更新和应用的方式。

之前，我们讲了很多种生成物品 Embedding 的方法，但在任何公司业务中，物品都不是一成不变的，就像视频网站要不断添加新视频，电商网站会不断加入新的商品。这个时候，新物品的 Embedding 该如何生成呢？

对于 Item2vec、DeepWalk 这类基于物品节点的 Embedding 模型来说，数据中必须包含新物品的 ID，才能够生成它相应的 Embedding。这就要求我们必须加快模型的更新速度，让模型尽快学

习新物品的数据。这就是业务场景的特点在倒逼我们改进模型的实时性。这个时候，我们就又要思考改进 Embedding 模型实时性的方法了。

这里，我总结出了下面三个备选方案。

方案一：从模型训练 pipeline 的各个环节入手，看看哪些环节可以优化。比如，我们可以优化 Spark 数据预处理的过程，加快预处理速度，优化 Embedding 上线的过程，让 Embedding 从产生到上线的过程更紧凑。

方案二：从模型的复杂度入手，看一看能否在不显著伤及效果的前提下，通过降低模型的复杂度来降低训练时间，进而加快模型更新的速度。

方案三：跳出 End2End 训练 Embedding 模型的限制，看看能不能通过其他策略性的手段来生成临时性的新物品 Embedding。

其中，前两个方案需要我们从细节入手来优化工程上的实现，而第三个则要求我们有更灵活的处理思路。这一点也是我想和你多聊一聊的。

我们在改进推荐系统的时候，不能总陷进机器学习的固定思维里，认为一定要用一些机器学习模型或者深度学习网络去解决问题，我把这样的思维叫做“技术洁癖”。我们应该清楚的是工程师的第一任务是解决问题，而不是搭一个精致好看的技术玩具，所以在推荐模型的基础上，用一些策略性的手段来修补一些边界情况，“bad case”是可以去尝试的。

回到这个物品 Embedding 冷启动的例子上，策略性地生成新物品的 Embedding 就是很好的补充方式，房屋短租行业的巨头 Airbnb 就给我们提供了一个很好的参考例子。

在 Airbnb 平台上，如果一个新的出租屋发布了，但模型还没有学到它的 Embedding，Airbnb 的推荐系统会怎么做呢？它会通过三个步骤生成新出租屋的 Embedding。

第一步根据新房屋的属性，比如租金价格、房屋类型、面积、几房几厅等特征，来找到和它相似的一批房屋；第二步，在这些相似房屋中找到离它最近的三个；第三步，通过平均这三个最近房屋的 Embedding 来生成新房屋的 Embedding。

就像图 3 展示的这样，颜色相近的点代表着 Embedding 非常接近的房屋，如果一个新房屋落在了图中的某个位置，Airbnb 就可以根据它的地理位置和属性，找到相近的房屋并且给它涂上相似的颜色。

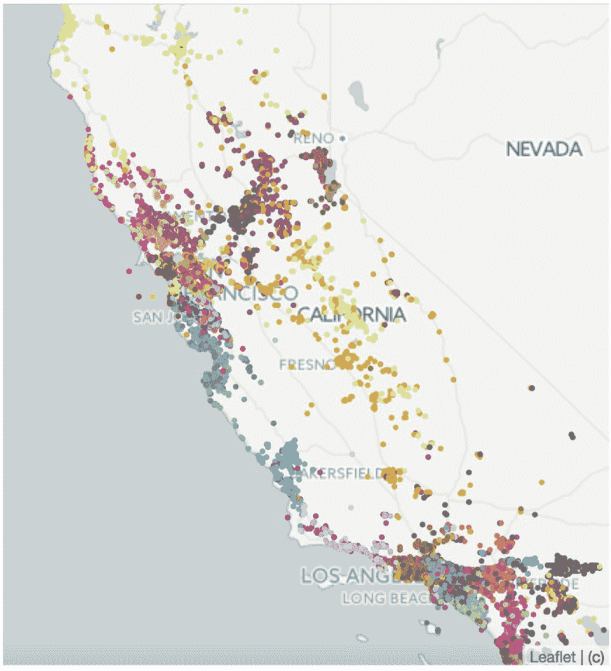

图3 Airbnb房屋聚类的例子

所以我们经常会看到，一些推荐系统中的补充策略往往可以高效地解决一些推荐模型无法解决的棘手问题。

当然，在我们日常工作中技术间的权衡无时无刻不在发生，小到一个变量取什么名字好，一行代码应该怎么写，大到一个平台应该怎么重构，一个模型应该如何构建。我们这节课的例子当然无法囊括所有的情况，只是希望能帮助你建立起正确的进行技术权衡的思路，做到抓住主矛盾点，做出有利于主要目标的取舍。

小结

这节课，我们通过三个例子一起探讨了工程落地中技术权衡的问题，希望能帮助你进一步加深了对工程师思维本质的理解。

事实上，对任何领域来说，工程师思维的本质都是“**在现有实际条件的制约下，以工程完成和技术落地为目标，寻找并实现最优的解决方案**”。

在推荐系统领域，典型的制约条件有三类：“软硬件环境的制约”，“研发周期的制约”，以及“实际业务逻辑和应用场景的制约”。在这些制约条件下，我认为的技术方案间的权衡之道，就是“抓住主矛盾点，列出备选方案，通过分析对比，作出有利于主要目标的取舍。”

课后思考

这节课，我们提到了物品冷启动问题的解决方案，你觉得基于Embedding，还有哪些好的冷启动解决方案呢？如果让你来解决的话，在实践中你会作出哪些取舍，倾向于哪种选择呢？

期待在留言区看到你的分享和思考，我们下节课见！